QIG: Quantifying the Importance and Interaction of GPGPU Architecture Parameters

Zhibin Yu, *Member, IEEE*, Jing Wang, Lieven Eeckhout, *Senior Member, IEEE*, and Chengzhong Xu, *Fellow, IEEE*

***Abstract*—Graphic processing units (GPUs) are widely used for general-purpose computing—so-called GPGPU computing. GPUs feature a large number of architecture parameters, resulting in a huge design space. To quickly explore this design space and identify the optimum architecture for a group of widely used computing kernels, it is critical to know how important each parameter is and how strongly these parameters interact with each other. This paper proposes an ensemble-learning-based approach, called quantifying the importance and interaction of Gpgpu architecture parameters (QIG), to quantify the importance of architecture parameters and their interactions with respect to performance. QIG employs a stochastic gradient boosted regression tree to construct performance models using performance data from a random set of GPU architectures. Leveraging these models, QIG observes the impact of each architecture parameter on performance, and calculates its importance and interaction intensity with other parameters. Using 25 widely used GPGPU kernels, we demonstrate that QIG accurately ranks the importance and interaction of GPU architecture parameters while the previously proposed Plackett–Burman design does not. Moreover, we show that QIG leads to a substantially more accurate performance model compared to prior work, including Starchart and approaches using artificial neural networks and supported vector machines: average error of 4.2% for QIG versus 23+% for prior work. Finally, QIG reveals a number of interesting insights for GPU architectures running GPGPU workloads.**

***Index Terms*—Architecture, design space exploration, modeling, performance evaluation.**

I. Introduction

GRAPHIC processing units (GPUs) deliver massive computational power by employing many cores to run hundreds of thousands of concurrent threads. Programming models, such as CUDA [1], ATI Stream Technology [2], and OpenCL [3] enable leveraging powerful graphics hardware to perform general-purpose computing, so-called GPGPU computing. As a sharply increasing number of emerging workloads, such as deep learning and big data analytics start to run on GPUs, quickly devising an optimized GPU architecture for a set of widely used GPGPU kernels is desirable.

A GPU architecture has up to several hundreds of design parameters, resulting in a huge design space. To further complicate matters, these parameters interact with each another in various complex ways. As a result, quickly designing an optimized GPU architecture for a given set of kernels is extremely challenging. Fortunately, the architecture parameters of GPUs are generally not equally important to performance, and neither are the interaction intensities between them. This offers an opportunity to accelerate the GPU architecture design process by only focusing on the important architecture parameters and their dominant interactions. In other words, knowing the key architecture parameters and interactions avoids wasting design effort and time on suboptimal parts of the design space.

Quantifying the importance and interaction intensity of GPU architecture parameters is quite challenging. Architectural simulation, while trivial to use, is not a viable solution, simply because it is too slow. Doing a number of parameter sweeps to understand parameter importance and interactions easily results in a huge number of simulations with each simulation taking a long time to complete. This easily results in unbearably long simulation times. Analytical and predictive modeling can significantly speed up this process, replacing slow simulation with fast prediction. Recent work in analytical modeling [4]–[6], as well as statistical reasoning [7]–[9] and machine learning [10] has made significant progress toward predicting GPU performance. Unfortunately, they are not accurate enough and do not readily identify the important parameters and interactions to accelerate design space exploration.

In this paper, we propose a novel approach, called quantifying the importance and interaction of Gpgpu architecture parameters (QIG), using stochastic gradient boosted regression trees (SGBRTs) [11] to quantify the importance and interaction intensity of GPU architecture parameters with respect to performance. SGBRT builds an *ensemble* model, i.e., it builds multiple empirical models which it then combines to form an overall model. SGBRT involves a training phase in which performance numbers need to be collected for

Manuscript received October 25, 2016; revised January 23, 2017; accepted April 5, 2017. Date of publication April 25, 2017; date of current version May 18, 2018. This work was supported in part by the National Key Research and Development Program under Grant 2016YFB1000204, in part by the Major Scientific and Technological Project of Guangdong Province under Grant 2014B010115003, in part by the Shenzhen Technology Research Project under Grant JSGG20160510154636747, in part by the Shenzhen Peacock Project under Grant KQCX20140521115045448, and in part by the Outstanding Technical Talent Program of CAS, and National Natural Science Foundation of China under Grant 61672511. This paper was recommended by Associate Editor S. Pasricha. *(Corresponding author: Jing Wang.)*

Z. Yu and C. Xu are with the Cloud Computing Center, Shenzhen Institutes of Advanced Technology, Chinese Academy of Science, Shenzhen 518055, China (e-mail: zb.yu@siat.ac.cn; cz.xu@siat.ac.cn).

