HALWPE: Hardware-Assisted Light Weight Performance Estimation for GPUs

Kenneth O'Neal Philip Brisk
Department of Computer Science and Engineering
University of California, Riverside

Emily Shriver Michael Kishinevsky
Strategic CAD Labs
Intel Corporation

ABSTRACT

This paper presents a predictive modeling framework for GPU performance. The key innovation underlying this approach is that performance statistics collected from representative workloads running on current generation GPUs can effectively predict the performance of next-generation GPUs. This is useful when simulators are available for the next-generation device, but simulation times are exorbitant, rendering early design space exploration of microarchitectural parameters and other features infeasible. When predicting performance across three Intel GPU generations (Haswell, Broadwell, Skylake), our models achieved low out-of-sample-errors ranging from 7.45% to 8.91%, while running 30,000-45,000 times faster than cycle-accurate simulation.

CCS CONCEPTS

- **Hardware** -> Electronic Design Automation;

Keywords

GPU, DirectX, Render Pipeline, Predictive Modeling

1. INTRODUCTION

Cycle-accurate simulation run-times for highly-threaded processors, such as GPUs, are becoming untenable. Industry simulators serve a dual-purpose of performance simulation and RTL performance validation, and thus offer greater detail and higher accuracy at the cost of longer runtimes compared to their academic counterparts. Ever-increasing simulation times are prohibitive for early-stage GPU design space exploration when many perturbations to the design must be considered.

This paper presents *Hardware-Assisted Light Weight Performance Estimation (HALWPE)*, a methodology that uses existing platform (host) and machine learning to predict the performance of future devices under development (targets). Our experiments, which focus on GPUs, uses as a host the Intel HD 4600 integrated GPU of the three targets (future-generation GPUs), achieving 7-9% average out-of-sample-error, while respectively running 30,000-45,000 faster than an industrial cycle-accurate GPU simulator.

To use HALWPE, first we configure the simulator to model a target design of the next-generation GPU. We then render a small set of frames of graphics workloads on the target GPU simulator and record the performance of each frame in terms of *cycles-per-frame (CPF)*. We render each frame on the commercially-available host GPU and record profiling features. We then train a suite of statistical models to predict the CPF of graphics workloads on the target, given the features obtained from execution on the host.

As shown in Fig. 1, application performance depends on several factors beyond the architecture itself, including vendor-provided drivers and APIs such as *DirectX* and *OpenGL*. Cross-generation prediction entails not only changes to the architecture but changes to the software stack as well. HALWPE enables collection of performance metrics for model training and inference without the need for software modifications. HALWPE achieves high accuracy (7-9% out-of-sample error) and speed, as it runs ~30,000-45,000x faster than traditional performance simulation in our experiments.

DAC '17, June 18 - 22, 2017, Austin, TX, USA
 ACM 978-1-4503-4927-7/17/06 $15.00
DOI: http://dx.doi.org/10.1145/3061639.3062257

2. MODELING FRAMEWORK

Figs. 2 and 3 illustrate the HALWPE model development flow and model training and prediction. The *Graphics Workload Library (GWL)* refers to our collection of benchmarks, listed in Table 1. The GWL contains rendering frames from 42 *DirectX* games and GPU benchmarking tools spanning the version 9, 10, and 11 APIs. We collected multiple frames per application and treat each as one workload. The GWL applications are assembled into one training set; we apply 10-fold cross validation to estimate out of sample error. We use a proprietary tool (*GfxCapture*), to collect single-frame traces in two formats: *HWTraces* (DirectX API commands) executed on our Haswell host GPU, and *SWTraces* (native GPU commands) executed on our GPU simulator. Two proprietary applications, *GfxPlayer* and *GfxProfiler*, replay isolated traces to collect DirectX program metrics and performance counter measurements, on the Haswell host GPU. To reduce profiling overhead, we only collect performance counters that can be read in one pass. Table 2 summarizes these tools.

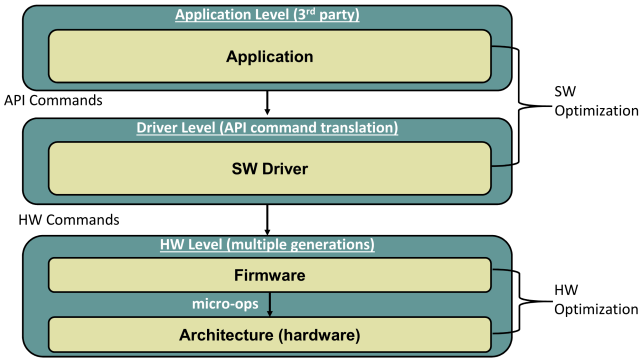

Figure 1. GPU performance depends on the application, driver/API commands, and architecture.

