Predictive Modeling for CPU, GPU, and FPGA Performance and Power Consumption: A Survey

Kenneth O'Neal and Philip Brisk
Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA, USA
konea001@ucr.edu, philip@cs.ucr.edu

***Abstract*— CPUs and dedicated accelerators (namely GPUs and FPGAs) continue to grow increasingly large and complex to support todays demanding performance and power requirements. Designers are tasked with evaluating the performance and power of similarly increasingly large design spaces during pre-silicon design for CPUs and GPUs to reduce time-to-market and limit manufacturing costs, or to figure out how to best map applications onto FPGAs using high-level synthesis tools. Typically, cycle-accurate simulators are used to evaluate workloads for pre-silicon CPUs and GPUs and to avoid the overhead of synthesis and place-and-route when targeting FPGAs; however, simulators exhibit prohibitively long run times that limit the number of design points and workloads that can be evaluated in a reasonable timeframe.**

This survey focuses on predictive modeling as an alternative to cycle-accurate simulation, which enables rapid evaluation of workloads and design points. When applied properly, predictive modeling can improve time to market, and can facilitate more comprehensive design space explorations with far less overhead than simulation. The survey focuses on predictive models applied to CPUs, GPUs, and FPGAs, noting that the general approach has been applied to many other computing platforms as well.

Keywords— CPU, GPU, FPGA, Predictive Model, Machine Learning, Accuracy, Error, Survey

I. Introduction

CPUs and GPUs must be designed and tested before manufacturing. FPGA-based accelerators aren't manufactured, but design efforts are impeded by long simulation and synthesis times. In both cases, designers must meet *quality of service* (QoS) requirements which dictate the requisite performance and power consumption of the design under test. CPU and GPU Cycle-accurate *instruction set simulators* (ISS's) [1] are used to estimate performance and power consumption and software simulators are used by commercial FPGA synthesis tools to obtain early estimates of design performance. In all cases, the simulators are used to perform *design space exploration* (DSE), and to prototype and functionally verify new architectures. CPU and GPU simulators also support co-optimization of hardware and software (e.g., API, firmware, and driver development). Typical software simulators execute orders of magnitude slower than native execution on commercial hardware [2], too slow to meet modern design productivity demands, despite efforts to raise the abstraction level [3], parallelize the simulations [4], and leverage hardware assistance [5] to improve simulation speed.

Predictive modeling has emerged as one potential solution to this conundrum. Although less precise than cycle-accurate simulation, predictive models can be evaluated faster, thereby increasing both the number of design points that can be explored and the number of workloads evaluated per design point. This situation shares some similarities to the introduction of statistical sampling into cycle-accurate simulators [6], [7]: in both cases, the productivity benefits that accrue from modifying existing methodologies outweigh the resulting loss in accuracy.

This paper surveys recent advances in predictive modeling targeting CPUs (Section II), GPUs (Section III), and FPGAs (Section IV), with specific emphasis on cross-platform models based on machine learning. Although cycle-accurate simulation is required for CPU and GPU model training and is explicitly leveraged as model input for DSE targeting FPGA-based accelerators, predictive modeling can and should reduce the amount of simulation and synthesis required once a model is deployed. This provides a substantial productivity advantage compared to existing design and synthesis methodologies for these targets.

II. CPU Modeling

Table I summarizes the characteristics of each CPU modeling approach and weighs their benefits and drawbacks.

A. Statistical models for CPUs

The majority of work on predictive modeling targets CPUs executing general-purpose workloads. The overall objective is to limit the number of simulations required to evaluate design points in a much larger architectural design space. For example, Ipek et al. [8] consider a design space comprising 250K design points, and sample a representative subspace (~20K points, or 1-2% of the total design space), using active learning techniques. Model training leverages iterative refinement to build an ensemble of *artificial neural networks* (ANNs), in which each represents one of a 10-fold *cross-validation* (CV) process. Error rates range from 2-5%, with mod els created for individual CPUs, a multiprocessor, and the memory subsystem.