J. Wang is with Capital Normal University, Beijing 100048, China (e-mail: jwang@cnu.edu.cn).

L. Eeckhout is with Ghent University, 9052 Ghent, Belgium (e-mail: lieven.eeckhout@ugent.be).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2698026

a range of GPU architecture configurations running GPGPU kernels of interest; SGBRT then builds an empirical model using the training data. QIG builds on top of SGBRT and computes the importance for each architecture parameter and all of their pairwise interactions.

QIG has several advantages over existing statistical reasoning and machine learning-based approaches. First, QIG does not build a single (complex) performance model as prior work in GPU performance modeling does. Instead, it combines many simple models to form an ensemble model. This leads to a more accurate overall model while requiring far fewer training examples. Second, QIG quantifies the importance of GPU architecture parameters and their interactions, which none of the prior work in GPU performance modeling provides and which is of critical importance for architects to efficiently explore the design space. Third, QIG not only accurately predicts performance but also reveals interesting insights based on the importance of and the interactions among architecture parameters, which helps architects understand and optimize GPU performance.

We apply QIG to a set of 25 GPGPU benchmarks and find that different benchmarks are sensitive to different GPU architecture parameters. The (by far) most dominant GPU architecture parameters are core frequency and the maximum number of thread blocks per core. Several benchmarks are sensitive to other architecture parameters, such as L1 data cache size and interconnect frequency. Moreover, we find that 8 out of 25 benchmarks are predominantly sensitive to a single pairwise parameter interaction; the other 18 benchmarks are sensitive to two to five pairwise parameter interactions. This reinforces the observation that the GPU architecture space is complex and requires efficient techniques to identify important parameters and interactions to efficiently cull the large design space toward the optimal design.

Prior work in CPU design space exploration proposed the Plackett–Burman (PB) design of experiment to identify important architectural parameters and their interactions [12]. PB is limited though to quantifying *select* pairwise interactions, i.e., it does not systematically explore *all* possible pairwise interactions, in contrast to QIG. Moreover, we find PB to yield misleading results in some cases, as we show in this paper.

In summary, this paper makes the following contributions.

1) We use ensemble learning, in particular SGBRT, to construct accurate GPU performance models.
2) We propose QIG to quantify the importance of GPU architecture parameters and their pairwise interactions.
3) We employ 25 GPGPU kernels to evaluate QIG and compare its accuracy against prior work. QIG is shown to be substantially more accurate (average error of 4.2%) compared to Starchart [8] and artificial neural network (ANN)/supported vector machine (SVM)-based models with an average error around 23+% for the same of training data. Moreover, QIG accurately identifies the important GPU architecture parameters and pairwise interactions, while the PB design approach does not.
4) Using QIG, we reveal three interesting insights: a) GPGPU performance is dominated by only a handful architectural parameters; b) although a number of benchmarks are sensitive to a single pairwise parameter interaction, many more benchmarks are sensitive to several pairwise parameter interactions; and c) the strongest interactions may not necessarily take place between the (two) most important architecture parameters.

The rest of this paper is organized as follows. Section II presents background in ensemble learning, and provides a description of SGBRT. Section III describes how we leverage SGBRT to build performance models and quantify the importance of GPU architecture parameters and their interactions. Section IV depicts our experimental methodology. Section V provides results and analysis. Section VII describes related work, and Section VIII concludes this paper.

II. Background

We first describe ensemble modeling, and then describe its two essential components (regression trees and boosting).

A. Ensemble Model

Statistical reasoning approaches, such as linear regression assume a parametric model and infer the model parameters using a set of training data (so-called parametric models). In contrast, machine learning techniques, such as ANNs do not employ a parametric model but instead infer a nonparametric data model to relate the independent and dependent variables (so-called nonparametric models). Both statistical reasoning and machine learning build a *single* model from the training data set. These techniques generally yield very accurate models but may require a lot of training data to do so.

Ensemble models have been proposed to increase the prediction accuracy across a wide range of data sets by combining many single models. The key intuition is that it is typically easier to build accurate models by combining many simple models than to build a single sophisticated highly accurate model. However, how to combine models significantly affects the accuracy of the final model as well as the efficiency of the model building process. A number of combination techniques have been proposed, including bagging [13], stacking [14], and model averaging [15]. Because there are no invalid GPU architecture parameter values (noise) in this paper and because boosting [16] generally produces more accurate results than other combination techniques in the absence of noise [17], we employ boosting in this paper.

