High Level Programming Framework for FPGAs in the Data Center

Oren Segal and Martin Margala
Department of Electrical and Computer Engineering
University of Massachusetts Lowell
Lowell, MA
{oren_segal,martin_margala}@uml.edu

Sai Rahul Chalamalasetti and Mitch Wright
Silicon Design Lab
HP Servers
Houston, TX
{sairahul.chalamalasetti,mitch.wright}@hp.com

Abstract—Heterogeneous computing offers a promising solution for energy efficient computing in the data center. FPGA based heterogeneous computing is an especially promising direction since it allows for the creation of custom hardware solutions for data centric parallel applications. One of the main issues delaying wide spread adoption of FPGAs as mainstream high performance computing devices is the difficulty in programming them. OpenCL was meant to address the difficulties and the non-uniformity related to programming heterogeneous devices, unfortunately because of its complexity it sets the bar high for many software programmers, preventing them from directly benefiting from the computing power and energy efficiency that OpenCL and heterogeneous computing have to offer. This work presents an effort to bridge the gap by extending an existing Java programming framework (APARAPI), based on OpenCL, so that it can be used to program FPGAs at a high level of abstraction and increased ease of programmability. We run several real world algorithms to assess the performance of the APARAPI framework on both a low end and a high end system. On the low end and high systems respectively we find up to 78-80 percent power reduction and 4.8X-5.3X speed increase running NBody simulation, as well as up to 65-80 percent power reduction and 6.2X-7X speed increase for a K-Means MapReduce algorithm running on top of the Hadoop framework and APARAPI (Abstract).

Keywords—OpenCL; FPGA; Java; Framework; APARAPI; Programming(key words)

I. INTRODUCTION

Using GPUs to accelerate computation has been shown to be more power efficient than using CPUs alone [1]. Recent research is suggesting that using FPGAs as accelerators can further increase power efficiency for certain types of data-centric applications [2, 3]. One main drawback to using FPGAs is the difficulty in programming them [4]. The traditional way to program FPGAs has been through the use of hardware description languages (HDLs) such as Verilog and VHDL; Languages which require technical abilities and know-how not found in the average computer programmer [7]. The emerging open programming standard for heterogeneous computing is OpenCL [5]. Recently Altera Corp released an OpenCL SDK for FPGAs [6] allowing FPGAs to be treated as OpenCL based accelerators. The OpenCL programming framework, although a major step in the effort to ease FPGA programming, still presents challenges to programmers since it requires both proficiency in C and in-depth understanding of the inner workings of the OpenCL programming model. In addition to that, recent surveys [7] show that a significant portion of software programmers use programming languages such as Java, C# and Python which offer rapid prototyping and easier software maintenance. In recent years several initiatives [11, 12, 19, and 20] have been launched in an effort to bring heterogeneous computing to the masses. One such effort is a framework called APARAPI [11] which was originally developed by AMD to target AMD GPU/APU architectures [13]. APARAPI is a Java based programming framework that aims to lower the heterogeneous programming bar. APARAPI offers programmers the ability to run heterogeneous computing applications using a familiar Java interface while hiding many of the intricate details that are associated with OpenCL programming. The main drawback of the APARAPI framework is that it was designed to specifically support AMD devices which make it unusable on other architectures. In this work our aim was to remove APARAPI’s dependency on the AMD architecture essentially enabling it to run on any other heterogeneous architecture that is OpenCL compatible. Extend APARAPI to support FPGA based devices and investigate the use of APARAPI on real world parallel algorithms in the data center. In addition we explore the possibility of incorporating APARAPI-FPGA into the Hadoop [16] framework, an industry standard Java software framework used in massively parallel distributed processing of Big Data for collaborative filtering, clustering, and classification.

The rest of the paper is organized as follows. Section II gives an overview of the APARAPI framework and reviews the steps performed in order to modify APARAPI to support Altera OpenCL SDK for FPGA devices. Sections III and IV describe the software and hardware used in our experiments, Section V details benchmark performance and power analysis of the modified framework on FPGAs, Section VI contains related work, and Section VII presents our conclusions.

II. APARAPI FRAMEWORK FOR FPGAS

APARAPI is a Java based framework that allows a programmer to write code in high level Java and have this code automatically translated and divided into host code and OpenCL code. The user of APARAPI is freed from the details of querying, instantiating and transferring data to and from the OpenCL compatible device.

Our work revolved around modifying the APARAPI framework so it can support other types of OpenCL compatible devices. Initially our efforts were geared towards targeting FPGAs, but we plan to investigate the ability to target multiple devices at the same time. Fig. 1 shows how the proposed multi target modified architecture would look like.