Similarly, Lee et al. [9] train regression models to predict performance and power consumption of a large number of design points (22 billion) using a representative sub-sample (4000), obtained via *uniform at random* (UAR) sampling. The regression models include linear least-squares models as well as non-linear spline functions, in which the predictive function is decomposed into multiple piecewise polynomials. Application-specific performance models achieve an average error of 4.1%, while regional power models, in which applications are grouped by feature similarity, achieve 4.3% average error.

This work was supported in part by NSF Award #152181.

2159-3477/18/$31.00 ©2018 IEEE
DOI 10.1109/ISVLSI.2018.00143

TABLE I. CPU POWER AND PERFORMANCE PREDICTIVE MODELING COMPARISON.

Paper	Model	Features	Accuracy (avg.)	Speed	Benefit	Drawback
[8]	Ensemble of ANNs	Architectural parameters and latencies	Performance: 95-98%	One order of magnitude fewer simulations required	Predictive models are trained on a subset of design points to predict the full space	Requires detailed architectural expertise and many simulations to characterize latencies.
[9]	Linear and piecewise cubic spline models	Architectural parameters and latencies	Performance: ~95.9% Power: ~95.6%	~7 orders of magnitue fewer simulations required	Predictive models are used to avoid exhaustive simulation	Requires detailed architectural expertise and many simulation to characterize latencies
[10]	Analytical model	Architectural parameters and throughput measurements	Power: ~95.3%	NA	Performance feedback after model training reduces energy and avoids thermal throttling	Performance overhead; Hand-tuned analytical modeling requires expert knowledge to select features and derivce equations; this limits portability.
[11]	PCA with Lasso and CLSLR	Host CPU performance counters collected once per workload	Lasso: ~83-73% CLSR: ~ 99% For total workload	Near-native direct execution on CPU host (~500 MIPs)	***Cross-platform***; executes faster than simulator or instrumented workload	End-to-end prediction sacrifices accuracy when compared to the phase-level approach. [12]
[12]	Modified Lasso	Host CPU performance counters collected once per phase	Performance and Power: >90% at phase boundaries	Near-native direct execution on CPU host (~500 MIPS)	***Cross-platform***; models power and performance; high fidelity phase-level predictions	Choice of phase-granulartiy impacts accuracy and speed; some parameter tuning is required to balance accuracy vs. overhead.

D*ynamic thermal management* (DTM) often employs power models to reduce energy consumption and prevent thermal emergency events. For example, Nath et. al. [10] developed an analytical multi-core power model for the Intel *Knights Ferry* (KNF) architecture, comprising static and dynamic models for compute, memory, and interconnect power, with a 4.73% average error rate. The model relies on expert knowledge to identify the most relevant performance counters for inclusion as features; the model itself is trained using the selected features and the HotSpot thermal simulator [13] configured to model KNF thermal properties. The model itself has low overhead and reduces energy consumption by 14%, and the occurrence of thermal emergency events by 58%.

B. Cross-Platform statistical models for CPUs

LACross [11] appears to be the first work to perform cross-platform and cross-ISA CPU performance prediction. The initial iteration runs at near-native hardware speeds, and accurately predicts the performance of 157 ACM *International Collegiate Programming Contest (*ACM-ICPC) programs from a variety of application domains. LACross used two commercially available processors as *host* and *target* interchangeably, an Intel Core-i7 920 with 24GB DRAM and the AMD Phenom II X6 1055T with 8GB DRAM. LACross first executes each workload on the host to collect performance counter measurements and executes each workload on the target to measure performance. The host performance counter measurements, one read per workload, are used to train two regression models, the *LASSO L1 regularization model (LASSO)* [14] and the *Constrained Locally Sparse Linear Regression Model* (CLSR), each combined with *Principle Components Analysis* (PCA) [15] to extract latent semantics in the feature data, which improves model accuracy. The average LASSO cross validation (CV) error, an estimate of model generalizability, was ~17% and ~27% for the two respective targets, while the CLSLR model, achieved a much lower CV error of less than 1%, on average, which suggests the existence of a non-linear relationship between host performance counters and target ISA. Although the average model prediction error is not reported, it is notably higher than the reported CV error. Due to the one-time counter collection, Lasso accuracy suffers due to a relatively limited number of data points.