B. Regression Tree

The ensemble model used in QIG uses regression trees for constructing the "simple" models. A regression tree partitions the parameter space (e.g., the GPU architecture design space) into rectangles, as shown in Fig. 1. In the first step, the whole design space is divided into two parts according to split point $sp1$ for architecture parameter $pv1$. (Note that a design space can be divided into more than two parts if two or more split points are used per architecture parameter.) The left part is further split into two subrectangles based on split point $sp2$ for architecture parameter $pv2$; the right part is split according to split point $sp4$ for architecture parameter $pv4$; etc. This procedure is performed recursively until a stop criterion is met.

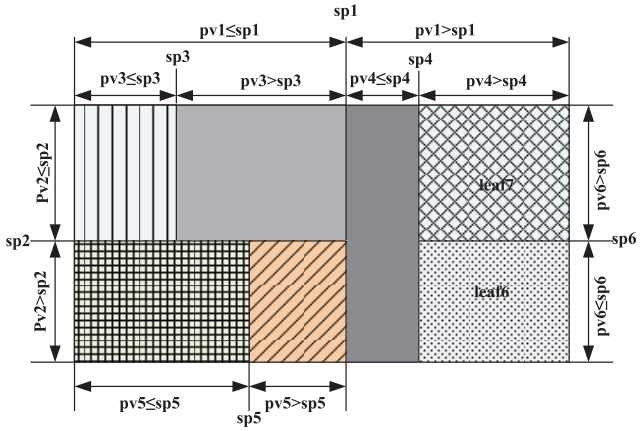

Fig. 1. Illustration of how regression trees work: "pv" represents a GPU architecture parameter; "sp" refers to a split point.

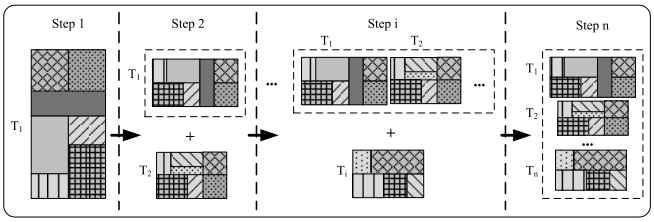

Fig. 2. Illustration of how boosting combines multiple regression trees into a single overarching performance model.

The smallest rectangles are called the leaves of the tree. For example, the two rectangles represented by *leaf*6 and *leaf*7 are two leaves generated by split point *sp*6 for parameter *pv*6.

The split points are chosen such that the data values in each group are most similar. For example, the data values in rectangle *leaf*6 are more similar to each other than to the data values in *leaf*7; in other words, architecture parameter *pv*6 and its split point *sp*6 are chosen, such as to maximize the similarity within each subrectangle. Regression trees use the mean performance within a leaf rectangle to predict performance for all designs in that rectangle, e.g., mean performance across all training examples in *leaf*6 is used as the performance prediction for all designs that end up in *leaf*6 during model evaluation. The regression tree is built such that the prediction error is minimized. In other words, at each step during the model building process are the architecture parameter and its corresponding split point selected, such as to maximize similarity within each subrectangle, thereby maximizing accuracy.

C. Boosting

Boosting combines multiple regression trees into an ensemble model, see Fig. 2. In the first step, a regression tree is grown under a given tree complexity or tree size (the number of nodes in the tree) to minimize its prediction error, e.g., instructions per second (IPS) prediction error in the context of the GPU design space. In the second step, a different regression tree is grown to reflect the variation in GPGPU performance that is not reflected by the first tree. Subsequently, an initial ensemble model is created by combining the first two regression trees: $\alpha_1 T_1 + \alpha_2 T_2$, with T_1 and T_2 the performance predictions by the two regression trees, respectively, and α_1 and α_2 the respective coefficients. This procedure is performed recursively, i.e., more regression trees are added to the ensemble model, until a specific criterion, such as a target accuracy of 90% is met.

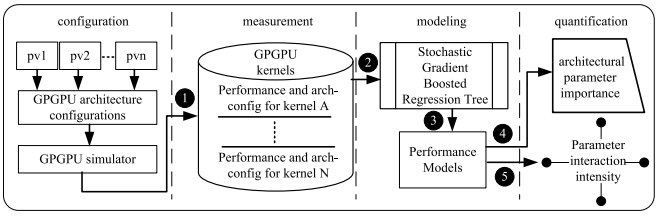

Fig. 3. QIG workflow.

