Machine Learning for Performance and Power Modeling of Heterogeneous Systems

(Invited Paper)

Joseph L. Greathouse
Radeon Technologies Group
Advanced Micro Devices, Inc.
Joseph.Greathouse@amd.com

Gabriel H. Loh
AMD Research
Advanced Micro Devices, Inc.
Gabriel.Loh@amd.com

ABSTRACT

Modern processing systems with heterogeneous components (e.g., CPUs, GPUs) have numerous configuration and design options such as the number and types of cores, frequency, and memory bandwidth. Hardware architects must perform design space explorations in order to accurately target markets of interest under tight time-to-market constraints. This need highlights the importance of rapid performance and power estimation mechanisms.

This work describes the use of machine learning (ML) techniques within a methodology for the estimating performance and power of heterogeneous systems. In particular, we measure the power and performance of a large collection of test applications running on real hardware across numerous hardware configurations. We use these measurements to train a ML model; the model learns how the applications scale with the system's key design parameters.

Later, new applications of interest are executed on a single configuration, and we gather hardware performance counter values which describe how the application used the hardware. These values are fed into our ML model's inference algorithm, which quickly identify how this application will scale across various design points. In this way, we can rapidly predict the performance and power of the new application across a wide range of system configurations.

Once the initial run of the program is complete, our ML algorithm can predict the application's performance and power at many hardware points faster than running it at each of those points and with a level of accuracy comparable to cycle-level simulators.

CCS CONCEPTS

• **Computer systems organization → Heterogeneous (hybrid) systems**; • **Hardware → Power estimation and optimization**; • **Computing methodologies → Machine learning**; *Simulation types and techniques*; Graphics processors;

KEYWORDS

Design exploration, Heterogeneous system design, Performance counters, Performance estimation, Power estimation

ICCAD '18, November 5–8, 2018, San Diego, CA, USA

ACM ISBN 978-1-4503-5950-4/18/11…$15.00
https://doi.org/10.1145/3240765.3243484

ACM Reference Format:
Joseph L. Greathouse and Gabriel H. Loh. 2018. Machine Learning for Performance and Power Modeling of Heterogeneous Systems: (Invited Paper). In *IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN (ICCAD '18), November 5–8, 2018, San Diego, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3240765.3243484

1 INTRODUCTION

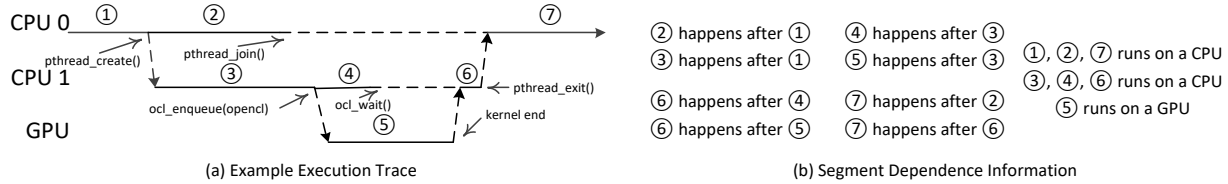
Two of the most important aspects in hardware design are the performance and the power usage of a product. Various market segments trade off computational performance with cost, size, programmability, power, and energy usage. Some of these goals can be statically calculated when defining a product. However, performance and power (and thus energy) can be difficult to estimate because they are a function not just of the hardware, but of the programs and inputs that will be run on the hardware.

Hardware designers can spend a great deal of time simulating their designs to ensure that they will meet market criteria. Unfortunately, highly accurate simulations often run slowly, require an inordinate amount of time to configure, and can only simulate short windows of time. Such low-level simulations are important for validation, catching problems before expensive tape-outs, and testing deep microarchitectural changes. However, they are inadequate for exploring the higher-level system design space.

Even when the microarchitecture of blocks like CPUs, GPUs, and memory controllers are complete, modern hardware designers must combine these pieces into a useful heterogeneous system [4]. These systems may require different high-level design points, depending on the target market and the applications of interest. Table 1 demonstrates multiple heterogeneous processors that are built from similar GPU components. Each of these systems is configured to meet particular market needs. More parallel compute units and higher frequencies and bandwidth may increase performance, but they can also increase area, power, and design complexity.