A subsequent extension to LACross [12] used a compiler to instrument program basic blocks, which exposed program phases at a much finer granularity than end-to-end execution. The user specifies the phase granularity: finer granularity increases the profiling overhead, but improves accuracy; lower granularity is faster, but has lower accuracy. Program counter values are collected once per phase, and each phase is treated as a data point. This work also employs an Intel Core-i7 920 as a host and ARM Cortex-series processors targets, representing not only cross-ISA prediction, but also demonstrating the ability to predict the performance and power of embedded targets using a desktop CPU as a host. The average error reported was less than 10% in each case. The viability of cross-platform performance prediction has profound implications for future CPU design methodologies. At present, architectural DSE necessitates the use of a simulator to characterize workload performance and/or power consumption at each design point. High simulation execution times limits both the number of design points that can be explored, and the number of workloads that could be used to characterize each design point. A cross-platform predictive model which uses a commercially available CPU as the host and targets a simulator could significantly increase the throughput of next-generation DSE processes. Another practical consideration is that analytical models require expert knowledge to design,

TABLE II. GPU POWER AND PERFORMANCE PREDICTIVE MODELING COMPARISON.

Paper	Ut the k	Features	Accuracy (avg.)	Speed	Benefit	Drawback
[16]	K-means clustering and ANNs	Performance counters	Performance: ~85% Power: ~90%	20% of target executions eliminated	Power and perf. models used to avoid executing all design points.	Large percentage of design points required to train model
[17]	Forward stepwise regression	Workload characteristics that expose inherent GPU-compatible parallelism	Kepler Performance: ~64% Maxwell Performance: ~73%	Static analysis incurs 10x-20x slowdown over native hardware	**Cross-platform;** Predict GPU performance from CPU features	High model error.
[18]	Analytical model	Single-threaded CPU memory and computation traces and latencies	Jetsen TK1 Performance: ~ 91%	CPU execution plus code instrumentation and PTX transform time	**Cross-platform;** Estimate GPU performance from CPU C code..	Input sizes must be chosen to limit insturmetation overhead.
[19]	Random Forest regression	Functional simulation execution statistics	Intel Skylake GPU Performance:~85.7%	~328x faster than cycle-accurate simulation	**Cross-abstracton;** Host and target use same SW stack.	Functional simulatior instrumetation overhead
[20]	Ordinary Least Squares with feature selection and Random Forest regression	Prev. generation performance counters and API metrics	Boradwell GT2/GT3 Performance: ~93% Skylake GT3 Performance: ~91%	29,000x - 44,000x faster than cycle-accurate simulation	**Cross-generation;** Predict pre-silicon next-generation GPU performance.	Cannot directly account for large-scale architectural. changes

while purely statistical models, such as those employed by LACross, can be derived automatically.

III. GPU MODELING

Table II summarizes the characteristics of each GPU modeling approach and weighs their benefits and drawbacks.

A. Predictive modeling for GPGPUs

Predictive modeling for GPUs is largely inspired by CPU approaches. These techniques are primarily leveraged for DSE wherein models are trained to avoid exhaustive simulation on the target platform.

One such example is a paper that uses ANNs to predict performance and power of OpenCL applications as architectural parameters of the GPU target scale [16], thereby avoiding a more costly exhaustive enumeration. Architectural parameters that are considered include core frequency, memory bandwidth, and the number of available *compute units* (CUs). The model clusters kernels with similar scaling behavior, a-priori, via k-means clustering; the scaling trend is a model that predicts the performance and power consumption of a workload from hardware performance counter measurements. An ANN classifier is trained to predict the cluster of scaling trends that a previously unseen workload most closely matches; given the classification, the scaling trend predicts the performance and power consumption of that workload. The new workload is simulated using one parameter combination to obtain a baseline that can be scaled according to its ANN-predicted trend.