As depicted, boosting is a sequential process in which the original model remains unchanged at each step. Moreover, the performance variation needed to explain gradually reduces as the ensemble model construction proceeds and the model becomes more accurate. Randomness is typically introduced into a boosted model to improve accuracy and speed, and to mitigate over-fitting [11]. Therefore, the ensemble learning technique used in this paper also includes a stochastic component, which leads to the overall model being SGBRT.

The overall model produced by SGBRT can be thought of as a linear combination of regression trees, as follows:

$$\text{IPS} = \alpha_1 T_1 \cdots + \alpha_i T_i + \cdots + \alpha_n T_n \tag{1}$$

with n the number of regression trees in the model, T_i the performance predicted by the ith regression tree model, and α_i the contribution of T_i to the model. Two parameters are crucial for SGBRT construction. The first one is the *learning rate* α_i, which we assume to be constant in this paper. The second is *tree complexity* or tree size which controls the maximum level of interaction between GPU architecture parameters that can be considered. We will elaborate on the determination of these two SGBRT parameters in the next section.

III. QIG

We now describe how we leverage SGBRT in QIG to quantify the importance of GPU architecture parameters and their interactions.

A. QIG Workflow

QIG is designed to quantify the importance of GPU architecture parameters and their interactions with respect to performance; we use IPS to represent performance in this paper. Fig. 3 shows a block diagram of the QIG workflow which consists of the following four modules: 1) *configuration*; 2) *measurement*; 3) *modeling*; and 4) *quantification*. The configuration module generates a set of GPU architectures by randomly choosing a value for each architecture parameter in its value range. Randomly choosing parameter values guarantees a uniform sampling distribution across the design space for constructing the training set. For each randomly selected configuration, we simulate a set of GPGPU kernels, see step ① in Fig. 3. We repeat step ① a number of times for each

Algorithm 1 SGBRT Training Algorithm

Input: the M measurements (matrix S)
Output: IPS
1: Initialize $\hat{f}_0(x_i) = \overline{IPS}$, with $\overline{IPS}$ the average of $\{IPS_i\}$
2: **for** $m = 1$ to M **do**
3: (a) compute the current residuals
4: $r_{im} = IPS_i - \hat{f}_0(x_i), i = 1, ..., p$
5: (b) partition the predictor space into H disjoint regions
6: $\{R_{hm}\}_{h=1}^{H}$ based on $\{r_{im}, x_i\}_{i=1}^{p}$
7: (c) compute the constant fit for each region
8: $\gamma_{hm} = argmin_\gamma \sum_{x \in R_{hm}} (r_{im} - \gamma)^2$
9: (Note: we randomly select some predictor variables from each region to compute the constant fit. This is controlled by the bag fraction.)
10: (d) update the fitted model
11: $\hat{f}_m(x) = \hat{f}_{m-1}(x) + \nu \times \sum_h \gamma_{hm} I(x \in R_{hm})$
12: Exit
13: **end for**

kernel, depending on the model accuracy we want to achieve in the later steps.

The measurement module collects performance (IPS) and the respective architecture parameter values during step ①. The results are stored in a vector

$$v_i = \{\text{IPS}_i, pv_{1i}, \ldots, pv_{ji}, \ldots, pv_{ni}\}, i = 1, \ldots, M \quad (2)$$

with v_i the vector generated from the measurements of the ith GPU architecture configuration; IPS_i is the performance of the ith configuration; and pv_{ji} is the jth GPU architecture parameter value of the ith measurement; n is the number of architecture parameters, and M is the total number of training measurements performed. Note that M is the product of the number of kernels times the number of GPU architectures.

The modeling module constructs performance models per kernel using SGBRT. To build the models, we need to construct a training set S which is a matrix, with each row being a vector v_i as defined by (2). As step ② in Fig. 3 shows, S is the input data set for SGBRT to build a performance model for a specific kernel (shown in step ③). Algorithm 1 formally describes SGBRT model construction. Note that at line 11, $I(\cdot)$ is an indicator function that returns 1 if its argument is true; if not, it returns 0. The ν represents the learning rate of SGBRT which is between 0 and 1. The learning rate is used to weight the contribution of each tree as it is added to the model, as previously described. Decreasing learning rate increases the number of trees required in the ensemble model. In general, a smaller learning rate results in higher accuracy because more trees contribute to the final result although longer computation time is required.