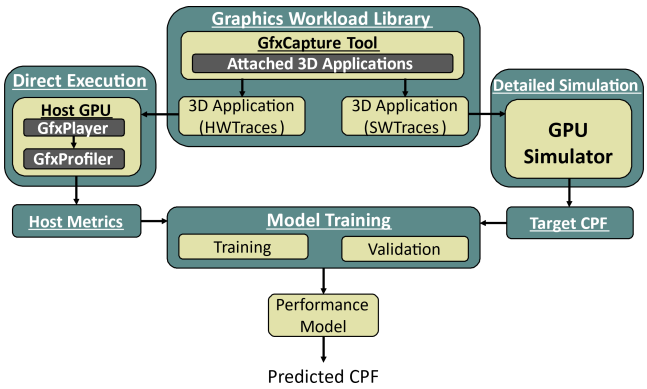

Figure 2. HALWPE Model Development Flow: (1) Traces are collected and stored in the GWL. (2) Workloads execute on the current-generation GPU host, and next-generation simulator. (3) Performance counter measurements and DirectX program metrics obtained from the host are used to train a model to predict the CPF that the GPU simulator would report.

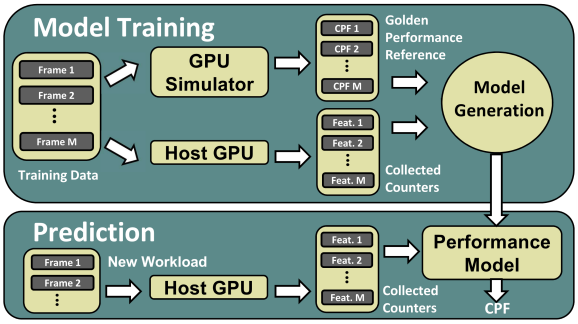

Figure 3. Training (Top): Performance counter measurements and DirectX program metrics obtained from direct execution on the GPU host are used to train a model to predict the performance of a GPU simulator configuration. Prediction (Bottom): An application runs on the GPU host; the collected performance counter measurements and DirectX program metrics are input to the performance model, which predicts the CPF that the GPU simulator would report for that application.

Our models produce a single value, the expected number of GPU execution CPF. Training is performed offline and is not included in the runtime comparison of HALWPE to cycle-accurate simulation, as the cost of training is amortized over repeated model usage.

In practice, the models are trained on one set of workloads, and then deployed to estimate the performance of a disjoint second set of workloads, of arbitrary size. Once a model has been trained, it can be applied to any 3D rendering workload, including standard executables, irrespective of its inclusion in the GWL.

3. INTEL GPU ARCHITECTURES

We experimented with HALPWE using three generations of Intel integrated GPUs (see Table 3). Our host platform is a desktop PC running a 4-core, 8-thread Intel Core i7-4790k, 16GB of DDR3 @ 1666MHz, an Intel HD 4600 Haswell GT2 GPU running at 1250MHz., and a 2TB 7200RPM hard disk. The Broadwell GT2 (Target 1), Broadwell GT3 (Target 2) the Skylake GT3 (Target 3) are later versions of this GPU for which simulators are available. Performance of the final machine may improve, even if CPF increases, e.g., due to higher clock frequencies.

Table 1. The GWL comprises 36 DirectX applications which collectively render 364 frames.

Workload	Frame Count	DX Version
Diablo 3	2	9
Dragon Age: Origins	11	9
Dragon Age 2	10	9
Left For Dead 2	10	9
Portal 2	12	9
Skyrim	9	9
Starcraft 2	1	9
StarWars The Old Republic	10	9
Witcher2	10	9
3D Mark Vantage	30	10
Hawx	5	10
Cut The Rope	3	10
Fishie	2	10
Resident Evil 5	10	10
Winsat Alpha Blend	3	10
Winsat ALU Perf	3	10
Winsat Batch Perf	3	10
Winsat Constant Buffer	3	10
Winsat Geometry Perf1	3	10
Winsat Geometry Perf2	3	10
Winsat Texture Load Perf	3	10
3DMark 11 entry	19	11
3DMark 11 perf	20	11
3DMark Cloud Gate	9	11
3DMark Ice Storm	9	11
Assassins Creed 3	5	11
AVP: Evolution	7	11
Batman Arkham City	8	11
Battlefield 3	5	11
Bioshock Infinite	7	11
Crisis 3	10	11
Dirt2	10	11
F1 2012	15	11
Farcry 3	10	11
Heaven	10	11
Lost Planet 2	26	11
MaxPayne 3	5	11
Metro Last Light	10	11
Saints Row 3	8	11
Saints Row 4	7	11
Sleeping Dogs	6	11
Stone Giant	11	11
Tomb Raider	1	11

Table 2. Summary of software tools and libraries used.