B. Cross-architecture modeling for GPUs

Cross-architecture models for GPUs can be categorized as follows: 1) *cross-platform* models, in which features obtained from a non-GPU host (e.g., a CPU) are used to predict GPU performance and/or power consumption; 2) *cross-abstraction* models, in which a functional simulator host provides features that predict the performance and/or power consumption of a target cycle-accurate simulator; and 3) *cross-generation* models, in which direct execution on an older GPU provides features that predict the performance and/or power consumption of a newer GPU within the same family, possibly at the pre-silicon stage of development, where only a cycle-accurate simulator is available.

XAPP [17] is a suite of cross-platform models that predict the degree of speedup or slowdown that would result from porting a C/C++ CPU workload to CUDA and executing it on a GPU. XAPP instruments program binaries to produce a set of microarchitecturally independent features, which are selected to represent characteristics that correlate with typical GPU execution behavior, and, by extension, performance. Workload characteristics are measured using MICA [21] or PIN [22] and capture characteristics such as instruction level parallelism, shared memory bandwidth, memory throughput, memory coalescing, and bank conflicts in shared memory, among others. XAPP collects 17 features in total, which are converted into a training set ensemble via random bootstrap sampling with replacement [23]. For each training set, XAPP trains a least-squares regression model that includes higher-order polynomial terms to capture non-linear relationships. For a new workload, the ensemble model reports the mean prediction of all models as the final predicted value. XAPP reported 36% average performance prediction error on a Nvidia Kepler GTX 660Ti, and 27% average performance prediction error on a Maxwell GTX 750.

CGPredict [18] is another cross-platform model that collects features from single-threaded non-optimized C code to predict the performance of CUDA code running on an embedded GPU, such as the Jetson TK1 Kepler. CGPredict employs an analytical model whose primary characteristics are based on the interaction between microarchitectural parallelism and memory access latencies and parallelism. CGPredict instruments source code via the compiler, and collects computation and memory traces, which are transformed during a memory behavior analysis stage,

converting single-threaded CPU memory accesses to reflect GPU memory accesses and cache configurations. A subsequent computational analysis converts the computation trace to Parallel Thread Execution (PTX) format, which is specific to Nvidia GPUs. These transformed streams are coupled with estimated GPU computation and cache access latencies, which were derived from repeated execution of micro-benchmarks. A comprehensive analytical model predicts performance from the modified streams, achieving a relatively low predicted error of 9% across 15 kernels. The data input size for each benchmark must be carefully chosen to avoid excessive instrumentation and transformation latencies, while remaining long enough to realistically stress the GPU's compute and memory resources.

O'Neal et al. [19] use features obtained from a lightly instrumented functional GPU simulator host [24] to train a *Random Forest* (RF) model to predict the performance reported by a much slower cycle-accurate GPU simulator target. The objective is to enable rapid evaluation of multiple workloads during pre-silicon design, while ensuring that the host and target share identical software configurations (API, driver, etc.), which themselves may be pre-release. The functional simulator and instrumentation layer produce 69 features; using the RF model is 327.8x faster than cycle-accurate simulation, with a reported 14.34% average out-of-sample error; the instrumentation layer improves overall accuracy by 38% in comparison to the otherwise unmodified functional simulator.