The resulting performance model can be represented as

$$\text{perf} = f(pv_1, pv_2, \ldots, pv_i, \ldots, pv_n) \quad (3)$$

with pv_i the value of the ith architecture parameter, and n the total number of architecture parameters. We can construct a model for each GPGPU kernel or for a group of kernels, depending on the requirements. (In this paper, we consider per-kernel models.) Based on these models, the quantification module then quantifies the importance of the GPU architecture parameters and their interactions, as shown by steps ④ and ⑤, respectively, and which we describe in more detail in the following sections.

Validating the model's accuracy is done by considering a GPU architecture configuration, and predicting performance using the per-kernel performance model using (3). Simulating the kernel for that same GPU architecture configuration then yields a point of comparison to validate the model's accuracy. When validating the model, we randomly generate a number of GPU architecture configurations which we then simulate, and predict performance for using the per-kernel models. Comparing the model predictions against the simulation results provides the required validation. Note we make sure the evaluation set of GPU architecture configurations for validating the model is disjoint from the training set of GPU architecture configurations to construct the models.

Example: We now illustrate the QIG workflow using an example for the breadth-first-search (BFS) benchmark and the GPGPU-sim simulator. The baseline GPU architecture is modeled after the NVidia GTX480 (see Table I); the architecture parameters that we explore are shown in Table III. (see Section IV for details about our experimental setup.) We first randomly choose a value for each parameter within its value range. For example, we choose 2 for `ccta`, 0.6 GHz for `cfrq`, 0.7 GHz for `infrq`, etc. We then use these values to configure the GPGPU-sim simulator, after which we simulate BFS. When the simulation completes, we collect the IPC value and in turn calculate IPS. By doing so, we obtain one vector as specified by (2). We repeat this procedure N times and we hereby get N vectors. We use these N vectors as a training set to build a performance model for BFS as a function of the architecture parameters listed in Table III.

B. Quantifying Parameter Importance

Once the SGBRT-based performance model is constructed, as just described, we can leverage the model to quantify the importance of GPU architecture parameters and their interactions. As aforementioned, we build an ensemble model which is a combination of multiple simple regression trees. For a single tree T, one can use $I_j^2(T)$ as a measure of importance for each architecture parameter pv_j, which is based on the number of times pv_j is selected for splitting a tree weighted by the squared improvement to the model as a result of each of those splits [18]. This measure of importance is calculated as follows:

$$I_j^2(T) = \text{nt} \cdot \sum_{i=1}^{\text{nt}} P^2(k) \quad (4)$$

with nt the number of times pv_j is used to split tree T, and $P^2(k)$ the squared performance improvement to the tree model by the kth split. In particular, $P(k)$ is defined as the relative IPS error $(\text{IPS}_k - \text{IPS}_{k-1})/\text{IPS}_{k-1}$ after the kth split. If pv_j is used as a splitter in R trees in the ensemble model, the importance of pv_j to the model equals

$$I_j^2 = \frac{1}{R} \sum_{m=1}^{R} I_j^2(T_m). \quad (5)$$

To ease understanding, the importance of a GPU architecture parameter is normalized so that the sum across all parameters adds up to 100. A higher percentage indicates

stronger influence of the corresponding architecture parameter on performance.

C. Quantifying Parameter Interactions

While *tree complexity* or tree size determines the maximum level of interaction between architectural parameters that can be studied in the ensemble model, no metric is readily available to quantify how strongly a pair of architectural parameters interacts with each other. To address this issue, we construct a linear regression model per pair of architecture parameters and consider the residual variance of the model as an indication for interaction intensity. The intuition is that if two architecture parameters are orthogonal (i.e., they do not interact), the residual variance will be small because the linear model will be able to accurately predict the combined effect of both parameters. If on the other hand, the architecture parameters interact substantially, this will be reflected in the residual variance being significantly larger than zero, because the linear model is unable to accurately capture the combined effect of the parameter pair. The linear regression model is trained for each pair of architectural parameters while setting the values of all other parameters to their respective means. This process is repeated for each possible parameter pair. The residual variance or interaction intensity for a particular parameter pair is computed as

$$v = \sum_{i=1}^{n} (p_i - \overline{p})^2 \tag{6}$$

with p_i the performance predicted by the linear regression model, $\overline{p}$ the observed performance, and n the number of predictions. Zero indicates that there is no interaction between two architecture parameters, and a higher value indicates a stronger interaction.