Tool	Acronym	Purpose
Graphics Workload Library	GWL	Collect frame traces
GPU Simulator	-	Collect golden reference CPF
Graphics Capture Tool	GfxCapture	GPU frame trace capture tool
GPU Hardware Trace	HWTrace	Capture traces executed on the host GPU
GPU Simulator Trace	SWTrace	Capture traces executed on the GPU simulator
HWTrace Replay Tool	GfxPlayer	Replay HWTraces on the host GPU
HWTrace Profiling Tool	GfxProfiler	Collect model features from execution on the host GPU

Table 3. Device legend. Further details about each device can be found in Refs. [8-10].

GPU Product	Available As	# Slices	# EUs	Driver Gen.	# Executable Traces
Haswell GT2	Hardware	1	20	1	300
Broadwell GT2	Simulator	1	24	2	282
Broadwell GT3	Simulator	2	48	3	364
Skylake GT3	Simulator	2	48	3	364

The *Slice* count is a measure of available parallelism. Each slice contains a parallel group of sub-slices as well as shared resources including atomics, L3 cache, shared local memory, and specific fixed functional units. The sub-slices contain *Execution Units (EUs)* and their supporting thread dispatch units, samplers, instruction cache, and other peripherals. A Broadwell GPU slice contains 3 parallel sub-slices with 8 EUs each.

To create hardware-assisted model scenarios, we used simulator configurations that execute a driver reflective of the GPU generation: version 1 (Haswell GT2), version 2 (Broadwell GT2 single-slice), and version 3 (Broadwell GT3 dual-slice, which we also use for Skylake GT3). In some situations, compatibility issues between the architecture and driver caused trace execution to fail on the GPU host and simulator. In Table 3, the Haswell GPU host can execute 300 of the available traces, while simulators for the Broadwell GT2, GT3 and Skylake GT3 can execute 282, 364, and 364 traces respectively. For any host-target prediction scenario (simulator-based or HALWPE), the number of traces that we use to build and evaluate the model is the minimum number that both host and target have the capability to execute.

4. REGRESSION MODELS

HALWPE includes 12 linear and one non-linear regression models, which are summarized in Fig. 4. For each prediction scenario reported in this paper, we train all 13 models and select the one that yields the smallest *out-of-sample error* as the most accurate. The choice to use an ensemble of models is driven by the suspicion that correlations between collected features (DirectX program metrics and performance counter measurements) and the amount of non-linearity in the relationship between features and CPF may vary as the number of generations between host and target GPU changes.

We err on the side of caution by producing several models, each judiciously chosen to target one of these behaviors. The standard linear models based on least squares (OLS/NNLS), shown on the left side of Fig. 4, are useful when the relationship between features and CPF is linear, and features are non-correlated; using the AIC and BIC criteria to remove features can simplify the model and help to avoid overfitting. Linear regularization, shown in the middle, selects features during model construction, as opposed to removing features afterward; this helps to mitigate variance and noise on the prediction curve and can improve model accuracy in many cases. Random Forest regression, the non-linear model, shown on the right, captures the presence of non-linear behavior which tends to become increasingly prevalent as the number of GPU generations between host and target increases.

Readers unfamiliar with the statistical concepts described in the following subsections are encouraged to consult Ref. [7] for details; distinct citations are omitted to conserve space.

4.1 Linear Regression Overview

Let M be the number of workloads and $X = [x_1, x_2, \ldots, x_N]$ be the set of features, i.e., the values of the performance counters that we measure for each workload.

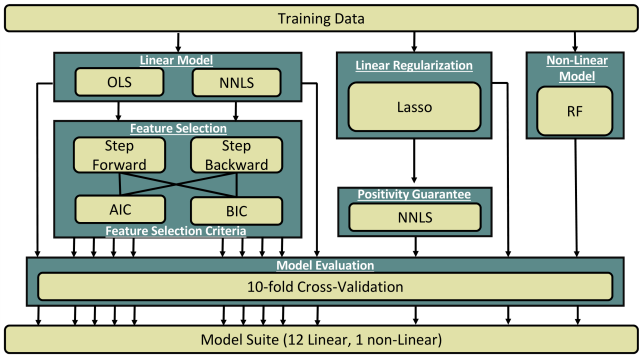

Figure 4. The HALWPE suite generates 13 regression models.