Table 1 Heterogeneous systems with various high-level design parameters that are all built from similar microarchitectural blocks.

Name	CUs	Max. Freq. (MHz)	Max. DRAM BW (GB/s)
AMD E1-6010 APU[9]	2	350	11
AMD A10-7850K APU[1]	8	720	35
Microsoft Xbox One™ processor[3]	12	853	68
Sony PlayStation 4 processor[3]	18	800	176
AMD Radeon™ R9-280X GPU[2]	32	1000	288
AMD Radeon™ R9-290X GPU[2]	44	1000	352



(a) Example Execution Trace (b) Segment Dependence Information

Figure 1 An example of a how applications may run on a heterogeneous system. Parallel "segments" of the application have some happens-before relationship. A high-level simulation can estimate the performance and power of these segments and find the critical path through the application to estimate total application performance and power.

An important step in the hardware design process is early hardware design space exploration. Analyzing the viability of a heterogeneous system design for applications of interest helps shape the final product and reduces the number of systems that must be studied using more complex low-level simulations. For instance, if a customer's application is found to be heavily bandwidth-bound, high-level explorations will focus design efforts on systems with more bandwidth; computational capabilities might be safely sacrificed to meet cost and power constraints in this case.

This paper describes work done in AMD to build high-level simulation techniques that allow rapid design space exploration of heterogeneous system. We will describe how to use machine learning (ML) techniques to quickly estimate the power and performance of applications under test while varying the high-level design parameters of heterogeneous processors.

We train our ML model by running numerous applications on existing hardware while varying hardware design parameters such as core frequency, memory bandwidth, and number of active parallel processors. While running these applications, we measure the performance and power of the application as well as hardware performance counters; we then use these measurements to ascertain the performance and power scaling curves of these applications. We automatically cluster these curves together into groups of applications that scale like one another and train a neural network to associate performance counter measurements with each cluster.

When we find new applications of interest that we wish to analyze, we can then run it at a single hardware point and measure its performance, power, and hardware performance counters. We can then put these measured performance counters through the neural network to estimate what the performance and power scaling curves of this application would look like. This allows us to rapidly model the performance and power of this application at different hardware configurations, increasing the speed at which we can study designs. This, in turn, allows us to study a wider range of applications when trying to make hardware design decisions.

2 HETEROGENEOUS SYSTEM ANALYSIS

Heterogeneous systems are built from various computational components, such as CPUs and accelerators like GPUs. In this paper, we focus on systems with CPUs and GPUs, but we believe the general methodology presented here could be useful for studying systems with other types of accelerators. In such heterogeneous systems, parallel work is launched to these components, and various parts of the application depend on one another at synchronization points.

At a high level, this paper describes a trace-based simulator that gathers performance and ordering information while running a program-under-test on commodity hardware. These traces are saved for later use in offline power and performance models. The tool then uses the estimated performance and power information, ordering constraints, and a description of the simulated machine to estimate how the program would run on the simulated hardware.

Figure 1 illustrates how an application running on various devices in a heterogeneous system may run. It is possible to divide the application into parallel *segments* with some happens-before relationship between them. These segments may be divided at, for instance, library calls or changes in program phase [13]; segments in the same thread are ordered by their serial execution order, while segments that can run in parallel can be ordered by API calls.

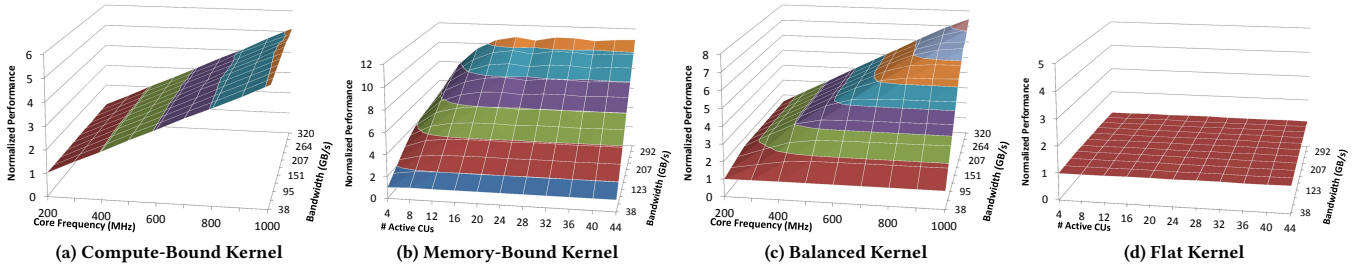
This simulator uses performance and power estimation mechanisms to model how the each of these segments changes with system configuration. Changes in an individual segment may not have proportional effects on the final application; for example, doubling the performance of segment ④ will not affect the total performance of the application, since it is not on the critical path.