Because interaction intensity as just defined does not directly show its importance among all possible pairs, we therefore normalize against the other pairs. This is done as follows:

$$I_i = \left(\frac{v_i}{\sum_{j=1}^{n} v_j} \right) \times 100\% \tag{7}$$

with I_i the importance of the ith parameter-pair interaction and v_i the ith parameter-pair interaction intensity. As such, we can tell how much more/less important a parameter pair is compared to another parameter pair, which reflects the relative interaction intensity of parameter pairs.

IV. Experimental Setup

In this section, we describe the simulator, benchmarks, GPU architecture design space, real hardware platforms, and tools used to evaluate QIG.

A. Simulator and Benchmarks

We employ a cycle-level GPGPU simulator, GPGPU-sim v3.2 [19] to validate the efficacy of QIG. The baseline GPU architecture is modeled after NVIDIA's GTX480; the key architecture parameters are listed in Table I. Note that we have to resort to simulation as it is impossible to explore the GPU design space on real hardware. The experimented GPGPU benchmarks are taken from the three most popular CUDA benchmark suites, CUDA SDK [20], Rodinia [21], and Parboil [22], along with a number of benchmarks taken from recent papers, see Table II. We select these benchmarks because they cover a wide range of application domains: encryption (AES), finance (BS), scientific computing (FWT, MM, MT, LPS, LIB, SP), graph processing (BFS, NE), artificial intelligence [back propagation (BP)]. We observe significant diversity in the performance characteristics at the GPU architecture level across this diverse set of benchmarks.

TABLE I
Baseline GPU Architecture Configuration Initialized in GPGPU-sim. SM—Streaming Multiprocessor

Configurations	NVIDIA Fermi GTX480
Num. of SMs	15
max_warp_core	48
core_cta	8
No. Schedulers per SM	2
core_registers	32768
shared_memory	48KB
L1_DCache	16KB per SM (32-sets/4-ways), LRU
L1_ICache	2KB per SM (4-sets/4-ways), LRU
L2_DCache	768KB (32-sets/16-ways/12-partitions), LRU
Min. L2 Access Latency	120 cycles
Min. DRAM Access Latency	220 cycles
Warp size	32
Warp Scheduler	Round Robin
Interconnect Topology	Mesh
Routing Mechanism	Dimension Order
Routing delay	1
Virtual Channels	2

TABLE II
Experimented Benchmarks

Benchmark	Abbr.	Source
Histogram 64	64H	CUDA SDK
Advanced Encryption Standard	AES	CUDA SDK
Bread-First Search on a graph	BFS	CUDA SDK
Binomial Options	BN	CUDA SDK
Black-Scholes	BS	CUDA SDK
Fast Walsh Transform	FWT	CUDA SDK
Matrix Multiplication	MM	CUDA SDK
Matrix Transpose	MT	CUDA SDK
Pairwise Sequence Alignment	MUM	CUDA SDK
N-Queens Solver	NQU	CUDA SDK
Scan(Parallel prefix sum)	SLA	CUDA SDK
Scalar Product	SP	CUDA SDK
Similarity Score	SS	CUDA SDK
Back Propagation	BP	Rodinia
HotSpot	HS	Rodinia
Hybrid Sort	HY	Rodinia
K-Means	KM	Rodinia
Particle Filter	PF	Rodinia
Coulombic Potential	CP	Parboil
Sum of absolute differences	SAD	Parboil
3D Stencil Operation	STO	Parboil
Cellular Automation	CL	[23]
LIBOR Monte Carlo	LIB	[24]
3D Laplace Solver	LPS	[25]
Nearest Neighbor	NE	[26]

B. GPU Architecture Parameters

We consider a GPU design space in which we vary 15 architecture parameters, each with 5 or 6 different values, see

TABLE III
GPU ARCHITECTURE DESIGN SPACE. NOTE THAT THERE ARE 12 L2_DCACHE PARTITIONS OUR BASELINE GPU ARCHITECTURE

Parameters	Abbrv.	Values	#
core_cta	ccta	1,2,4,8,16	5
core_frequency	cfrq	0.5 - 1.0 GHz : 0.1+	6
intconn_frequency	infrq	0.5 - 1.0 GHz : 0.1+	6
L2_frequency	L2frq	0.5 - 1.0 GHz : 0.1+	6
DRAM_frequency	dmfrq	0.5 - 1.0 GHz : 0.1+	6
L1_DCache	L1D	2, 4, 8, 16, 32 KB	5
L1_ICache	L1I	1, 2, 4, 8, 16 KB	5
L1_tex_cache	L1tex	3, 6, 12, 18, 24 KB	5
L1_con_cache	L1con	2, 4, 6, 8, 10 KB	5
L2_DCache/partition	L2	16, 32, 48, 64, 80 KB	5
core_registers	cregs	4, 8, 16, 32, 64 K	5
max_warp_core	mwc	32, 36, 40, 44, 48	5
shared_memory	shm	2, 4, 8, 16, 32 KB	5
DRAM_queue	dmq	8, 16, 24, 32, 40, 48	6
DRAM_r_queue	dmrq	64, 72, 80, 88, 96, 104	6