A *model* is a function f that computes a scalar predicted performance value, $\hat{y} = f(X)$. Under a linear model, f has the form:

$$f(X) = \sum_{j=j}^{N} x_j \beta_j + \beta_0, \quad (1)$$

where $\beta = [\beta_1, \beta_2, \ldots, \beta_N]$ is a coefficient vector that corresponds to the features, and β_0 is a bias term called the *intercept*, which serves as a model adjustment factor.

The *error* associated with the i^{th} workload is $y_i - f(X_i)$, where y_i is the empirically obtained CPF, and $f(X_i)$ the predicted CPF. Given training data, the generation of a coefficient vector is formulated as a constrained optimization problem that tries to minimize aggregate error. The model generation techniques employed by HALPWE differ in terms of the optimization problem formulation and how it is refined by post-processing steps (Fig. 4).

4.2 Ordinary Least Squares (OLS)

Given a coefficient vector β, the *aggregate error* of the training data set is the *Residual Sum of Squares (RSS)*:

$$RSS(\beta) = \sum_{i=1}^{M} (y_i - f(X_i))^2. \quad (2)$$

Ordinary Least Squares (OLS) computes the coefficient vector β and intercept β_0 that minimizes $RSS(\beta)$.

4.3 Non-Negative Least Squares (NNLS)

OLS may produce models that estimate negative CPF values for certain data sets, which is physically impossible. *Non-Negative Least Squares (NNLS)* can be applied to ensure that model estimates cannot be negative. NNLS implicitly removes certain features from model by setting negative-valued coefficients to zero and distributing their impact amongst the remaining positive values. NNLS may degrade model accuracy as it no longer minimizes $RSS(\beta)$.

4.4 Feature Selection

OLS and NNLS are *full regression* models that may use all input features. *Feature selection*, which removes feature x_j from the model by setting coefficient β_j to zero, can improve *prediction accuracy* by sacrificing bias to reduce variance, as well as *interpretation*: identifying a subset of features that exhibits the strongest effect on model accuracy enhances understanding of the underlying mechanisms.

Forward Stepwise Selection greedily selects coefficient pairs that achieve the maximal incremental improvement to the model; the process terminates when adding more features is no longer beneficial to model prediction accuracy. *Backward Stepwise Selection* is similar, but starts with a full regression model and iteratively removes one feature at a time.

We apply the *Akaike Information Criterion (AIC)* and the *Bayesian Information Criterion (BIC)* as feature ranking criteria during stepwise selection. This yields four feature selection methods: {Forward, Backward}×{AIC, BIC}, which can be applied to either OLS or NNLS models.

4.5 Linear Regularization Model: Lasso

Lasso is a *linear regularization model* that constructs a model while simultaneously selecting features using an RSS penalty term. Lasso penalizes features in a blanket fashion, unlike step-wise selection, which is iterative. Lasso selects features via shrinkage, which reduces "small enough" coefficients to zero, depending on the value of the regularization term coefficient. We produce two variants of a Lasso, with and without the NNLS criterion.

4.6 Model Evaluation

We use 10-fold cross validation as a precursor to quantify an estimate of the usefulness of a trained model in practice. We report the out-of-sample error (E_{out}), the mean absolute percentage error averaged of all ten folds, as our primary measure for model accuracy; E_{out} reflects the ability of a model to accurately predict a response when applied to previously unseen data. We also evaluate models in terms of their inlier ratios. Given a percentage threshold T, trace X_i is an *inlier* if X_i's *absolute relative percentage error (APE)* is less than T. Given T, the *inlier ratio* (I_R) is the percentage of traces that are inliers. We report 10% and 20% inlier ratios for each model that we produce, and compare inlier ratios across varying thresholds.

4.7 Random Forest Regression

Random Forest (RF) regression is a non-linear supervised learning model in which the prediction is an aggregate of individual predictions made by a set of regression trees. Due to space limitations, we cannot describe RF prediction in detail. We construct our RF using *bootstrap aggregation (bagging)*, applying *feature bagging* to reduce correlation among trees. We compute the out-of-sample error using 10-fold cross-validation, by averaging the *out-of-bag error* for each fold. Each regression tree comprises a random non-overlapping subset of features, while forests tend to include all features as the number of trees grows large.