This work was carried out by Oren Segal as a Summer Intern at Hewlett Packard.
In order to modify the framework to support Altera OpenCL for FPGAs we had to disconnect the dependency on the AMD OpenCL SDK, link the APARAPI native layer against the Altera OpenCL Libraries and make any code modifications needed to support the new types of devices, specifically FPGAs. Step one and two required modification to the make files so they will be linked against the new set of Altera OpenCL libraries. Several libraries had to be replaced and minor build environment modifications made to accommodate that step. Step three was more involved and required changes to the native APARAPI layer that calls into OpenCL, handles memory allocation, device selection, kernel compilation etc.

### III. SOFTWARE LEVEL EXPERIMENT DESIGN

#### A. Standalone Benchmark Algorithms

Three algorithms that can potentially exploit parallelism very well were chosen: Mandelbrot fractal set calculation, Black-Scholes option pricing and NBody physics simulation. These algorithms were run on both APARAPI framework versions without any Java user code modifications. The OpenCL kernel that is auto generated by the APARAPI framework was fed to and automatically optimized by the Altera OpenCL compiler.

#### B. MapReduce Based Algorithms

In the next stage we turned our interest to MapReduce algorithms [15] based on the common Hadoop framework [16]. Specifically the K-Means algorithm [17] was chosen as a popular representative of a MapReduce algorithm.

#### C. K-Means Experiment Design

We started with a standard reference Java K-Means map reduce sequential algorithm for the Hadoop framework [33], converted it to work with floating number precision and optimized it. In the next stage we created a new parallel Java algorithm in APARAPI. In order for the algorithm to run effectively on an FPGA we had to find the most computational intensive part of K-Means and make sure it has enough data to process on each invocation of the kernel. To do that we implemented the kernel as an aggregated mapping task, calculating the Euclidian distances of points from centers, which has the complexity of O(NKD) floating point operations per Hadoop iteration.

### IV. HARDWARE LEVEL EXPERIMENT DESIGN

In order to simulate several typical data center scenarios we chose two types of systems:

- **System A (SysA/SA):** Low end server equipped with an Intel i7 3770 3.4GHz processor with 12GB DDR3 1333MHz RAM, one that could be found in a typical ad hoc machine cluster.
- **System B (SysB/SB):** High end server HP DL180 G6 with two Intel Xeon L5630 2.13GHz processors with 144GB DDR3 1333MHz RAM, one that would be more typically found in a traditionally organized data center environment.

Both machines were equipped with a Nallatech PCIe-385N A7 FPGA board (Fig 4) with 8GB RAM connected through PCIe second generation connection to the host system.

### V. RESULTS

#### A. Standalone Benchmark Algorithms

1) **Speed Analysis**

Performance results for the Mandelbrot fractal set calculation, black-scholes option pricing and NBody physics simulation are presented in Table II. All three algorithms show significant speed increase when compiled with auto optimization on. These algorithms are computational intensive and the overhead associated with data exchange over PCIe between the host and the FPGA device is relatively small, meaning that the ratio between the amount of computations done in the FPGA kernel and the amount of data that is transferred between the host machine and the FPGA in each kernel invocation is relatively high.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Speedup System Idle Time Vs Power Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>mandel</td>
<td>SysA  SysB 0%  20%</td>
</tr>
<tr>
<td>black-scholes</td>
<td>SysA  SysB 0%  20%</td>
</tr>
<tr>
<td>nbody</td>
<td>SysA  SysB 0%  20%</td>
</tr>
</tbody>
</table>

#### Table I. AVERAGE POWER CONSUMPTION

<table>
<thead>
<tr>
<th>System Type</th>
<th>System Power in Watts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Idle</td>
</tr>
<tr>
<td>CPU &amp; FPGA</td>
<td>42.6</td>
</tr>
<tr>
<td>CPU</td>
<td>23.1</td>
</tr>
<tr>
<td>Difference (W)</td>
<td>19.5</td>
</tr>
</tbody>
</table>

#### Table II. SPEED UP AND POWER SAVINGS

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Speedup</th>
<th>System Idle Time Vs Power Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>mandel</td>
<td>4X</td>
<td>73.4%  71.8%  77.27%  76.87%</td>
</tr>
<tr>
<td>black-scholes</td>
<td>4X</td>
<td>73.4%  71.8%  76.21%  75.79%</td>
</tr>
<tr>
<td>nbody</td>
<td>4.8X</td>
<td>77.8%  76.5%  80.70%  80.36%</td>
</tr>
</tbody>
</table>
2) Power Analysis

As can be seen in Table I the mere inclusion of the FPGA in the system cause a 19.5W power increase in idle mode and running CPU threads (JTP) at max utilization causes an additional power drain that takes the total power overhead to 20.6W and 24.7W on systems A and B respectively. The power consumption balance moves in favor of the FPGA system when the system is at high utilization. When the system operates at 100 percent utilization in JTP mode, all cores and threads of the CPU are active and while operating at FPGA mode only one thread is fully active (12.5-6.25 percent), while the rest of the computational intensity is offloaded to the FPGA device. At this point the difference in power consumption between the FPGA and CPU drops significantly and while still in favor of the CPU, as we can see in Table II, since the total processing speed of the CPU+FPGA far surpasses the CPU (up to 4.8X-5.3X), the total power savings goes up-to 77.8-80.7 percent on system A/B respectively.

