An evaluation of contemporary heterogeneous computing platforms for data intensive applications

Vincent T. Lee, Carlo C. del Mundo, Eddie Yan, Armin Alaghi, Mark Oskin, Luis Ceze
University of Washington
{vlee2, cdel, eqy, armin, oskin, luiceze}@cs.washington.edu

***Abstract*—The end of Dennard scaling is forcing innovation in computing architectures; the old regime, where exponential benefits accrued by using a newer process technology, no longer holds. One path of innovation is to exploit heterogeneous platforms, and match applications to the platforms on which they execute efficiently. Heterogeneous processing in the server domain is a big hardware/software design space. As a result, research that looks narrowly at one or two applications, and hardware options does not usually generalize. In this work, we present a broad comparison of a variety of hardware accelerators and applications. We investigate an Intel Xeon E5-2620, ARM Cortex A15, Nvidia Tegra Jetson K1, Nvidia Titan X, Intel Xeon Phi coprocessor, and Xilinx Zynq FPGA, and evaluate how efficiently they execute key data mining applications. We focus on energy efficiency, but also reflect on the developer effort required to utilize each platform. At the highest level, the results mirror conventional wisdom: FPGAs are about as efficient as GPUs which are more efficient than low-power processors. Developer effort is the exact opposite; processors are the easiest to program, followed by GPUs and then FPGAs. If the producers of more energy-efficient compute platforms (FPGAs, GPUs) wish to be ubiquitous in the datacenter, they (still) must work on improving their programming environments.**

I. Introduction

The total number of bits stored in the datacenter is projected to grow at a classic Moore's law rate, from 4.4ZB in 2013 to 44ZB in 2020 [1]. To support this growth, datacenters must scale both storage and compute capability. Unfortunately, the end of Dennard scaling means classic server architectures are not seeing commensurate gains in compute efficiency from new process technologies. Fundamentally, there are two paths forward and the future will likely entail a mixture of both: (i) scale out by adding more servers and consequently more datacenters or (ii) perform computations more efficiently.

Heterogeneous architectures offer one approach towards improving energy efficiency. Heterogeneity includes different core-types [2]: programmable accelerators (GPUs and Intel Xeon Phi coprocessors) and FPGA substrates [3]. Each of these options has been studied in the past, but often in isolation. A definitive look at all of these options, with the same set of applications has been lacking. That comparison is the essential contribution of our work. We present a broad evaluation across hardware options available for constructing a heterogeneous compute platform; we evaluate their energy efficiency and programmability for five data mining applications that have remained relevant over the past decade [4]: k-means, k-nearest neighbors, linear regression, support vector machines (SVMs), and adaptive boosting (AdaBoost). We evaluate six hardware options: Intel Xeon E5-2620, ARM Cortex A15, Nvidia Tegra Jetson K1, Nvidia Titan X, Intel Xeon Phi coprocessor, and Xilinx Zynq XC7Z020 FPGA.

At a high level, our results match the conventional wisdom in our field: FPGAs are more energy efficient than GPUs which are more energy efficient than CPUs. Programmer effort and tool quality run in the exact opposite direction as energy efficiency. Programming FPGAs is still a monumental endeavor and general purpose computations on a GPU are still second class citizens compared to graphics; tool chains for CPUs on the other hand are mature and well known. However, beneath these high level results are some surprises. From the vantage point of a CPU-only system, GPUs provide most of the energy efficiency of an FPGA. While the Intel Xeon Phi coprocessor is easy to write software for, it yields the least energy efficiency gains compared to GPUs and FPGAs.

II. Computing Platforms and Applications

We consider general purpose CPUs, GPUs, FPGAs, and the Intel Xeon Phi coprocessor for our evaluation. Our CPU processor baselines are an Intel Xeon E5-2620 processor which has six physical cores, and an ARM Cortex A15 processor on the Tegra Jetson K1 SoC development kit which has four cores. For our GPU evaluations we use an Nvidia Titan X GPU, and a Tegra Jetson K1; for our FPGA evaluation, we use a Xilinx Zynq XC7Z020 FPGA. The exact process technology, programming tool chains, and source languages are shown in Table I. The lithographies across each platform are comparable so we do not apply technology scaling to our results. While we expect higher capacity FPGAs to achieve higher performance by instantiating additional parallel cores, we expect the energy efficiencies to remain within the same order of magnitude due to the additional power draw.