We described this system-wide trace-based analysis in more depth at ModSim 2013 [7]. This paper focuses on how we may estimate the performance and power changes of these segments after we have gathered application data from existing hardware. We described in previous works how to estimate performance and power changes caused by high-level CPU design changes [15, 16]. This paper focuses on estimating the power and performance changes of GPU segments as we modify GPU design parameters [19].

3 GPU KERNEL SCALING

GPUs gain much of their performance from running many parallel compute units (CUs) at a moderate frequency in conjunction with a high-bandwidth memory system. Work is often asynchronously launched to the GPU, and threads on the CPU may perform other work in parallel with these GPU *kernels*. GPUs can be configured in a variety of ways to meet performance and power demands; CU counts, operating frequency, and available bandwidth can all vary widely (as shown in Table 1). As such, our first step is to study how GPU kernels scale as we change these parameters; more details about this study were presented at IISWC 2015 [11].

We performed a series of studies on AMD FirePro™ W9100 GPU; we varied the core frequency between 200 MHz and 1 GHz at 100 MHz increments and changed the number of active CUs from 4 to 44, in steps of 4. Similarly, we change the frequency of the GDDR5 connections from 150 MHz to 1250 MHz (inclusive), in 110 MHz



(a) Compute-Bound Kernel (b) Memory-Bound Kernel (c) Balanced Kernel (d) Flat Kernel

Figure 2 These figures illustrate how the performance of kernels that run on a GPU may scale as computational capability (controlled by frequency and number of parallel compute units) and memory bandwidth change.

increments, yielding a range of 38.4-320 GB/s. This allows us to study how the performance of our test applications changed as the hardware configuration was modified. We explored 97 OpenCL™ applications and took measurements for 267 of their kernels.

Figure 2 demonstrates some of the performance scaling curves we observed; we saw similar scaling effects with power. For example, Figure 2a illustrates a traditional compute-bound kernel. As core frequency changes, the GPU can perform more computation per unit time; performance in these kernels scales with this computational throughput, regardless of available bandwidth. In contrast, Figure 2b illustrates a kernel that is almost entirely bound by memory bandwidth. Only when the computational throughput of the chip is very low does it affect this kernel's performance.

Unlike these two straightforward scaling curves, Figure 2c illustrates a slightly more complex curve; in this case, the performance of the kernel can depend on either computational throughput or memory bandwidth, depending on the ratio between the two. Essentially, this curve is sketching out the roofline of the kernel [18]. Finally, Figure 2d shows the performance scaling of a kernel that appears to be poorly suited for running on a GPU; in this case, neither bandwidth nor more CUs helps its performance. Such situations often occur if there is not enough parallelism in the kernel to take advantage of the GPU.

We illustrate these various scaling curves to show that the performance and power of GPU kernels can scale in broadly similar fashions. To quantify this similarity, we automatically categorized the performance scaling behavior of the kernels in our tests. This is inspired by previous benchmark studies [5, 12].

We first constructed a matrix where each row represents the kernel and the columns indicate the normalized performance for each of the hardware configurations. Because this type of matrix is very large, it is difficult for clustering algorithms to draw meaningful conclusions. Motivated by this, we applied principal component analysis (PCA) to reduce the dimensionality of this matrix.

We then ran hierarchical agglomerative clustering (HAC) to generate a dendrogram, which clusters the kernel iterations based on their relative similarity of performance scaling. HAC starts by putting each point in its own cluster and then recursively groups each pair of clusters with minimum inter-cluster linkage distance. We considered Spearman rank correlation coefficient [14] to represent similarity, and Ward's linkage criterion [17] for linkage distance.