Table III. We keep the values for the other GPU architecture parameters the same as the NVIDIA GTX480 configuration. The design space explored in this paper includes 91 billion design points in total. `core_cta` specifies the maximum number of thread blocks that a core can support. The next four parameters specify clock frequency of the shader cores, interconnection network, L2 cache, and dynamic random access memory (DRAM), respectively; we use six values for each, ranging from 0.5 to 1.0 GHz with a step size of 0.1 GHz. The next five parameters relate to the L1 data cache, instruction cache, texture cache, constant cache, and L2 data cache size. A cache has at least four design options including the number of sets, associativity, cache line size, and replacement policy. For simplicity but without loosing generality, we only vary the number of sets to form five different cache sizes. `core_registers` specifies the number of registers of a GPU core. `max_warp_core` controls the maximum number of warps that can concurrently run on a core. `shared_memory` is the size of shared memory of a GPU core, used to share data between warps within a thread block. The last two parameters relate to DRAM queue sizes. `DRAM_queue` specifies the size of the DRAM request queue, and `DRAM_return_queue` sets the size of the DRAM return queue. These two queues provide a buffer to sustain memory-level parallelism.

Note that although we consider a GPU design space by varying the above 15 architecture parameters, this does not imply that QIG is limited to this particular design space. On the contrary, QIG can handle many more GPU architecture parameters as long as enough training data can be collected.

C. Modeling Tools

We use *R*, an open-source software environment [27], to perform SGBRT modeling, SVM, and ANN. Within this environment, we use the "gbm" package published in May 2013 [28] to build our SGBRT-based performance models. In addition, we compare QIG against Starchart [8] using the software publicly released along with the prediction tool. We provide the same training set to all prediction models (ANN, SVM, Starchart, and QIG). Finally, we reimplement and evaluate the PB design of experiment and compare it against QIG.

V. RESULTS AND ANALYSIS

We first evaluate model accuracy of QIG compared to Starchart, SVM, and ANN. We then evaluate QIG's sensitivity to the training set size. We subsequently show and analyze the importance of architecture parameters and their interactions. Next, we conduct a case study to demonstrate how to use QIG. Finally, we compare QIG against the PB design.

We define performance prediction error as follows:

$$\text{IPS}_{\text{err}} = \frac{\left|\text{IPS}_{\text{pred}} - \text{IPS}_{\text{meas}}\right|}{\text{IPS}_{\text{meas}}} \tag{8}$$

with IPS_{pred} the IPS predicted by the performance model, and IPS_{meas} the IPS measured using the GPGPU simulator.

A. Model Accuracy

We first compare QIG against prior work in terms of model accuracy. We compare against Starchart [8], as well as machine learning techniques using ANN and SVM, as previously used in GPU performance prediction [10]. It is important to note that all models are given the same set of 240 training examples. The evaluation is done using the same set of 60 randomly generated GPU architecture configurations; the evaluation set is disjoint from the training set.

Fig. 4 reports per-benchmark IPS prediction errors. Clearly, QIG achieves the lowest IPS prediction error (average error of 4.2%). Starchart, ANN, and SVM yield substantially higher average prediction errors: 23.4%, 23.9%, and 24.5%, respectively, with maximum errors up to 48.9%, 53%, and 72.8%, respectively. QIG yields consistently more accurate predictions with an IPS prediction error below 10% for 24 out of 25 benchmarks; the maximum error of 13.3% is observed for the NE benchmark. The reason for the much higher accuracy obtained using QIG is a result of employing several simple models in an ensemble model rather than a single complex model, which leads to higher accuracy for the given (relatively small) set of training examples.