5. EXPERIMENTAL METHODOLOGY

We present three hardware-assisted predictive models based on dynamic performance counter measurements and application/API profiling from a Haswell GT2 GPU, which provides 577 features.

Scenario$_1$ (282 traces) uses a Haswell GT2 GPU host to predict the CPF of a simulated Broadwell GT2 GPU.

Scenario$_2$ (300 traces) uses a Haswell GT2 GPU host to predict the CPF of a simulated Broadwell GT3 GPU.

Scenario$_3$ (300 traces) uses a Haswell GT2 GPU host to predict the CPF of a simulated Skylake GT3 GPU.

To instrument the host GPU, we attach *GfxProfiler* directly to the *device context*, which is created along with its *device* when the GPU renders a frame. The device creates resources and queries the GPU's rendering capabilities, while the device context comprises the GPU's pipeline and resource states, which generate rendering commands. *GfxProfiler* collects three classes of features: performance counter measurements (via *HWTraces*), profiled DirectX API commands (via *HWTraces*), and *hardware queries* (via the device context) which leverage exposed parts of the API.

Workload execution is performed using an unmodified operating system (OS; Windows 7) and driver. To reduce variability introduced by the OS, we suppress non-OS background processes and run traces in full-screen mode. By leaving the OS and driver unmodified, we eschew control of sleep states. By adjusting BIOS settings, we can disable deep sleep state RC6 and suppress dynamic frequency scaling and Turbo Boost. The sources of variation that remain are competing background tasks, which affect CPU-GPU communication latency and access to shared resources, along with the aforementioned sleep states that we cannot control.

We perform outlier detection and elimination to mitigate variation. We apply the *Median Absolute Deviation (MAD)* test to identify runs that exhibit abnormal behavior. We empirically determined a threshold of ±7 MADs using 10 representative frames, executing each frame 100 times.

During model construction and evaluation, we execute each frame 100 times on the host GPU using *GfxProfiler* to collect features. We remove outliers, i.e., all runs whose CPF values are outside of the ±7 MAD threshold. The CPF and feature values reported for the frame are averaged across the remaining inliers. As an example, Fig. 5 reports the CPF of 100 executions of Witcher 2 Frame 769 normalized to the smallest CPF that we observed. To avoid cold-start issues, we insert a generic "warmup" frame that is executed but not profiled. Most executions are within the MAD window, although some non-negligible variation in CPF is clearly visible.

6. EXPERIMENTAL RESULTS

We generated 13 models for each scenario. For each model, we report the out-of-sample error, 20% and 10% inlier ratios, the number of selected features, and the number of available features; we also report the APE for each workload.

6.1 Hardware-Assisted Models

Tables 4 and 5 respectively report the best-performing non-NNLS and NNLS models that minimized the out-of-sample error for each of three scenarios listed above. For Scenario$_1$ and Scenario$_2$, OLS/Forward/BIC produced the lowest out-of-sample errors: 7.45% and 7.47% respectively. For these two scenarios, models generated using the Haswell host (20 EUs) were able to predict the CPF one generation into the future (Broadwell) with small (24 EUs) and large (48 EUs) increases in parallelism; doubling the number of EUs did not noticeably degrade prediction accuracy.

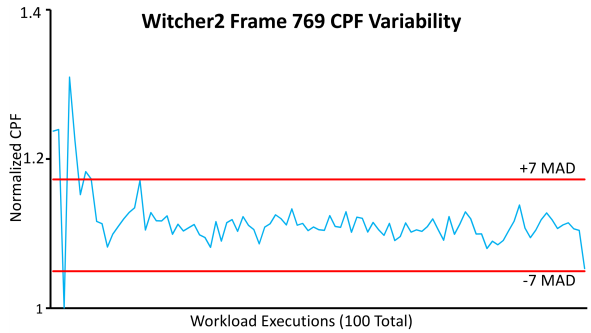

Figure 5. CPF variability for Witcher 2 Frame 769 when executed 100 times; the first execution was an extreme outlier, due to cold start issues, and was removed. Among the remaining 99 runs, 7 frames were correctly identified as outliers and removed using the ±7 MAD threshold.

Table 4. The most accurate non-NNLS models in terms of minimizing out-of-sample error (E_{out}) for Scenarios$_{1-3}$.

Scenario	Best-Performing Model	E_{out}	I_R (20%)	I_R (10%)	Features Selected	Available Features
Scenario$_1$	OLS/Forward/BIC	7.45%	92.76%	85.13%	40	577
Scenario$_2$	OLS/Forward/BIC	7.47%	91.33%	83.67%	30	577
Scenario$_3$	Random Forest (RF)	10.28%	95.85%	62.85%	577	577

Table 5. The most accurate NNLS models in terms of minimizing out-of-sample error (E_{out}) for Scenarios$_{1-3}$.