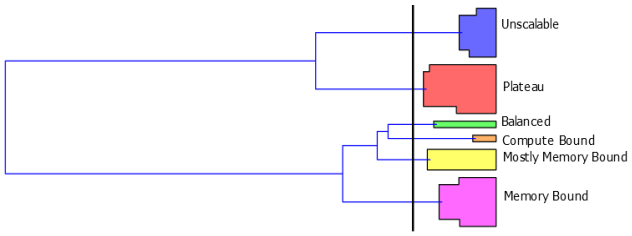


Figure 3 Dendrogram illustrating how many GPU kernels scale similarly to one another. Each cluster contains an automatic classification of scaling curves (like those seen in Figure 2); kernels with similar scaling curves are classified closer together.

Figure 3 illustrates the clusters of scaling curves that we observed in our benchmarks. Benchmarks within each of these clusters generally scale in a similar fashion to one another; we observed similar clustering in power curves. For example, a bandwidth-bound kernel such as sparse matrix-vector multiplication [6] and another bandwidth-bound kernel such as a bitonic sort would both likely scale similarly as hardware parameters are changed.

The next section will describe how we take advantage of this information to build a machine learning model that will automatically identify the scaling cluster a kernel is in without requiring an expensive exploration of the full hardware design space.

4 AUTOMATIC GPU SCALING ANALYSIS WITH MACHINE LEARNING

Based on the studies shown in Section 3, we can see that many GPU kernels scale in a similar fashion to one another. Our goal for the simulator discussed in Section 1 is to automatically estimate the performance and power of a GPU kernel after running it at a single hardware point. Towards that end, we designed a machine learning model that takes as input performance and power measurements plus performance counter readings from running the kernel at a single hardware point. This ML model will then estimate what the performance or power of that kernel would be at a different hardware point [19]. This model is depicted in Fig. 4.

The algorithm used to construct the ML model requires multiple steps. First, we gather performance or power scaling curves across a large number of test kernels. As shown in Figure 5, these training kernels are clustered to form groups of kernels with similar scaling behaviors across hardware configurations. This can be done using

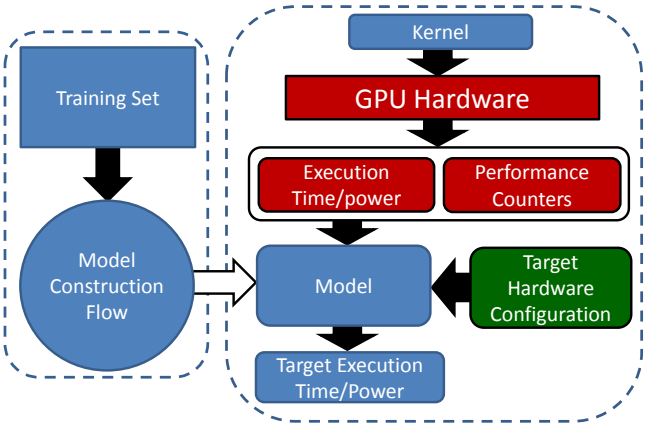


Figure 4 The flow of our model for estimating GPU performance and power. The model is trained over many hardware configurations, but afterwards, we only require one hardware measurement to estimate power and performance for new kernels.

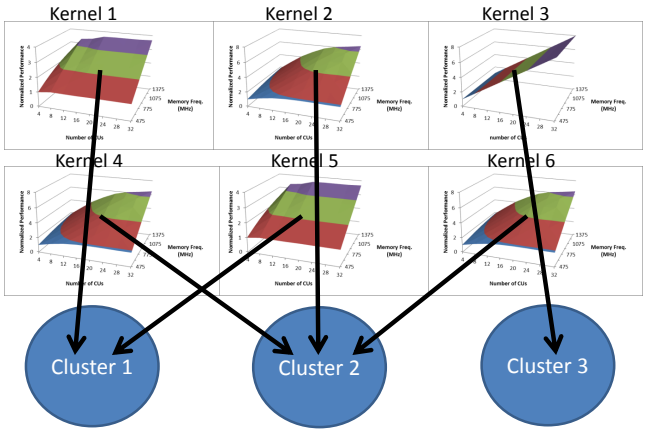


Figure 5 GPU kernels that scale similarly to one another are put into clusters, akin to what is shown in Figure 3

techniques such as HAC, as we showed in Figure 3, or through simpler techniques such as k-means clustering of Euclidean distances between the curves. However this is done, each cluster represents one scaling behavior found in the training set.