Note that the best power performance results are obtained in an ideal scenario where the FPGA system is utilized at hundred percent meaning no system idle time. But since this scenario is highly unlikely in a real data center environment [9] an eighty percent utilization rate is displayed in the table as well. The power savings show that as long as the FPGA based system remains sufficiently utilized the benefit of using such a system outweighs the increase in idle power consumption.

B. K-Means Algorithm

1) Speed Analysis

The amount of floating point operations that the K-Means map kernel algorithm has to perform is in the order of N*K*D per iteration. Our experiments show that for the algorithm to run faster on the FPGA this complexity measure needs to exceed 2E+08. Fig. 2 shows the speedup, relative to the optimized K-Means sequential Java algorithm, in time to complete 2 iterations of the K-Means MapReduce on Hadoop with N*K*D equal to 1.6E+11. The results show a maximum of 4.8X gain for system A and 7X for system B, for the FPGA accelerated version compared to the sequential Java version. An interesting note is that for different dimensions (D), different versions of the FPGA kernel perform better. This can be related to the level of loop unrolling. For example on both systems, for the 8 dimension workloads the best performance is on the 8 processing units/8 times loop unrolling version (8PU-8UL), on the other hand for 6 dimensions workloads the fastest version is the 8 processing units/6 times loop unrolling version (8PU-6UL). The performance of the JTP version is much higher than the sequential Java version since it is a highly optimized version that runs the kernel on all available CPU cores, at full utilization, but it still runs up to 44% slower and consumes up to 20% more power than the FPGA version to complete the same task. Since the amount of time to setup and transfer data to an acceleration device is significant, for a low computation/data ratio, there is a negative effect to running the algorithm on the FPGA device.

2) Power Analysis

In order to assess the relative power efficiency of the different algorithm variations, we measured their average power during execution. We then calculated the average power savings in the top ten biggest data loads, in all of which the N*K*D factor was bigger or equal to 3E+10. The results in Fig. 3 show that running the algorithm on an FPGA device can save up to 65% power on system A and up to 80% on system B when compared to the sequential version. In addition when comparing the FPGA version to the JTP version we save up to 18% and 20% on system A/B respectively.

VI. RELATED WORK

Several high level OpenCL based programming frameworks have been developed in recent years [11, 19, and 20]. Although these libraries support OpenCL their focus is on GPU and CPU development. Higher then HDL level programming frameworks and DSLs for FPGAs have been an active research area [21, 22, 23, 29] and several open source projects exist [27, 28, 29] and continue development in the field, but they generally stop at the C/C++ level. Several commercial tool chains [24, 25] that allow a hardware flow from C/C++ to FPGA have become standard industry tools in recent years. The recent built-in Xilinx support for C/C++ and SystemC [26] and Altera’s current and Xilinx planned [32] support for OpenCL in their FPGAs are more signs indicating the
importance placed on high level programming of FPGAs. Existing higher then C/C++ level frameworks such as the python based MyHDL [30] and Java based MaxCompiler [31] allow high level programming of FPGAs but do not offer the programming standardization that OpenCL has to offer.

VII. CONCLUSIONS
On highly parallel algorithms written in Java, assuming sufficient system utilization, we find significant speedup and power savings for a system with an FPGA accelerator. On the MapReduce Hadoop framework, using the k-means algorithm we lose some of the advantages of the FPGA accelerator because of the difficulties in maximizing parallelism for a single machine. Careful analysis of computational complexity vs. data transfer overhead is needed in order to utilize the FPGA acceleration device optimally. To achieve optimal performance, different OpenCL kernel versions with different optimization options should be employed in unison. Custom kernel modifications and optimizations were required for some algorithms to get good performance results, auto optimizations do not always work well.

VIII. ACKNOWLEDGMENTS
We would like to thank Hewlett Packard, Nallatech and Altera for their generous hardware and software donations. We would especially like to thank HP Servers for their continuous support throughout the lifetime of this project.

IX. REFERENCES


[14] Timothy Prickett Morgan. Nvidia, Continuum team up to sling Python at GPU coprocessors http://www.theregister.co.uk/2013/03/18/nvidia_continuum_pyhton_on_cuda_gpu/


[29] ROCCC. http://www.jacquardcomputing.com/roccc/


Figure 2.Nallatech PCIE-385n A7 Altera Stratix V board