On the applications side, we choose five data intensive workloads: k-means, k-nearest neighbors, linear regression, support vector machines, and adaptive boosting. These workloads are identified as common core algorithms used across machine learning, data mining, and computer vision [4]. In addition, these workloads are highly data parallel, which allows heterogeneous computing platforms to express their full parallel compute capabilities. We set parameters for each workload such that they are agnostic to the composition of the dataset being processed (e.g., fixed number of iterations in k-means). Specific workload implementation sources are shown in Table II. When available, we try to use readily available

TABLE I
COMPUTING ARCHITECTURES AND CONFIGURATIONS

Platform	Cores	Process (nm)	Compiler/ Tool Flow	Source
Intel Xeon E5-2620	6	32	gcc/g++	C/C++
ARM Cortex A15	4	28	nvcc	CUDA
NVIDIA Jetson TK1	192	28	nvcc	CUDA
NVIDIA Titan X	3072	28	nvcc	CUDA
Xilinx Zync ZC702	N/A	28	Vivado	Verilog
Intel Xeon Phi coprocessor M5110P	60 + 1	22	icc	C/C++

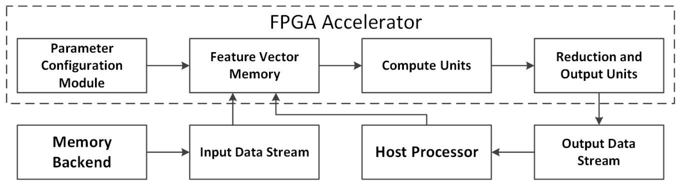

Fig. 1. High level architectural design pattern of FPGA accelerator designs

optimized GPU and CPU implementations; however, for cases where these implementations do not exist, we implement and validate our own. For the Intel Xeon Phi coprocessor, we reuse OpenMP implementations.

III. Accelerator Designs

At a high level we chose to implement a streaming architecture for each of our FPGA accelerators. First, we expect that for large scale data mining applications that the working set of the data will far exceed the capacities of onboard FPGA memories that typically cap at around several 100s of Megabits for Xilinx 7-series devices [5]. Second, we find that local data reuse is limited for the data mining applications we explore; typically the entire data stream is accessed and processed once in its entirety before it is accessed again. Third, a streaming architecture allows us to exploit inherent parallelism in these workloads and scale appropriately to higher or lower bandwidths. This scalability is crucial as memory substrates continue to evolve and provide increasing bandwidth or latency improvements.

The high level architecture for our FPGA accelerator designs is shown in Figure 1 and can be decomposed into four components: a set of configuration registers, a feature vector memory, compute units, and reduction units. For each of our five applications, we implement some variant of this high level design pattern to evaluate the FPGA performance. The configuration registers hold parameters such as vector dimensionality and dataset cardinality, as well as algorithm specific parameters such as number of clusters in k-means or number of neighbors in kNN. We assume a host processor is responsible for reconfiguring these registers appropriately depending on the workload. The feature vector memory serves as a small scratchpad which is used to hold workload specific data that is reused such as the query vector in kNN, or initial solution vector in SVM. The compute units vary depending on the application but at a high level are composed of several arithmetic units in parallel to exploit the inherent data parallelism in each workload. The specific role of each component in the context of each application is outlined in Table II.

IV. Methodology

To make comparisons for each workload as fair as possible across platforms, we select dataset sizes that are large enough to eliminate last level caching effects and maximize platform utilization. For the ARM Cortex A15, Intel Xeon E5-2620, and Intel Xeon Phi coprocessor we increase thread counts as necessary; for the Nvidia Tegra Jetson K1 and Titan X GPUs, we use dataset sizes which allow ample parallelism to exploit. To normalize across dataset sizes, we apply a scaling factor based on the dataset size while preserving algorithm specific parameters across executions.

A. *Power Measurement*

With the exception of the FPGA, we measure the average load power and idle power for each platform using a power meter and use the difference as the platform power consumption. We report dynamic compute power instead of overall system power because it more precisely highlights the differences between computing substrates; using overall system power is subject to additional variation such as number of memory modules, mass storage devices, PCIe cards, or other peripherals. We run multiple consecutive executions of each workload and report average power after application initializations; for certain targets, multiple executions are necessary to ensure the platform has “warmed up” and power readings have stabilized before taking average power measurements.