After this clustering is complete, we build a classifier to predict which cluster's scaling behavior best describes a new kernel. We use performance counters as input to this classifier; when we run a new kernel, we can simultaneously gather hardware performance counters which describe how it uses the GPU. Kernels in different clusters use the hardware differently, so a classifier can identify a kernel's cluster by its performance counter values.

Figure 6 illustrates such a classifier built using a neural network. In our studies, we used a three layer, fully connected neural network where the input layer is linear and the hidden and output layers use sigmoid functions. We set the number of hidden layer nodes to be equal to the number of output nodes; there is one input node per measured hardware performance counter, and one output node for each performance or power scaling cluster. The network outputs

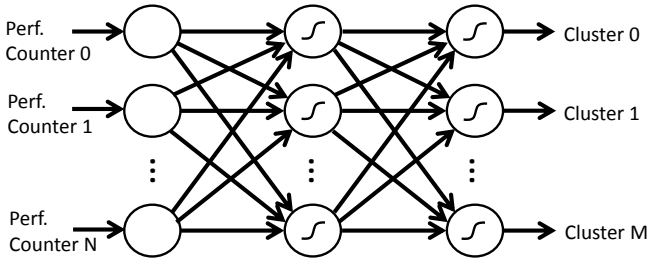


Figure 6 Classifier built using a neural network that allows us to find which performance or power scaling cluster a new kernel matches based on the performance counter measurements of the kernel's execution at one hardware point.

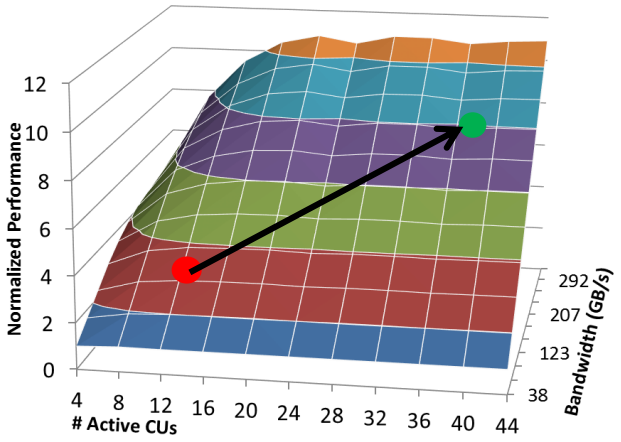


Figure 7 Estimating the performance of a kernel that was originally measured at 12 CUs and 95 GB/s. After finding this curve using the neural network in Figure 6, we estimate the new performance of this kernel at a different hardware configuration (36 CUs and 235 GB/s) by following this scaling curve.

one value, between 0 and 1, per cluster; we choose the cluster with the highest value and estimate that this kernel will scale similarly to the kernels within that cluster.

Once we have estimated which cluster this new kernel-under-test belongs to, we use the resulting scaling curve to estimate the performance or power of this kernel at the desired hardware configuration. This is illustrated in Figure 7; we start at the original hardware point (red) and walk through the scaling curve to estimate the total performance difference that would result from moving to the final hardware configuration.

Overall, this process only requires a small amount of computational effort after gathering the data at our initial hardware configuration. We must perform a small inference calculation through the neural network to estimate how this kernel will scale. After this, estimating the performance (or power) at all other supported hardware configurations only requires us to perform a series of multiplies as we walk through the scaling curve. As such, we can rapidly estimate the performance of even very long-running kernels; this allows us to simulate a large number of hardware points at speeds that can far surpass what we could attain even performing analysis on real hardware.