Scenario	Best-Performing NNLS Model	E_{out}	I_R (20%)	I_R (10%)	Features Selected	Available Features
Scenario$_1$	Lasso/NNLS	13.87%	75.06%	58.95%	23	577
Scenario$_2$	NNLS/Backward/AIC	13.68%	84.00%	73.67%	83	577
Scenario$_3$	NNLS	8.91%	92.63%	77.89%	59	577

Fig 6 depicts the observed CPF, predicted CPF, and APE for the OLS/Forward/BIC models generated for Scenario$_1$ and Scenario$_2$ and the NNLS model generated for Scenario$_3$. Small differences between predicted and observed CPF for the Scenario$_1$ and Scenario$_2$ models can be seen by the naked eye; the differences are more pronounced for Scenario$_3$'s model. The degradation in model quality is readily apparent between scenarios. All three models exhibit the largest APEs at the low-CPF end of the spectrum, although the NNLS model generated for Scenario$_3$ has slightly more high APEs on the higher end. In contrast, the RF model generated for Scenario$_3$ has a more uniform distribution of high APEs across the CPF spectrum. This is similar to distribution of APEs reported for the RF model in Fig. 6 for Scenario$_3$.

6.2 HALWPE Speedup

Fig. 7 compares the execution time of HALWPE to that of the simulator configured as a Broadwell GT2, Broadwell GT3, and Skylake GT3 GPU on the 282 workloads that all three simulator configurations can execute (Table 3). On average, HALWPE achieved a speedup of 29,481x over the Broadwell GT2 simulator, 43,643x over the Broadwell GT3 simulator, and 44,214x over the Skylake GT3 simulator. Compared to cycle-accurate simulators, predictive models sacrifice accuracy to provide a rapid result. The execution time of a model on a frame entails running the frame trace on the host GPU and then generating model using the obtained features; in most cases, the latter is negligible.

Fig. 8 reports the time to train all 13 HALWPE models --excluding target simulation time and host GPU execution time to render each frame once. In addition to rendering, host GPU execution time includes overhead associated with loading the application, profiling, and streaming API commands from the trace player. The longest model training and host GPU execution time was ~2.5 hours for Scenario$_3$. We rendered each frame 100 times, thus execution time is dominated by the host GPU, not model training.

7. RELATED WORK

GPU simulators [3, 4, 18] run orders of magnitude slower than native execution [6, 19]. Representative sampling and multi-threading [12] remain prohibitively slow. Predictive models sacrifice accuracy but run much faster.

7.1 Predictive Models for CPUs

HALWPE is inspired by predictive models for CPUs based on performance counter readings and program metrics [11, 13]; however, the selected features are quite different. Using simulator internals as features can yield higher accuracy than hardware execution [16], but run at simulator speeds. Cross-architecture/ISA performance prediction using performance counters has recently been demonstrated as well [21, 22].

