Implementation and Performance Analysis of OpenCL Programming Model on CPU, GPU (K20 and K80) and FPGA

Md Maruf Hossain[1], Sowmya Suresh Chander[1], and Shanmathi Rajasekar[1]

{*mhossa10, ssureshc, srajase1*}*@uncc.edu*
University of North Carolina at Charlotte

***Abstract*— Over the recent years, high-performance computing is rapidly developing and inclining towards heterogeneous architectures. This shift is seen to improve the performance of the applications when compared to sequential execution. The level to which GPU and FPGA perform varies according to the benchmark algorithms. The aim of this project is to run and analyze OpenMP codes on CPU and OpenCL codes on GPU (K20 and K80) and FPGA platforms for Rodinia benchmarks. The performance is analyzed for all the three platforms (CPU, GPU, FPGA) on parameters such as speed and power consumed to determine the best-suited platform for the selected applications.**

I. INTRODUCTION

In todays high performance computing domain, the benefits of shrinking transistors have not made them more energy efficient. On the other hand, disruptive technologies like nonvolatile memory, photonic interconnects, growing user requirements for improved performance as well as the limitation of multi-core CPUs to effectively utilize the multiple cores have highlighted the need for building over traditional architectures. To address all these concerns, the computer industry is moving towards delivering improvements across power, performance, programmability, and portability. This has led to the transition from traditional compute units to heterogeneous systems to meet the demands of data-centric applications.

Heterogeneous systems combine heterogeneous computing elements into one coherent environment making them more power efficient. These architectures come across as flexible, efficient and portable. The processing elements can be GPUs or reconfigurable units like FPGAs. Both GPUs and FPGAs have their own advantages. FPGAs offer lower latencies than GPU and are more deterministic in their approach. While GPUs are throughput oriented and are massively parallel that enables them to run a software algorithm faster than a conventional processor. In addition, GPUs offer backward compatibility that enables them to run on older chips with new software algorithms.

II. MOTIVATION

With computer systems adapting to heterogeneous models such as combining graphics processors(GPU), Field programmable gate array (FPGA) with conventional CPUs, research in parallel computing is essential to ensure future progress in the world of high-performance computing. The future innovation lies in choosing application specific hardware architecture that accelerates the performance of the application. A standard benchmark is required to understand the behavior of platforms, evaluate performance and pave way for optimizations. The existing parallel programs are more CPU specific and support CPU architectures and are not developmental with respect to the accelerator architecture that differs to a great extent from the CPU architectures.

We overcome the dearth of available codes that support heterogeneous architectures by utilizing Rodinia benchmark. In our comparisons between CPU, GPU and FPGA using Rodinia, we have understood that the major architectural differences between three platforms have important implications for software. We start by choosing 6 kernels from Rodinia benchmark which are GPU friendly and port it to FPGA using C to comprehend FPGA working.

The benchmarks are evaluated on Intel 2.6GHz 16-core processor E5-2697A V4 CPU, Nvidia K80 GPU, Nvidia K20 GPU and Altera Arria 10 1150 GX FPGA platforms.

Our Contribution to the project:

- We evaluate and perform application case study of 6 kernels belonging to Rodinia benchmarks with different problem sizes on CPU, GPU and FPGA and identify the bottlenecks in each platform.
- We present reasoning of the behavior of kernel in each platform and also the best-suited platform for each kernel.

III. RODINIA BENCHMARK

Accelerators are increasingly becoming popular due to the ease of programmability and better-promised performance. Existing benchmarks are suited for conventional CPU and there is the shortage of benchmarks that sheds light on the architecture strengths and weakness of accelerators.

TABLE I: Rodinia Benchmarks

Benchmark	Dwarf	Domain
CFD	Unstructured Grid	Fluid Dynamics
Pathfinder	Dynamic Programing	Grid Traversal
LUD	Dense Linear Algebra	Linear Algebra
K-means	Dense Linear Algebra	Data Mining
Leukocyte	Structured Grid	Medical Imaging
K-NN	Unstructured Grid	Data Mining

To compare different platforms and identify bottlenecks, Rodinia Benchmark is used. Rodinia benchmark suits target heterogeneous computing platforms including both host CPUs and devices such as GPUs and FPGA. It is a set of free and open benchmarks that fill the dearth of available code for heterogeneous platforms. These benchmarks address heterogeneous computing using OpenMP, OpenCL, and Cuda.