For the FPGA platform we use post-placement and route power estimates using Vivado 2015.4 using worst case activity factors. We do not measure the FPGA using a power meter because of our power meter’s limited precision. The dynamic compute power is agnostic to the dataset size and run time so we do not need to scale these results. We report the full power number for only the FPGA accelerator and omit SoC CPU power; in practice the CPU dynamic power would be minimal since it is largely inactive for these workloads and can be put in a low power idle state.

TABLE II
EVALUATED WORKLOADS AND MAPPING TO STREAMING ARCHITECTURE

Workload	K-Means [6]	K-Nearest Neighbors [7]	Linear Regression [8]	Support Vector Machine [9]	Adaptive Boosting [10]
Application	Unsupervised Clustering	Similarity Search	Model Training	Classifier Training	Classifier Training
CPU Source	NU-Minebench [11]	FLANN [12]	Custom	LibLinear [13]	Custom
GPU Source	Catanzaro et al. [14]	Garcia et al. [15]	Custom	LibLinear GPU [16]	Custom
Vector Memory	Cluster centroids	Query feature vector	Bias and weight vector	Bias and weight vector	Haar classifier
Compute Units	Euclidean distance	Euclidean distance	Mean squared error loss gradient	L2 regularized, hinge loss gradient	Exponential loss gradient
Reduction Unit	Nearest cluster and partial new centroids	Priority queue and top-k candidates	Accumulation of gradients	Accumulation of gradients	Accumulation of misclassified and total sample weight

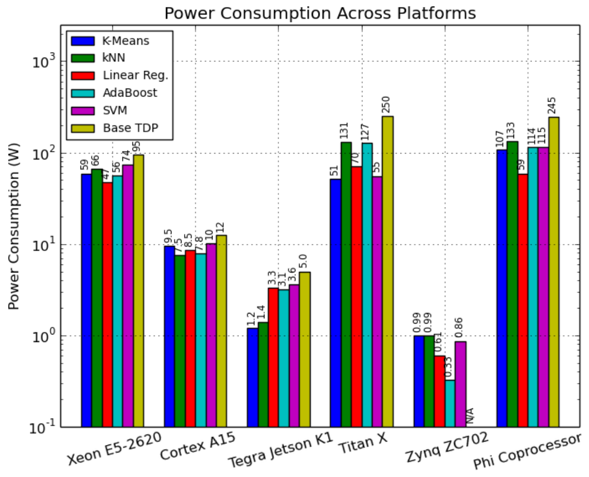

Fig. 2. Application average dynamic power consumption - measurements reflect average load power minus average idle power

B. Application Run Time

For each benchmark, we record the wall clock time for each workload kernel excluding data initialization and validation. To evaluate GPU benchmarks, we exclude the time to transfer the dataset to the GPU as initialization overhead since it is not part of the main kernel. When measuring run times for the Intel Xeon E5-2620, ARM Cortex A15, and Intel Xeon Phi coprocessor platforms, we sweep different thread counts and report the results for the best configuration. Finally for the FPGA, we report the run time by simulating the design and estimating the total time required to stream all of the data. Host processing time for each of the applications is negligible since most of the application time is spent processing data on the FPGA. The run time for each benchmark is then used to estimate the energy efficiency.

C. Energy Efficiency

To estimate the energy efficiency, we first compute the total energy required to process the given dataset by taking the measured run time and multiplying by the power. We then normalize across dataset sizes and compute the throughput per watt; this is done by taking the dataset size and dividing by the total energy (GB/J). This provides a "gigabyte of dataset processed per joule" metric which is agnostic to the actual dataset size across platforms. Our FPGA designs were synthesized to meet a 100 MHz clock target however for higher clock targets we expect the energy efficiency to remain roughly the same.

V. EVALUATION

In this section we present the power and energy efficiency results we observe across each of the evaluated platforms and benchmarks.

A. Dynamic Power Consumption

In Figure 2, we show our dynamic power measurements across each of our evaluated platforms for different benchmarks. We also show the thermal design power (TDP) or estimated maximum power for comparison [17]–[21]; FPGA maximum power estimates are omitted since they are not readily available and we use estimated peak power of the Tegra Jetson K1 platform for the ARM Cortex A15 TDP estimate. Our results are largely in line with TDP estimates (i.e. our power measurements are lower). The results also confirm conventional knowledge that embedded computing units such as the Tegra GPU, ARM Cortex A15, and Zynq FPGA consume an order of magnitude less power than their server counterparts.