HALWPE is a cross-generation GPU model that obtains performance counter and DirectX API measurements via direct execution on a commercially available GPU, and trains a suite of machine learning models to predict the performance of a cycle-accurate simulator configured to model a pre-silicon successor in the same family [20]; in addition to architectural evolution, there may be unavoidable differences in the software configuration, unlike the functional to cycle-accurate GPU simulator prediction scheme described above. The models, once again, include a suite of linear models, along with one non-linear model: Random Forest regression. For each host-target prediction scenario, the entire suite of models is retrained and the one with the smallest out-of-sample error is selected for use. The host is an Intel Haswell HD4600 single-slice GPU with 20 *execution units* (EUs); three targets are selected: a single-slice (24 EU) Broadwell GT2, a dual-slice (48 EU) Broadwell GT3, and a single-slice (48 EU) Skylake GT3. The Broadwell targets are a single-generation difference over the Haswell host, with varying parallelism, while the Skylake targets represent a two-generation difference. The reported out-of-sample errors and speedups over cycle-accurate simulation are, respectively, 7.45% and 29,481x (Broadwell GT2), 7.47% and 43,643x (Broadwell GT3), and 8.91% and 44,214x (Skylake GT3). One limitation of this approach it is impossible to train models that encompass large-scale cross-generation architectural redesigns that introduce new features that do not correlate with features collected from the post-silicon host.

IV. FPGA Predictive Modeling and DSE

Table III summarizes the characteristics of several FPGA-based predictive modeling approaches, in the context of larger DSE frameworks and weighs their benefits and drawbacks.

FPGAs are often used as acceleration engines for parallel and streaming workloads, due to their inherent reconfigurability. FPGA accelerator design methodologies have traditionally been based on hardware design, and has more recently transitioned to *High-Level Synthesis* (HLS) [25,26]. Although HLS aims to be fully automatic, the typical designer is tasked with the problem of finding the correct combination of parameters (e.g., unroll factor, pipelining depth, etc.), which is a form of DSE. Due to long HLS times, direct evaluation of each design point is infeasible, which necessitates a turn toward modeling.

Liu et al. [27] explore predictive modeling for DSE via *Transductive Experimental Design* (TED) [28], which identifies and samples representative microarchitectural design points. These training sets are then used to build an RF regression model which is iteratively refined via repeated training and synthesis of additional directive permutations, wherein the training sets are updated after evaluating the prior trained model's accuracy. When used in conjunction with TED, iterative refinement improves prediction accuracy and identifies the *Pareto Optimal* set of design points, as measured using the *average distance from reference set* (ADRS) [29]. The user can specify an HLS budget (the maximum number of HLS synthesis run) and report Pareto Optimal design points using up 20 training workloads for budgets of up to 50 runs, and as few as 10 workloads for larger budgets up to 120 runs; this represents a reduction of more than half of the 242 total directive combinations in the search space. This model requires HLS-in-the-loop due to the feature choice (post-HLS design specifications) and iterative improvement techniques (HLS is used to verify performance improvement).

Koeplinger et al. [30] present a DSE methodology that repeatedly calls a bespoke HLS tool with inherent predictive modeling capabilities. Their models are created from C/C++ descriptions which the programmer has annotated with pragmas to indicate design patterns such as map, reduce, filter and groupBy [31]. The framework analyzes the annotated source code and performs transformations such as loop fusion and tiling and converts the program to a more precise specification called *Delite Hardware Definition Language* (DHDL). The DHDL specification is converted to a set of parameterizable architectural templates, which account for the FPGA's on-chip resources (LUTs, BRAMs, routing, etc.) and off-chip memory bandwidth. Template parameters determine tiling sizes, parallelization factors, and coarse-grain pipelining, which are used alongside cycle-count and area utilization estimators to identify the Pareto Front of many design points. To construct the estimators, it's to necessary first characterize the board's various runtime latencies and resource availability using full synthesis runs, averaging 6 runs per template across several parameter configurations; this yields analytical models for utilization and cycle estimation for each template. The models are trained using ANNs with 200 design samples to compensate for the DHDL models, which do not capture on-board utilization. For six workloads consisting of millions of possible design points, they achieve average cycle estimation error of 6.1% and LUT, DSP, and BRAM utilization estimates with 4.8%, 7.5% and 12.3% error respectively, while running 6,533x faster than DSE using Vivado HLS alone. While expedient, this technique requires extensive source code instrumentation, and is only compatible with applications that use standard design patterns.