The benchmark applications used - Computational fluid dynamics(CFD), Nearest Neighbor(NN), K-means, Leukocyte, Lower Upper Decomposition(Lud) and n Pathfinder. Important features of Rodinia benchmarks, that differentiate it from other benchmark suits are as follows:

- Rodinia has non-traditional memory hierarchies like scratchpad and texture units.
- It provides multiple versions of the same application, allowing the user to compare the application across different platforms.
- It adopts the offloading model where accelerators use separate memory other than the main memory.
- It covers a diverse range of domains with wide input ranges.

A. Computational fluid dynamics(Cfd)

This application solves for 3-D Euler equations used in Fluid Dynamics. CFD Solver is an unstructured grid finite volume solver for the three-dimensional Euler equations for inviscid, compressible flow. We have used three datasets of sizes 97000, 193000, and 230000.

B. Pathfinder

This benchmark finds the shortest path of a 2D grid, row by row, by choosing the smallest accumulated weights. Every iteration is parallelized which calculates the shortest path. We have used three datasets of sizes 1000, 5000 and 10000.

C. Lower Upper Decomposition(Lud)

This parallel application is used for calculating a set of linear equations. It is an algorithm to decompose a matrix to a product of a lower triangular matrix and an upper triangular matrix. The decomposition is done in parallel. We have used three datasets of sizes 1024, 2048 and 5120.

D. Kmeans

This data-mining application is used for clustering given data set based. This is based on mean-based data partitioning method which is done by calculating the mean value of sub-clusters and grouping the data point to the nearest cluster. We have used three datasets of sizes 1024, 2048 and 5120.

E. Leukocyte

This application detects and tracks white blood cells in blood vessels. The cells are detected in the first video frame and are tracked from second video frames. We have used three datasets of sizes 10000, 104400 and 200000.

F. Nearest Neighbor(K-nn)

This data-mining application is used for classification and regression and finding k-nearest neighbors from a given data set. It is used in pattern recognition, machine learning, computer vision and coding theory. We have used three datasets of sizes 1024, 2048 and 5120.

IV. ARCHITECTURAL ANALYSIS

Platform	Hardware	Software
CPU	Intel 2.6GHz 16-core processor – E5-2697A V4	GCC Compiler
GPU	NVIDIA TESLA K80 NVIDIA TESLA K20	OpenCL Version 2
FPGA	Altera Arria 10 1150 GX FPGA	OpenCL Version 2.0 Altera SDK Profiler Tool NSF Chameleon FPGA CLOUD

Fig. 1: Hardware Platforms and Compilers

A. OpenMP

OpenMP is an application programming interface that is used for multi-threaded shared memory parallelism. It proves to be portable and scalable model. OpenMP is a method of parallelizing whereby a master thread forks a specified number of slave threads and the system divides a task among them. The threads then run concurrently, with the runtime environment allocating threads to different processors.

B. OpenCL

OpenCL is an open framework targeted for Heterogenous platforms like GPU, FPGA, and other processors. It provides a set of APIs and programmed in C language.

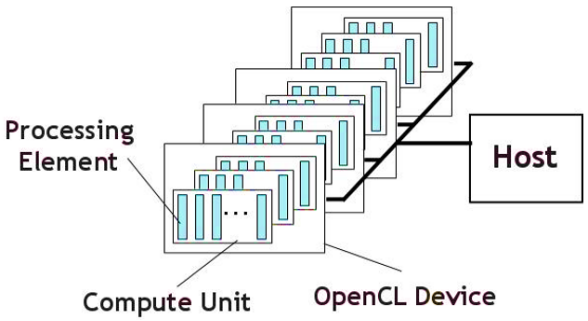

Fig. 2: OpenCL Platform Model

From Figure 2, it consists of host and coarse-grained compute devices. A host is a CPU and compute devices can be GPU, FPGA, and other processors. In this paper, we consider GPU and FPGA. Each of these OpenCL or compute devices consists of Compute Units like group of execution units and arithmetic processing unit. These compute units will have many Processing elements at the lowest level that executes kernels.