B. Energy Efficiency

In Figure 3, we show the relative energy efficiency across each application workload and platform. We normalize our results to the energy efficiency of the Intel Xeon E5-2620 CPU and report relative improvements across platforms. Unsurprisingly our results show that general purpose cores have the worst energy efficiency and the ARM Cortex A15 energy efficiency is roughly the same order of magnitude as the Intel Xeon E5-2620 core (1.0x to 2.4x). Our results also show

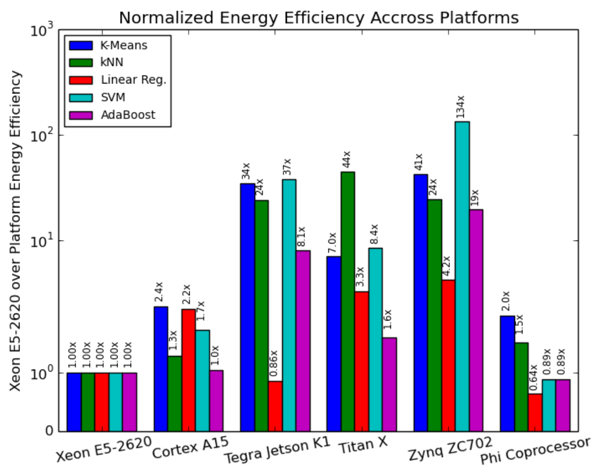

Fig. 3. Application energy efficiency relative to Intel Xeon E5-2620 CPU

the Intel Xeon Phi coprocessor operates at roughly the same energy efficiency as general purpose cores (0.64x to 2.0x). We also observe that a well optimized GPU implementation such as k-means or kNN operating on a Tegra Jetson K1 GPU can achieve roughly the same order of magnitude gains in energy efficiency as FPGAs. For our GPU custom implementations, while we see modest performance improvements, overall we do not see the improvements such as those garnered by well optimized variants. For linear regression, we find our GPU implementation is bandwidth bound and for AdaBoost our implementation is limited by regions of control divergence resulting in lackluster energy efficiency gains. Finally, our results show that the energy efficiency of embedded platforms such as the ARM Cortex A15 and Tegra Jetson K1 are generally more energy efficient than their server counterparts which is in line with our power results. This is not entirely surprising but suggests that from an energy efficiency standpoint, server compute platforms can still achieve additional gains.

VI. Discussion

Based on our exploration of these data parallel applications and computing platforms, we find that the programmability for each of these platforms is a key distinguishing factor which will influence programmer adoption and widespread deployment. Our results confirm that FPGA and GPU platforms can provide compelling energy efficiency gains over general purpose processors. For data intensive applications, a well optimized GPU implementation (ex. k-means, and kNN) can get comparable performance to an FPGA. However, achieving these energy efficiency gains requires varying levels for programmer effort.

GPU programming frameworks such as CUDA make it much easier to program for GPUs than for FPGAs and, in our experience, a well versed GPU programmer can compose a reasonable design within a week (40hr/week). While there exist higher level languages to Verilog such as Chisel [22] and Bluespec [23] which substantially reduce hardware implementation time, designers must still reason about and design the underlying accelerator architecture in addition to programming driver interfaces to successfully reap the energy efficiency gains of FPGAs. The task of defining the correct hardware design architecture in itself is a non-trivial design space exploration in itself which can easily extend the design, implementation, validation, and deployment time for an FPGA design to weeks or even months for a single developer. Thus in terms of programmer effort, for highly data parallel applications the GPU is likely a better option to target for acceleration than FPGAs.

We find that the key strength of the Intel Xeon Phi coprocessor is its ease of programmability, however it does not yield as promising energy efficiency benefits compared to FPGA, and GPU platforms. Because the Intel Xeon Phi coprocessor supports OpenMP, it is almost trivial to repurpose CPU benchmarks for the Phi and makes the underlying architecture mostly transparent to the programmer. Unfortunately when compared to GPU and FPGA platforms, we find that the energy efficiency of the Intel Xeon Phi coprocessor is about one order of magnitude worse. While we do see run time improvements of up to $3.64\times$, the Intel Xeon Phi coprocessor uses roughly twice the power which results in around $2\times$ energy efficiency at best. We also tested a custom OpenCL implementation against a custom OpenMP implementation for SVM (not LibLinear) but we found that it yielded no significant performance or energy efficiency improvements.