TABLE III. FPGA HLS PREDICTIVE MODELING COMPARISON

Paper	Model	Features	Accuracy (avg.)	Speed	Benefit	Drawback
[27]	Random Forest regression	HLS design details	>99% ADRS from Pareto-optima.	2-4x fewer HLS runs required.	Avoids exhaustive enumeration of HLS design space	Requires repeated HLS calls to improve model accuracy, limiting speedup
[30]	Performance: Analytical Utilization: Hybrid Analytical + ANNs	Architectural template parameters that capture parallelism	Performance: ~94% Utilization: 88%-95%	279x - 6333x faster than Vivado HLS.	Faster DSE by replacing HLS with predictive models	Requires HLS to characterize target FPGA; requires non-traditional HLS flow
[32]	Analytical	Source-level instrumentation and HLS simulation	Performance: ~99% (compute-bound workloads); ~95% (memory-bound workloads)	~2 orders of magniute faster than target FPGA bitsream generation	Highly accurate; avoids HLS and target FPGA bitstream generation	Requires HLS simulation and board characterization; Vivado simulation is a performance bottleneck.
[33]	Performance: Anaytical Utilization: Gradient Boosted Machine model	HLS directives and workload features from compiler instrumentation.	Performance: ~88% Utilization: 81-87%	2 - 3 orders of magnitude faster than target FPGA bitstream generation	Avoids HLS and target FPGA bitstream generation after model training and target FPGA device characterization	Requires custom microbenchmarks to characterize FPGA resource characeristics; higher instrumentation overhead than other approaches

HLScope+ [32] uses C code instrumentation and analytical modeling to improve the accuracy of the Vivado HLS simulator, adding the capability to handle input-dependent loop bounds. The instrumentation includes *dependency analysis*, which identifies independent regions of code to parallelize and *loop analysis*, which estimates loop cycle counts, culminating in execution cycle counts and DRAM transaction counts for each code module. This helps HLScope+ identify the application's critical execution path, from which it estimates the per-module stall rate. To compensate for the inaccuracy inherent to static analysis, loop analysis is extended with an analytical model.

HLScope+ accounts for DRAM bandwidth and latency by creating a high-level external memory module, which abstracts away individual memory accesses while accounting for resource contention among *processing elements* (PEs); this model is more accurate than the Vivado HLS simulator, which optimistically assumes that memory can be fetched each cycle and ignores memory bandwidth and contention among PEs. An analytical model is created to predict memory access time and can be combined with the compute cycle count to estimate execution time. HLScope+ is evaluated using 14 HLS workloads, resulting in an 1.1% average error for compute-bound workloads and 5.0% average error for memory bound workloads. This is much faster than fully synthesizing each workload; however, the key drawback is that HLScope+ employs analytical models which require expert knowledge of the target FPGA and its memory interface and are not transferrable from one target to another.

MPSeeker [33] performs DSE using HLS simulation in conjunction with C/C++ source code instrumented at the LLVM IR level, HLS design directives, and both predictive and analytical modeling techniques. MPSeeker's DSE considers several design directives including tile sizes, the number of PEs, loop unrolling, loop pipelining, and array partitioning. MPSeeker extends a single-PE analytical model [34] with the ability to estimate cycle counts for multi-PE designs that encompass coarse-grained parallelism. MPSeeker also includes an ensemble tree predictor called the *Gradient Boosted Machine* (GBM) [35], which uses 14 program features and user-defined parallelism directives to predict FPGA resource usage.

MPSeeker's profiler accepts C source code (tiled nested loop structure), FPGA resource constraints, and user-supplied directive settings (tile size, loop unrolling, pipelining, and array partitioning) to produce features. The features track key workload characteristics correlated to parallelism and memory subsystem behavior and are input to Lin-Analyzer to predict performance and the GBM model to predict resource utilization. A subsequent analytical equation combines the outputs of the two models to account for further FPGA resource restrictions, which limit the number of PEs that can execute concurrently.