The OpenCL memory model consists of private memory pertaining to the work item, the local memory that is shared within the work items in a workgroup, global memory that is shared by all the workgroups and host memory which is the memory on the CPU. The OpenCL application runs on the CPU which then submits work to the compute devices.

The OpenCL standard:

- Supports both data- and task-based parallel programming models.
- Utilizes a subset of ISO C99 with extensions for parallelism.
- Defines consistent numerical requirements based on IEEE 754.
- Defines a configuration profile for handheld and embedded devices.
- Efficiently interoperates with OpenGL, OpenGL ES, and other graphics APIs.

C. K20 and K80

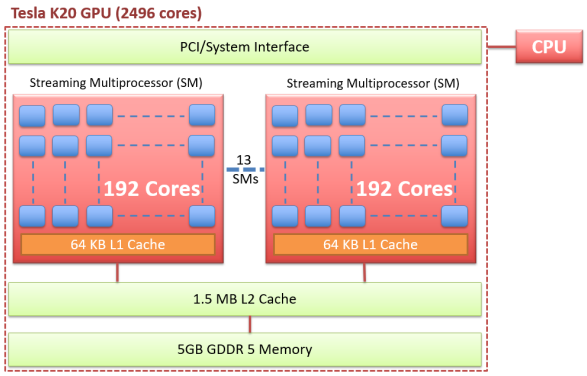

Fig. 3: Architecture of TESLA K20

General-purpose computing on graphics processing units is the use of a graphics processing units (GPU), which typically handles computation only for computer graphics to perform computation in applications traditionally handled by the CPU. Essentially, a GPGPU pipeline is a kind of parallel processing between one or more GPUs and CPUs. While GPUs operate at lower frequencies, they typically have many times the number of cores. Thus, GPUs can process far more pictures and graphical data per second than a traditional CPU[1]. GPUs are simply SIMD vector processors. Both K20 and K80 belong to Kepler architecture. The block diagram of K20 GPU and K80 are seen in Figure 3 and Figure 4 respectively. A Streaming multiprocessor(SMX) is a SIMD processor and these perform the actual computation. Unified memory requests are supported by Kepler architecture. Each SMX has an L1 cache of size 64 KB, 1536 KB of dedicated L2 cache memory. The L2 cache is the primary point of data unification between the SMX units, servicing all load, store, and texture requests and providing efficient, high-speed data sharing across the GPU.

Nvidia K20 GPU implements a full Kepler GK1120 that includes 13 SMXs each containing each contain 192 singleprecision CUDA cores, 64 doubleprecision units, 32 special function units (SFU), and 32 load/store units (LD/ST) and use the primary GPU clock. Each of the 192 core has a (texture) cache, a register file and runs multiple threads in parallel with simultaneous multithreading. The SMX schedules threads in groups of 32 parallel threads called warps. Each SMX has four warp schedulers and eight instruction dispatch units, allowing four warps to be issued and executed concurrently. This warp scheduler selects four warps, and two independent instructions per warp can be dispatched each cycle. Apart from dual issuing, a warp scheduler can issue back to back independent instructions to a warp. If there is a dependency or the unit executing the next instruction then next eligible warp is picked. own set of general purpose registers, condition codes, predicates codes and local memory.

Similar to K20, K80 implements two GK210 GPUs. Each GPU has 13 SMXs each possessing 192 cores, 64 doubleprecision units, 32 special function units (SFU) and 32 load/store units (LD/ST) and use the primary GPU clock. Altogether a total of 2x2496 cores in K80.In addition, there is an L1 cache of size 128KB in each SMX and dedicated L2 cache memory of size 1536KB. Each of the 192 core has a (texture) cache, a register file and runs multiple threads in parallel with simultaneous multi-threading.