VII. Conclusions

We presented a broad evaluation of contemporary heterogeneous computing substrates across CPUs, GPUs, FPGAs, and the Xeon Phi coprocessor for data intensive applications. Interpreting the results is in the eye of the beholder. In our view, it suggests the Intel Phi coprocessor while easy to program simply does not achieve the energy efficiency gains as other readily available options in practice. Our results also suggest that FPGAs are not really worth the programming effort when a well optimized GPU can achieve the same order of magnitude energy efficiency improvements for data parallel applications. While programming using GPU tools is not trivial, it is significantly easier than fielding a full FPGA design flow. As we reflect back to the effort expended for this study, it seems clear that for data-intensive applications with ample parallelism, a developer should really consider whether it's necessary to port to FPGAs when GPUs can achieve comparable gains.

VIII. Acknowledgments

This work was done in collaboration with Oracle Labs, and the authors would like to thank Sungpack Hong, Michael Delorimier, and Shrikumar Hariharasubrahmanian for their support and guidance on this project.

References

[1] V. Turner, J. F. Ganzt, D. Reinsel, and S. Minton, "The digital universe of opportunities: Rich data and the increasing value of the internet of things," 2014. Accessed: 2016-04-23.

[2] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan, "Heterogeneous Chip Multiprocessors," *Computer*, vol. 38, pp. 32–38, Nov. 2005.

[3] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, (Piscataway, NJ, USA), pp. 13–24, IEEE Press, 2014.

[4] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowl. Inf. Syst.*, vol. 14, pp. 1–37, Dec. 2007.

[5] "7 series fpgas memory resources." http://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf, 2014. Accessed: 2016-4-23.

[6] S. Lloyd, "Least squares quantization in pcm," *IEEE Trans. Inf. Theor.*, vol. 28, pp. 129–137, Sept. 2006.

[7] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theor.*, vol. 13, pp. 21–27, Sept. 2006.

[8] A. Ng, "Cse229 lecture notes." University Lecture. Accessed: 2016-04-02.

[9] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Min. Knowl. Discov.*, vol. 2, pp. 121–167, June 1998.

[10] Y. Freund and R. E. Schapire, "A short introduction to boosting," in *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pp. 1401–1406, Morgan Kaufmann, 1999.

[11] R. Narayanan, B. zs. Ikylmaz, J. Zambreno, G. Memik, and A. Choudhary, "Minebench: A benchmark suite for data mining workloads," in *2006 IEEE International Symposium on Workload Characterization*, pp. 182–188, 2006.

[12] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, 2014.

[13] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *J. Mach. Learn. Res.*, vol. 9, pp. 1871–1874, June 2008.

[14] B. Catanzaro, "kmeans." https://github.com/bryancatanzaro/kmeans, 2013.

[15] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pp. 1–6, June 2008.

[16] M. Moghimi, "liblinear." https://github.com/ucsd-vision/liblinear, 2013.

[17] "Intel xeon processor e5-2620." http://ark.intel.com/products/64594/Intel-Xeon-Processor-E5-2620-15M-Cache-2_00-GHz-7_20-GTs-Intel-QPI. Accessed: 2016-04-21.

[18] "Intel xeon phi coprocessor 5110p." http://ark.intel.com/products/71992/Intel-Xeon-Phi-Coprocessor-5110P-8GB-1_053-GHz-60-core. Accessed: 2016-04-21.

[19] "Specifications." http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x/specifications. Accessed: 2016-04-21.

[20] M. Harris, "Jetson tk1: Mobile embedded supercomputer takes cuda everywhere." https://devblogs.nvidia.com/parallelforall/jetson-tk1-mobile-embedded-supercomputer-cuda-everywhere/, 2014. Accessed: 2016-04-27.

[21] "Nvidia jetson tk1 development kit." http://developer.download.nvidia.com/embedded/jetson/TK1/docs/Jetson_platform_brief_May2014.pdf, 2014. Accessed: 2016-04-27.

[22] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," in *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, (New York, NY, USA), pp. 1216–1225, ACM, 2012.

[23] R. S. Nikhil, "Bluespec system verilog: efficient, correct RTL from high level specifications," in *2nd ACM & IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE 2004), 23-25 June 2004, San Diego, California, USA, Proceedings*, pp. 69–70, 2004.