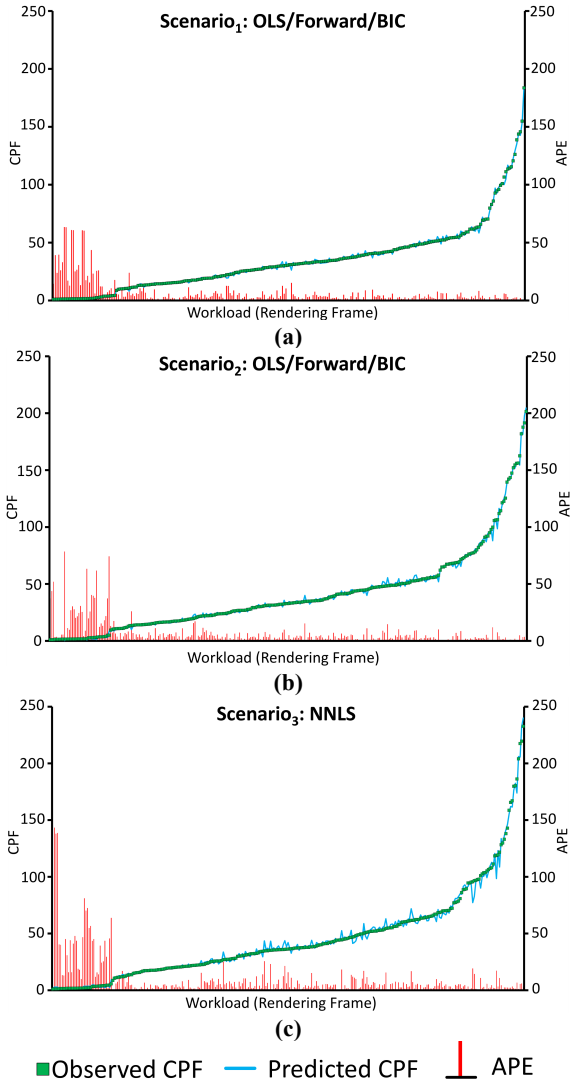

Figure 6. The observed CPF, predicted CPF, and per-workload APE for OLS/Forward/BIC models generated for Scenario$_1$ (a) and Scenario$_2$ (b) and the NNLS model generated for Scenario$_3$ (c). Workloads are ordered from left-to-right in non-decreasing order of observed CPF.

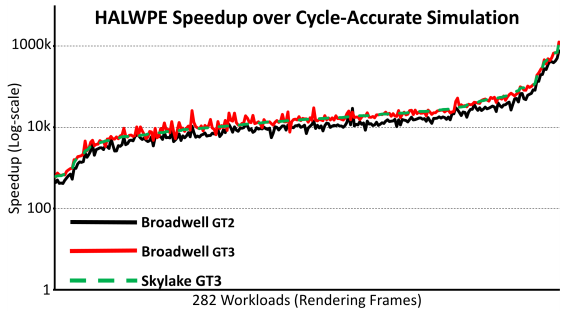

Figure 7. HALWPE speedup over simulators for Broadwell GT2/GT3 and Skylake GT3 GPUs using the 280 workloads common to all three simulators (Table 3). Frames are ordered by increasing Skylake GT3 speedup. Average speedups were 29,481x for Broadwell GT2, 43,643x for Broadwell GT3, and 44,214x for Skylake GT3.

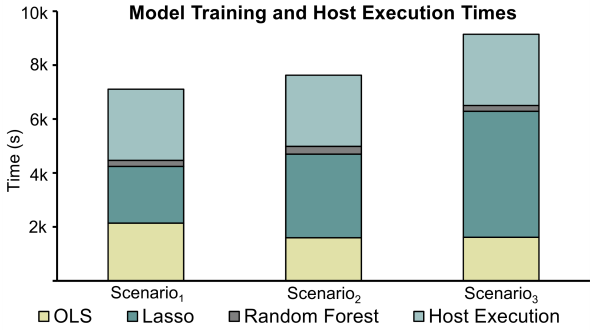

Figure 8. Model training and host GPU execution time for the three scenarios.

7.2 Predictive Models for GPUs

GPU models have been able to accurately predict performance and power consumption for the devices on which they were trained [2, 5, 15, 17]; however, cross-generation prediction has not yet been achieved. One recent model can predict application performance and power consumption across a variety of GPU configurations with three degrees of freedom (frequency, parallelism, and memory bandwidth) using performance counter measurements [19]. The model was limited to a current-generation GPU, and required extensive firmware modifications, whereas HALWPE, does not modify the firmware and can use production drivers.

Prior work on single-generation predictive performance and power models for ATI GPUs based on random forest regression [20] shares many principle similarities with our work. Of particular interest is a technique to rank model features in order of their impact on predicted performance and/or power, which we intend to integrate into HALWPE in the future.

XAPP [1] uses CPU performance counter measurements to predict which portions of a program will reap the greatest benefit from GPU acceleration; in contrast, HALWPE focuses on early-stage GPU architectural design space exploration and early performance feedback for graphics software development.

8. CONCLUSION AND FUTURE WORK

HALWPE has established the feasibility of cross-generation GPU CPF prediction using performance counter readings, DirectX metrics, and hardware queries. HALWPE achieved an out-of-sample error rate of 8.91% when predicting across two GPU architecture generations, which include extra parallelism, microarchitecture changes, and driver upgrades. Compared to cycle-accurate simulation, HALWPE achieved a speedup of 44,214x compared to a cycle-accurate simulator for this specific prediction scenario. Predictive modeling can aid early-stage microarchitecture design space exploration, and may be able to help with identification of performance bottlenecks; however, it must be applied with care.

Several open questions remain: (1) It is unclear if HALWPE can predict energy consumption with equal effectiveness. (2) It would be nice to be able to reuse a model trained for one target to predict the CPF of a similar target; criteria for model reuse and retraining would be beneficial. (3) HALWPE was applied to GPUs using frame rendering workloads from games; it is uncertain if the approach generalizes to GPGPUs with greater workload diversity.