D. Altera 10 FPGA

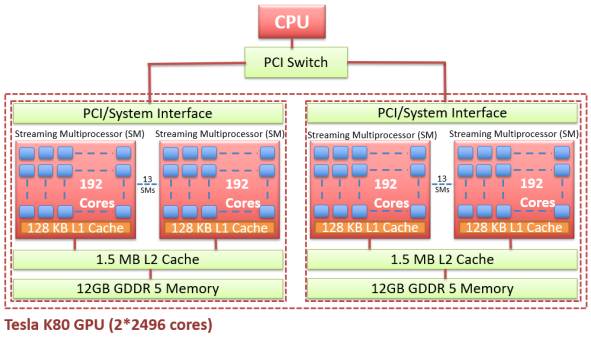

Fig. 4: Architecture of TESLA K80

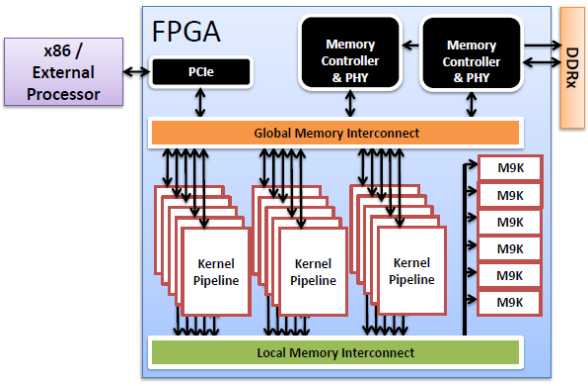

Fig. 5: Architecture of ALTERA FPGA

Alteras Arria FPGAs and SoCs deliver optimal performance and power efficiency in the midrange. By using TSMC's 20-nm process technology on a high-performance architecture, Arria 10 FPGAs and SoCs deliver higher performance than previous-generation high-end FPGAswhile simultaneously reducing power by offering a comprehensive set of power-saving technologies. Alteras Arria 10 SoCs offer a second generation SoC product that both demonstrates a long-term commitment to the SoC product line and extends Alteras leadership in programmable devices that feature the ARMbased hard processor system (HPS). Important innovations in Arria 10 devices include:

- Enhanced core architecture delivering 60 percent higher performance than the previous generation midrange (15 percent higher performance than previous fastest high-end FPGAs)
- Integrated transceivers with short reach rates up to 28.05 Gbps and backplane capability up to 17.4 Gbps
- Hard PCI Express Gen3 intellectual property (IP) blocks
- Hard memory controllers and PHY up to 2666 Mbps
- Variable precision digital signal processing (DSP) blocks
- Fractional synthesis PLLs
- Up to 40 percent lower power compared to prior midrange FPGAs and up to 60 percent lower power compared to prior generation high-end FPGAs due to a comprehensive set of advanced power-saving features
- 2nd generation ARM Cortex-A9 hard processor system (HPS) for SoC variants
- Integrated 10GBASE-KR/40GBASE-KR4 Forward Error Correction (FEC). [7]

E. Altera SDK Profiler Tool

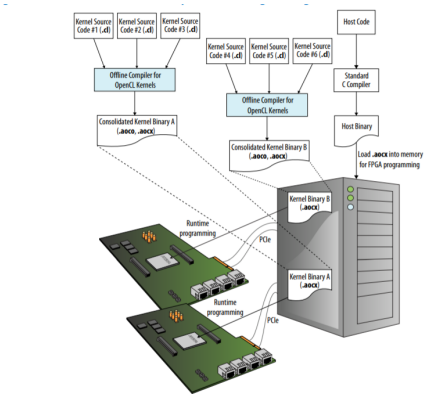

Fig. 6: Hardware Specification of ALTERA FPGA

The Intel FPGA SDL for OpenCL has a host program and FPGA programming bitstream. The host program manages both the application as well as FPGA accelerator. The execution consists of :

- Compiling kernels of OpenCL to a file that is used by the host program to program the FPGA.
- The C compiler on host works on the host program and link it to the Intel FPGA to obtain the runtime libraries.