MPSeeker uses 10 microbenchmark kernels to ascertain the degree of loop resource consumption, communication interface behavior, and other similar properties, along with 5 larger benchmarks for evaluation; each kernel has 280 unique design configurations. MPSeeker's DSE procedure achieves 90% of the Pareto optimal performance, while running ~421x to ~4308x faster than FPGA bitstream generation. Lin-Analyzer reports an average error of 12.8%, while the GBM model reports average errors of 13.2%, 14.7%, 12.7%, and 19.4% for LUTs, flip-flops, DSP blocks, and BRAMs respectively. The reliance on expert-developed microbenchmarks to characterize FPGA resource constraints and the necessity to execute instrumented source code on the simulator are notable drawbacks of this approach.

Cross-architecture prediction (e.g., CPU host to FPGA target) could overcome many of these shortcomings. Doing so would allow for the creation of automatically derived statistical regression models that automatically select requisite features from direct execution on the host, eliminating the need for expert guidance and increasing portability. Given that cross-platform predictive models for both CPUs and GPUs has been successful, they can and should be applied to DSE for FPGAs as well.

V. CONCLUSION

Predictive modeling has immense potential but has not yet been fully integrated into the CPU or GPU architectural design

process; on the other hand, predictive models are widely used in design exploration for FPGAs, but often rely on repeated calls to HLS tools and/or non-portable analytical modeling efforts. On the CPU/GPU side, the next step is to evaluate the potential of cross-platform modeling as an alternative to cycle-accurate simulation, especially during early design stages. In contrast, cross-platform models for FPGAs have not yet been demonstrated; however, once this is accomplished, DSE tools that target FPGAs will benefit significantly. Long-term, there is still a considerable amount of work to be done to convince the research community that predictive models are more than an intellectually meritorious curiosity and have the capability to significantly increase designer engineering productivity.

REFERENCES

[1] Q. Guo, T. Chen, Y. Chen, and F. Franchetti, "Accelerating architectural simulation via statistical techniques: A Survey," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 35, no. 3, pp. 433–446, Mar. 2016.

[2] J. E. Miller *et al.*, "Graphite: A distributed parallel simulator for multicores," in *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, 2010, pp. 1–12.

[3] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *SC '11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, p. 1.

[4] D. Sanchez and C. Kozyrakis, "ZSim: fast and accurate microarchitectural simulation of thousand-core systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture - ISCA '13*, 2013, vol. 41, no. 3, pp. 475–486.

[5] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer, "HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, 2011, pp. 406–417.

[6] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling," in *30th Annual International Symposium on Computer Architecture, 2003. Proceedings.*, pp. 84–95.

[7] E. Perelman *et al.*, "Using SimPoint for accurate and efficient simulation," in *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems - SIGMETRICS '03*, 2003, vol. 31, no. 1, p. 318.

[8] E. Ïpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz, "Efficiently exploring architectural design spaces via predictive modeling," *ACM SIGPLAN Not.*, vol. 41, no. 11, p. 195, Oct. 2006.

[9] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems - ASPLOS-XII*, 2006, vol. 40, no. 5, p. 185.

[10] R. Nath, D. Carmean, and T. S. Rosing, "Power modeling and thermal management techniques for manycores," in *2013 IEEE Symposium on Computers and Communications (ISCC)*, 2013, pp. 000740–000746.

[11] X. Zheng, P. Ravikumar, L. K. John, and A. Gerstlauer, "Learning-based analytical cross-platform performance prediction," in *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2015, pp. 52–59.

[12] X. Zheng, L. K. John, and A. Gerstlauer, "Accurate phase-level cross-platform power and performance estimation," in *Proceedings of the 53rd Annual Design Automation Conference on - DAC '16*, 2016, pp. 1–6.

[13] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: modeling and implementation," *ACM Trans. Archit. Code Optim.*, vol. 1, no. 1, pp. 94–125, Mar. 2004.

[14] R. Tibshirani, "Regression shrinkage and selection via the Lasso," *J. R. Stat. Soc. Ser. B*, vol. 58, pp. 267–288, 1996.

[15] J. Shlens, "A Tutorial on principal component analysis," Apr. 2014.