ACKNOWLEDGMENT

This work was support in part by NSF Award #1528181.

REFERENCES

[1] N. Ardalani, et al., "Cross-architecture performance prediction (XAPP) using CPU to predict GPU performance" in Proc. Int. Symp. Microarchitecture (MICRO-48), 2015, pp 725-737.

[2] P. E. Bailey, et al., "Adaptive configuration selection for power-constrained heterogeneous systems," in Proc. Int. Conf. on Par. Proc. (ICPP), 2014, pp. 371-380.

[3] A. Bakhoda, et al., "Analyzing CUDA workloads using a detailed GPU simulator," in Proc. Int. Symp. Perf. Analysis of Systems and Software (ISPASS), 2009, pp. 163-174.

[4] V.M. del Barrio, et al., "ATILLA: a cycle-accurate execution drive simulator for modern GPU architectures" in Proc. Int. Symp. Perf. Analysis of Systems and Software, (ISPASS), 2006, pp. 231-241.

[5] J. Chen, et al., "Tree structured analysis on GPU Power study" in Proc. Int. Conf. Computer Design (ICCD), 2011, pp. 57-64.

[6] A. Gutierrez, et al., "Sources of error in full-system simulation," in Proc. of the Int. Symp. Perf. Analysis of Systems and Software (ISPASS), 2014, pp. 13-22.

[7] T. Hastie, R. Tibshirani, and J. Friedman. The Elements of Statistical Learning. Springer, New York, 2001.

[8] Intel Corporation, "The Compute Architecture of Intel Processor Graphics Gen7.5." [Online]. Available: https://goo.gl/5HZ54v

[9] Intel Corporation, "The Compute Architecture of Intel Processor Graphics Gen8." [Online]. Available: https://goo.gl/TnpAGc

[10] Intel Corporation, "The Compute Architecture of Intel Processor Graphics Gen9." [Online]. Available: https://goo.gl/RMmUc6

[11] E. Ipek, et al., "Efficiently Exploring architectural design spaces via predictive modeling," in Proc. Int. Conf. Arch. Support for Prog. Languages and Operating Systems (ASPLOS), 2006, pp. 195-206.

[12] W. Jia, K. Shaw, and M. Martonosi, "Starchart: hardware and software optimization using recursive partitioning regression trees," in Proc. Int. Conf. Parallel Architectures and Compilation Techniques (PACT), 2013, pp. 257-268.

[13] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in Proc. Int. Conf. Arch. Support for Prog. Languages and Operating Systems (ASPLOS), 2006, pp. 185-194.

[14] S. Lee and W. W. Ro, "Parallel GPU architecture simulation framework exploiting work allocation unit parallelism," in Proc. Int. Symp. Perf. Anal. Systems and Software (ISPASS), 2013, 107-117.

[15] X. Ma, et al., "Statistical power consumption analysis and modeling for GPU-based computing," in Proc. ACM SOSP Workshop on Power Aware Computing and Systems (HotPower), 2009.

[16] B. Ozisikyilmaz, G. Memik, and A. Choudhary, "Machine learning models to predict performance of computer system design alternatives" in Proc. Int. Conf. Par. Proc. (ICPP), 2008, pp. 495-502.

[17] S. Song, et al., "A simplified and accurate model of power-performance efficiency on emergent GPU architectures", in Proc. Int. Symp. Parallel & Distributed Proc. (IPDPS), 2013. pp. 673-686.

[18] R. Ubal, et al., "Multi2Sim: a simulation framework for CPU-GPU computing," in Proc. Int. Conf. Parallel Architectures and Compilation Techniques (PACT), 2012, pp. 335-344.

[19] G. Wu, et al. "GPGPU performance and power estimation using machine learning" in Proc. Int. Symp. High Perf. Comp. Arch. (HPCA), 2015, pp. 564-576.

[20] Y. Zhang, Y. Hu, B. Li, and L. Peng, "Performance and power analysis of ATI GPU: a statistical approach," in Proc. Int. Conf. Networking, Architecture and Storage (NAS), 2011, pp. 149-158.

[21] X. Zheng, L.K. John, and A. Gerstlauer, "Accurate phase-level cross-platform power and performance estimation" in Proc. Design Automation Conf. (DAC), 2016, article no. 4.

[22] X. Zheng, et al., "Learning-based analytical cross-platform performance prediction" in Proc. Int. Conf. Embedded Computer Sys., Arch., Modeling and Simulation (SAMOS), 2015, pp. 52-59.