The figure 6 shows a flowchart where the kernel source file(.cl) has the kernel source that is run on the FPGA. The kernels are then grouped into a temporary file and then compiles this file to form ".aoco" and ".aocx" file object files. The ".aoco" is a temporary object file and ".aocx" is the hardware configuration file containing information to run the FPGA at runtime. After this, the host loads object into memory and uses these to program the FPGA.

V. Methodologies

The OpenMP codes of the six benchmarks were compiled on Intel 2.6GHz 16-core processor E5-2697A V4 CPU and the run times of the benchmarks were determined. With respect to GPU, the benchmarks were compiled both of Nvidia Tesla K20 and Nvidia Tesla K80 and the run times of the GPU were calculated. The power consumed on GPU was initially to be obtained with the help of 'nvprof', however, it supports only CUDA code on Nvidia platform.

Taking into account, FPGA, the OpenCL codes of the six applications were ported to the FPGA platform and were initially compiled on the TACC server by setting up a Chameleon cloud account and connecting to it using TACC tokens. Each benchmark took about 3-10 hours for compilation and was compiled using Altera SDK Profile tool to generate .aocx binary file. Following this, these benchmarks were run on the Chameleon cloud to obtain the run times of all six benchmarks. The resource utilization of all the six benchmarks was obtained from a text file generated along with .aocx binary file. With these resource utilization values, the power was computed using Altera 10 Powerplay Early Power Estimator.

VI. Results and Analysis

Six Rodinia benchmarks were executed on CPU, GPU (K20 and K80) and Altera FPGA and their runtime were obtained.In addition, FPGA's resource utilization and power consumption were also calculated.

A. CFD: Time Complexity

CFD was implemented on four different architecture and time complexity for various data sizes were obtained. The result shows CFD is suitable for heterogeneous architecture, GPU. FPGA is better than CPU but not GPU.

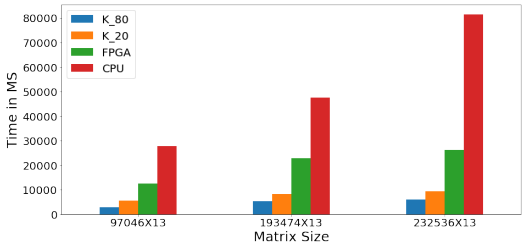

Fig. 7: Time Complexity of Rodinia Computational Fluid Dynamics(cfd) Benchmark

B. Pathfinder: Time Complexity

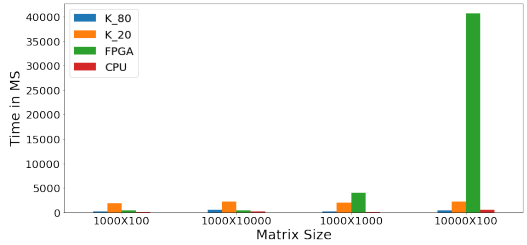

Fig. 8: Time Complexity of of Rodinia Pathfinder Benchmark

The result set indicates that it has more sequential nature than parallel. Thus, it performs well in CPU compared to other architectures. We used four different data sizes for this benchmark.

C. LUD: Time Complexity

GPU consists of execution units which makes it massively parallel. For smaller dataset, many of these execution units are free, thus reducing the performance in GPU. Thus for a smaller dataset in LUD, CPU outperforms GPU and FPGA.

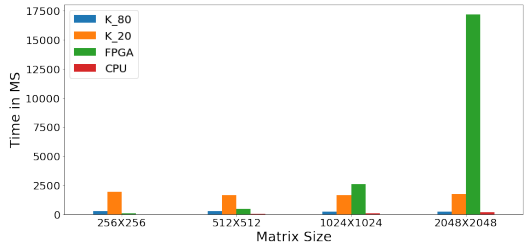

Fig. 9: Time Complexity of Rodinia Lower Upper Decomposition(lud) Benchmark

D. Kmeans: Time Complexity

For K-means, GPU K80 was better than CPU and FPGA. This is because means benchmark had many parallelized codes. k80 is better than k20 because there is twice the number of cores in K80 than in K20.