[16] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "GPGPU performance and power estimation using machine learning," in *Proceedings of HPCA 2015*, 2015, pp. 564–576.

[17] N. Ardalani, C. Lestourgeon, K. Sankaralingam, and X. Zhu, "Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance," *Proc. 48th Int. Symp. Microarchitecture - MICRO-48*, pp. 725–737, 2015.

[18] S. Wang, G. Zhong, and T. Mitra, "CGPredict: embedded GPU performance estimation from single-threadedaApplications," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 1–22, Sep. 2017.

[19] K. O'Neal, P. Brisk, A. Abousamra, Z. Waters, and E. Shriver, "GPU performance estimation using software rasterization and machine learning," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 1–21, Sep. 2017.

[20] K. O'Neal, P. Brisk, E. Shriver, and M. Kishinevsky, "HALWPE: hardware-assisted light weight performance estimation for GPUs," in *Proceedings of the 54th Annual Design Automation Conference 2017 on - DAC '17*, 2017, pp. 1–6.

[21] K. Hoste and L. Eeckhout, "Comparing benchmarks using key microarchitecture-independent characteristics," in *2006 IEEE International Symposium on Workload Characterization*, 2006, pp. 83–92.

[22] C.-K. Luk *et al.*, "Pin: building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation - PLDI '05*, 2005, vol. 40, no. 6, p. 190.

[23] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug. 1996.

[24] "OpenSWR — Gallium 0.4 documentation." [Online]. Available: http://gallium.readthedocs.io/en/latest/drivers/openswr.html. [Accessed: 30-Apr-2018].

[25] Xilinx, "Vivado high-level synthesis." [Online]. Available: https://goo.gl/2kpNwy. [Accessed: 30-Mar-2018].

[26] A. Canis *et al.*, "LegUp : high-Level synthesis for FPGA-based processor / accelerator systems," *FPGA '11 Proc. 19th ACM/SIGDA Int. Symp. F. Program. gate arrays*, pp. 33–36, 2011.

[27] H.-Y. Liu and L. P. Carloni, "On learning-based methods for design-space exploration with High-Level Synthesis," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–7.

[28] K. Yu, J. Bi, and V. Tresp, "Active learning via transductive experimental design," in *Proceedings of the 23rd international conference on Machine learning - ICML '06*, 2006, pp. 1081–1088.

[29] G. Palermo, C. Silvano, and V. Zaccaria, "ReSPIR: A response surface-based pareto iterative refinement for application-specific design space exploration," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 28, no. 12, pp. 1816–1829, Dec. 2009.

[30] D. Koeplinger *et al.*, "Automatic generation of efficient accelerators for reconfigurable hardware," in *ACM SIGARCH Computer Architecture News*, 2016, vol. 44, no. 3, pp. 115–127.

[31] A. K. Sujeeth *et al.*, "OptiML: An implicitly parallel domain specific language for machine learning," in proceedings of the 28th internaional conerence on machine learning., *ICML*, 2011.

[32] Y. Choi, P. Zhang, P. Li, and J. Cong, "HLscope+: fast and accurate performance estimation for FPGA HLS," in *Proceedings of the 36th International Conference on Computer-Aided Design*, 2017, pp. 691–698.

[33] G. Zhong, A. Prakash, S. Wang, Y. Liang, T. Mitra, and S. Niar, "Design space exploration of FPGA-based accelerators with multi-level parallelism," in *Proceedings of the 2017 Design, Automation and Test in Europe, DATE 2017*, 2017, pp. 1141–1146.

[34] G. Zhong, A. Prakash, Y. Liang, T. Mitra, and S. Niar, "Lin-analyzer: a high-level performance analysis tool for FPGA-based accelerators," in *Proceedings of the 53rd Annual Design Automation Conference on - DAC '16*, 2016, pp. 1–6.

[35] C. Click et al., "Gradient boosting machine (GBM) — H2O 3.18.0.8 documentation," 2016. [Online]. Available: https://goo.gl/viLQGw. [Accessed: 01-May-2018].