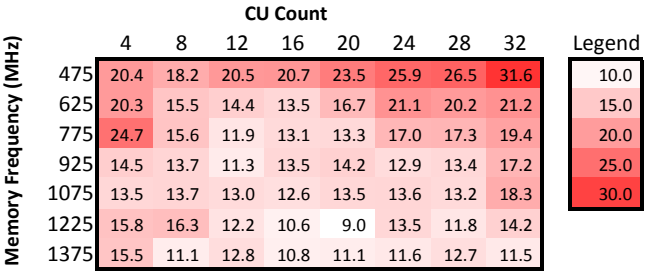


Figure 8 Validation set error heat map at 1000 MHz core frequency. Each point represents the average error of estimating from that point's base configuration to all other configurations (including all other frequencies).

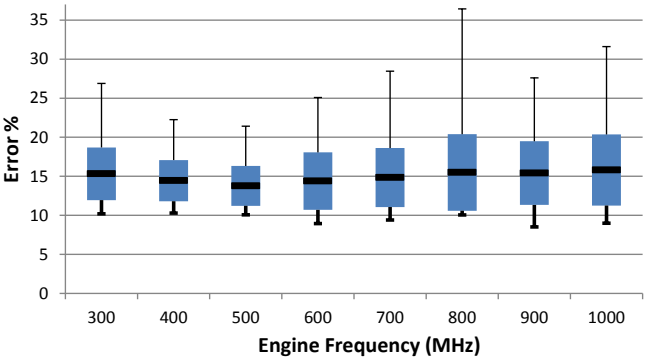


Figure 9 Performance prediction error summarized across all of the base hardware configurations.

4.1 Accuracy of our ML Model

In order to validate the accuracy of our models, we executed 108 OpenCL™ kernels from 49 open source and academic benchmark applications [19]. We ran these on an AMD Radeon™ HD 7970 GPU. We ran all of our benchmarks across a range of eight CU settings (4, 8, …, 32), eight core frequencies (300, 400, …, 1000 MHz), and seven memory frequencies (475, 625, …, 1375 MHz). For each kernel across all of these configurations, we gather the kernel execution time, performance counters, and the average power of that kernel over its execution. The data from a random 80% of the 108 kernels were used to train and construct our ML model, while the data from the remaining 20% were used for validation.

The base hardware configuration (i.e., the single configuration we use to measure our kernel-under-test) is a key parameter of the model and can influence its accuracy. As such, we constructed a model for each of the 448 possible base hardware configurations. We then used all of these models to predict the execution time of each of the 22 validation kernels on each of the model's other 447 possible target hardware configurations; we exclude a model's base hardware configuration from its list of possible target hardware configurations.

A heat-map of validation set error values from 56 models is shown in Fig. 8. All 56 models have a base configuration frequency of 1000 MHz. Their base CU counts and memory frequencies take on all combinations of the eight possible CU counts (4, 8, …, 32) and the seven possible memory frequencies (475, 625, …, 1375).

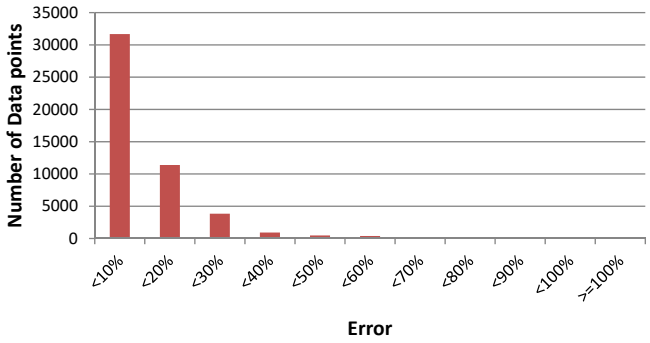


Figure 10 Histogram of the estimation error for our power model.

We do not show engine frequency variation to make it easier to visualize the data and gain insights into how base configuration impacts model accuracy.

Each entry in Fig. 8 is the average error of one model across all validation kernel and target configuration combinations. We can see that the base configuration can have a strong effect on the total accuracy of our model. For instance, the base configurations with the lowest memory frequency (i.e., configurations with little available memory bandwidth) show worse results, especially when the configuration also has more compute units (i.e., more available computational throughput). At these extremely low memory bandwidth configurations, the number of kernels that become memory bottlenecked increase. As the percentage of memory bottlenecked kernels becomes greater, the performance counters values, which are gathered on the base hardware configuration, become less diverse between kernels. This makes it difficult for the classifier to differentiate between kernels that should be in different clusters.