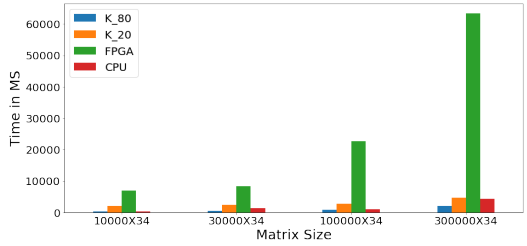

Fig. 10: Time Complexity of Rodinia Kmeans Benchmark

E. Leukocyte: Time Complexity

For Leukocyte benchmark, GPU k80 was better than CPU and FPGA. This benchmark had many parallelized codes, which makes runtime smaller when compared to other platforms.

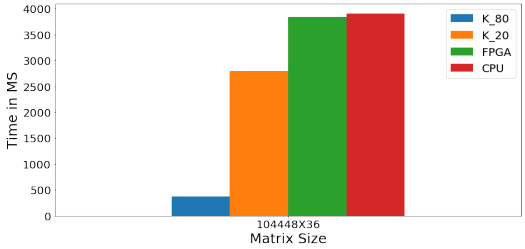

Fig. 11: Time Complexity of Rodinia Leukocyte Benchmark

F. K-NN: Time Complexity

For K-nn, CPU runtime and GPU runtime are almost same for this benchmark. This is because it had lesser parallelized codes when compared to sequential codes.

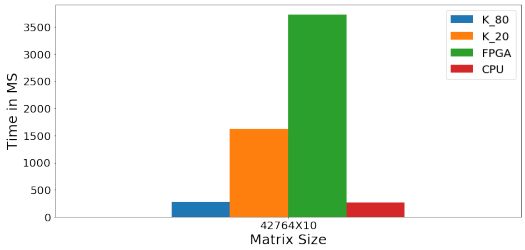

Fig. 12: Time Complexity of Rodinia Nearest Neighbor(nn) Benchmark

G. GPU vs FPGA Speedup

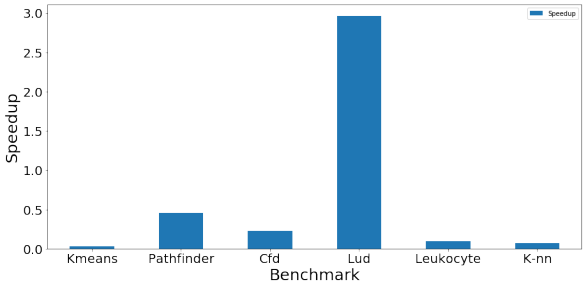

Fig. 13: Speedup: GPU vs FPGA

Speedup of FPGA is calculated with respect to GPU. Using runtime of GPU and FPGA, we calculate the GPU vs FPGA speedup. In Table II, if the value of speedup is greater than 1, then FPGA outperforms GPU and if it is less than 1 then GPU is better. From speedup, we can see GPU outperforms FPGA for 5 out of the 6 benchmarks.

TABLE II: Speedup: GPU vs FPGA

Benchmark	Program size	GPU	FPGA	Speedup	GPU or FPGA
Leukocyte	104448X36	371.83	3845.92	0.096	GPU
K-NN	42764X10	270.96	3724.9	0.072	GPU
LUD	256X256	258.68	87.17	2.96	FPGA
K-means	10000X34	347.54	7019.15	0.029	GPU
Pathfinder	1000X100	199.33	437.18	0.455	GPU
CFD	97046	2938.75	12523.36	0.23	GPU

H. Resource Utilization

Estimated Resource Utilization and Actual Resource Utilization were obtained from a text file generated after compilation and execution.

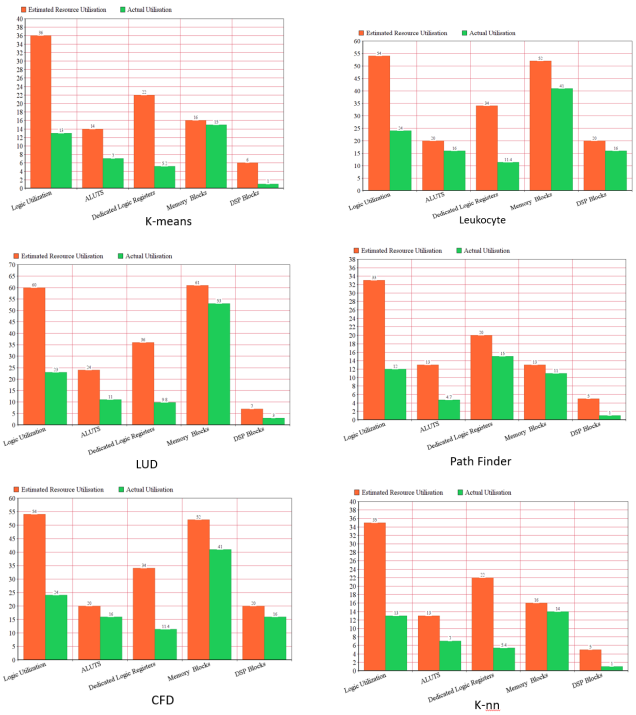

Fig. 14: Resource Utilization Report of Six Benchmark

I. FPGA Power Usages

TABLE III: FPGA Power Consumption

Benchmark	FPGA Watts
Leukocyte	2.556
K-NN	2.389
LUD	2.701
K-means	2.406
Pathfinder	2.406
CFD	2.950

Using Arria 10 Early Power Estimator, we were able to find the runtime power consumption. The power estimator is an excel tool with Altera's macros, which calculates power using resource utilization. As we are implementing OpenCL on Nvidia's K20, we were not able to find the power using NvProf. Fig. 15 represents the bar chart of power vs six benchmarks.

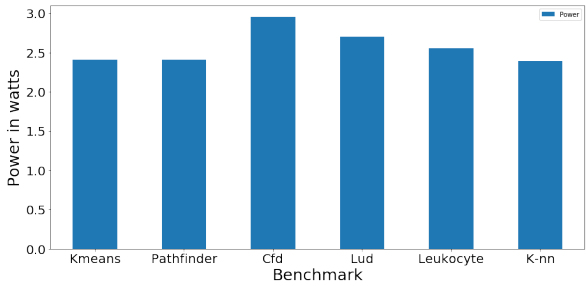

Fig. 15: Power Consumption

VII. CONCLUSIONS AND FUTURE WORK

Based on the Run time obtained, we observed that 3 of 6 benchmarks seems to perform better in K80 GPU platform. Depending on choices of parameters, the best platform can be chosen according to the benchmark. To further improve the performance of benchmarks on each platform, the OpenCL codes can be optimized for specific platforms. However, to decisively conclude GPU or FPGA to be the best platform, requires further research and analysis of diverse applications.

ACKNOWLEDGMENT

Special thanks to our course instructor Dr. Hamed Tabkhi.

REFERENCES

[1] wiki: $https://en.wikipedia.org/wiki/Graphics_processing_uni$
[2] Jie Shen, Ana Lucia Varbanescu: A Detailed Performance Analysis of the OpenMP Rodinia Benchmark.
[3] Taneem Ahmed: OpenCL framework for a CPU, GPU, and FPGA Platform.
[4] Qiang Wu, Yajun Ha, Akash Kumar, Shaobo Luo, Ang Li, Shihab Mohamed: A Heterogeneous Platform with GPU and FPGA for Power Efficient High Performance Computing.
[5] Mohammad Alawieh, Maximilian Kasparek, Norbert Franke and Jochen Hupfer: A High Performance FPGA-GPU-CPU Platform for a Real-Time Locating System.
[6] Karl Pereira, Peter Athanas, Heshan Lin and Wu Feng: Spectral Method Characterization on FPGA and GPU Accelerators.
[7] $http://www.mouser.com/ds/2/591/arria_10_aib-268767.pdf$