As the compute-to-memory bandwidth resources in the base hardware configuration becomes more balanced, the diversity among the kernel performance counter values increases. As a result, the neural network classifier can more easily distinguish between kernel scaling behaviors and the overall accuracy improves.

Figure 9 shows the summary of our performance prediction accuracy across all of the base hardware configurations. Each box and whisker set shows the distribution of the average error of models with a common base engine frequency. The lines through the middle of the boxes represent the distribution averages. The ends of the boxes represent the standard deviation of the distributions and the ends of the whiskers represent the maximum and minimum of the errors. We can see from these results that the average error of our ML-based performance estimation methodology is roughly 15%, which is in line with the accuracy of many cycle-level simulations [8].

In addition to performance modeling, we applied our ML methodology to create power models. We studied the power model accuracy versus the number of clusters using the same experimental setup used for the performance modeling experiments. A histogram of the power model's errors on the validation set is presented in Figure 10. The power model error rate is lower than the performance model's; it is below 10% for most configurations. In addition, we found that the variability of the error was smaller for power modeling than it was for performance modeling.

5 CONCLUSION

In this work, we have described a methodology for performing high-level performance and power estimation on heterogeneous systems. The goal of this methodology is to allow early, rapid design-space explorations of heterogeneous systems over a large number of design points and test applications. Towards this end, we first described a trace-driven method for splitting apart applications into parallel segments that run on the various heterogeneous devices and that are ordered through application-level semantics. We measure the performance and power of these segments on real hardware. We then try to estimate how the performance and power of these segments would change as we modify high-level hardware configuration parameters.

Previous works showed how to make these estimates on CPUs; this paper describes work we have done to make such estimates viable on GPUs. In particular, we describe a methodology that takes performance and power measurements from a kernel running on a real GPU. We then put these measurements through a machine learning model that will estimate how this kernel's performance and power will scale to different hardware points.

Our ML model is trained on many existing kernels measured on multiple hardware configurations, and it automatically cluster kernels with similar performance and power scaling curves. We then show that, by using a neural network that is trained on the hardware performance counters values associated with the kernels in each cluster, we can automatically recognize what cluster a kernel belongs to from a single hardware measurement.

With this model, we can rapidly estimate the performance and power of GPU kernels across a wide range of hardware configurations. Each estimation only requires a small, fixed amount of computation – this can allow the estimation to occur much faster than the kernel itself. In addition, we demonstrated that the accuracy of these models (10-15% error rate) is roughly in line with published values for academic cycle-level simulators. These performance and accuracy values imply that this model is useful for high-level SoC and heterogeneous system design space explorations.

This ML-based performance estimation model has limitations. It is primarily useful for modeling high-level changes in configuration parameters, such as frequency or bandwidth. It assumes that most of the underlying microarchitecture of the devices stays the same, so it is not a replacement for low-level simulators that can model deep details of the hardware.

Finally, our model requires a large amount of training data. Like most machine learning models, we require a large number of kernels in the training set; the larger this set, the more accurately the model can represent general applications. However, our model also requires a large numbers of measurements *per kernel*, because we must explore the entire hardware configuration space to fill in the scaling curves. As the number of hardware design dimensions increases, this may become prohibitively expensive. We required hundreds of tests across the three dimensions we tested; we could require tens of thousands of tests per kernel if we added more.

Recent works have focused on reducing the number of tests required for such design-space explorations [10]. We feel that such techniques may be useful in reducing the amount of training data needed for machine learning models used in this field.

REFERENCES

[1] Advanced Micro Devices, Inc. 2018. AMD A-Series APU Processors. https://www.amd.com/en/products/a-series-processors-desktop-7th-gen-fm2-plus.
[2] Advanced Micro Devices, Inc. 2018. AMD Radeon™ R9 Series Graphics. http://www.amd.com/en-us/products/graphics/desktop/r9.
[3] Sebastian Anthony. 2013. Xbox One vs. PS4: How the Final Hardware Specs Compare. http://www.extremetech.com/gaming/156273-xbox-720-vs-ps4-vs-pc-how-the-hardware-specs-compare.
[4] David Brooks, Mark Hempstead, Mike Lui, Parnian Mokri, Siddharth Nilakantan, Brandon Reagen, and Yakun Sophia Shao. 2015. Research Infrastructures for Accelerator-Centric Architectures. Tutorial Presented at HPCA.
[5] Shuai Che, J.W. Sheaffer, M. Boyer, L.G. Szafaryn, Liang Wang, and K. Skadron. 2010. A Characterization of the Rodinia Benchmark Suite with Comparison to Contemporary CMP Workloads. In *Proc. of the IEEE Int'l Symp. on Workload Characterization (IISWC)*.
[6] Joseph L. Greathouse and Mayank Daga. 2014. Efficient Sparse Matrix-Vector Multiplication on GPUs using the CSR Storage Format. In *Proc. of the Conference on High Performance Computing, Networking, Storage and Analysis (SC)*.
[7] Joseph L. Greathouse, Alexander Lyashevsky, Mitesh Meswani, Nuwan Jayasena, and Michael Ignatowski. 2013. Simulation of Exascale Nodes through Runtime Hardware Monitoring. In *ASCR Workshop on Modeling & Simulation of Exascale Systems & Applications (ModSim)*.
[8] Anthony Gutierrez, Joseph Pusdesris, Ronald G. Dreslinski, Trevor Mudge, Chander Sudanthi, Christopher D. Emmons, Mitchell Hayenga, and Nigel Paver. 2014. Sources of Error in Full-System Simulation. In *Proc. of the Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*.
[9] Klaus Hinum. 2014. AMD E-Series E1-6010 Notebook Processor. http://www.notebookcheck.net/AMD-E-Series-E1-6010-Notebook-Processor.115407.0.html.
[10] Wenhao Jia, Kelly A. Shaw, and Margaret Martonosi. 2013. Stargazer: Automated Regression-Based GPU Design Space Exploration. In *Proc. of the Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*.
[11] Abhinandan Majumdar, Gene Wu, Kapil Dev, Joseph L. Greathouse, Indrani Paul, Wei Huang, Arjun Karthik Venugopal, Leonardo Piga, Chip Freitag, and Sooraj Puthoor. 2015. A Taxonomy of GPGPU Performance Scaling. In *Proc. of the IEEE Int'l Symp. on Workload Characterization (IISWC)*.
[12] Aashish Phansalkar, Ajay Joshi, and Lizy K. John. 2007. Subsetting the SPEC CPU2006 Benchmark Suite. *SIGARCH Computer Architecture News* 35, 1 (2007), 69–76.
[13] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. 2002. Automatically Characterizing Large Scale Program Behavior. In *Proc. of the Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.
[14] C. Spearman. 1907. Demonstration of Formulæfor True Measurement of Correlation. *The American Journal of Psychology* 18, 2 (1907), 161–169.
[15] Bo Su, Joseph L. Greathouse, Junli Gu, Michael Boyer, Li Shen, and Zhiying Wang. 2014. Implementing a Leading Loads Performance Predictor on Commodity Procesors. In *Proc. of the USENIX Annual Technical Conf. (USENIX ATC)*.
[16] Bo Su, Junli Gu, Li Shen, Wei Huang, Joseph L. Greathouse, and Zhiying Wang. 2014. PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration. In *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*.
[17] Joe H. Ward. 1963. Hierarchical Grouping to Optimize an Objective Function. *J. Amer. Statist. Assoc.* 58, 301 (1963), 236–244.
[18] Samuel Williams, Andrew Watterman, and David Patterson. 2009. Roofline: an Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 52, 4 (April 2009), 56–76.
[19] Gene Wu, Joseph L. Greathouse, Alexander Lyashevsky, Nuwan Jayasena, and Derek Chiou. 2015. GPGPU Performance and Power Estimation Using Machine Learning. In *Proc. of the Int'l Symp. on High Performance Computer Architecture (HPCA)*.