B. Training Set Size

Ensemble learning relies on training data to build a performance model. The number of training examples needs to be determined upfront, during the training phase, and needs to be balanced: a large number of training configurations increases profiling and training time, whereas a small number of configurations may compromise model accuracy. To understand this tradeoff quantitatively, we have done the following experiment. We start to train the performance models for each kernel using 80 GPU architecture configurations and we increase the training set by 40 each time. All GPU architecture configurations are randomly generated by randomly generating a value for each architecture parameter within its value range (ranges are shown in Table III). To evaluate model accuracy, we randomly generate a set of n GPU architecture

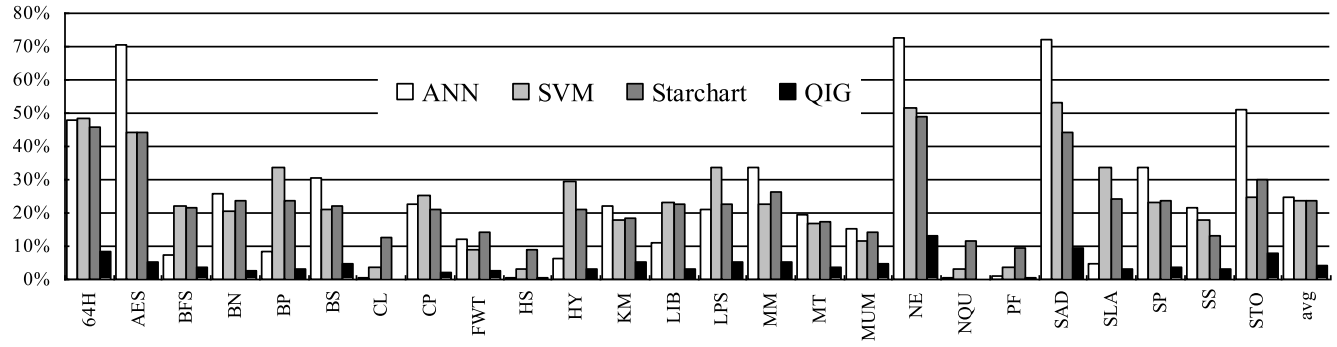

Fig. 4. Model accuracy for QIG compared to Starchart, ANN, and SVM.

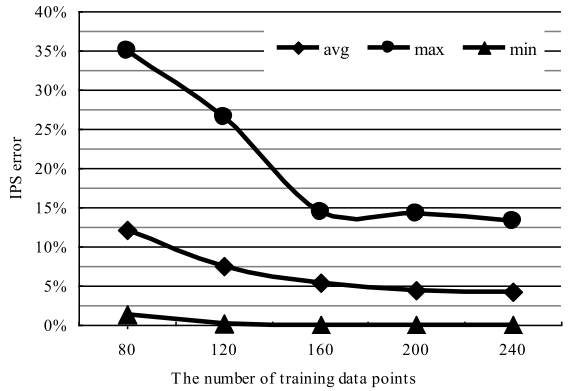

Fig. 5. Maximum, minimum, and average IPS prediction error as a function of training set size.

configurations that is disjoint from the set of training examples; *n* is a quarter of the number of the corresponding training examples.

Fig. 5 quantifies how accuracy is affected by the number of training examples; we show the maximum, minimum, and average IPS prediction error across the 25 experimented GPGPU kernels. As expected, prediction error decreases with an increasing number of training examples. We find the performance model's accuracy to converge once given at least 160 training examples. When given more than 160 training examples, model accuracy continues to improve, albeit at a slower rate. In the remainder of this paper we consider 240 training examples, which yields an average error of 4.2% and maximum error of 13.3%.

Building an ensemble performance model incurs modest time overhead. Simulation time, i.e., collecting performance numbers for the 240 training examples, takes four days on our experimental platform. We run simulations in parallel on four servers, with each server an 8-core hyper-threaded processor; we run 16 simulations in parallel per server, or 64 simulations in parallel in total. Once the profiling data is collected, building the performance model takes only 1–2 s.

C. Parameter Importance

We now quantify parameter importance for the experimented benchmarks. Fig. 6 shows the importance of each experimented GPU architecture parameter for each benchmark. There are a number of interesting observations to be made here.

Observation 1 (There Are Few Dominant GPU Architecture Parameters): Although we consider fifteen GPU architecture parameters, only six of them show up as being the most important parameter across all the experimented benchmarks: number of thread blocks per core, L1 constant cache size, L1 data cache size, shared memory size, core frequency, and interconnect frequency. The other nine parameters never appear as the most important one. The two parameters that are most important across all benchmarks are `core_frequency` (most important parameter for 10 out of 25 benchmarks) and `core_cta` (most important parameter for 11 benchmarks). This indicates that very few GPU architecture parameters have a major impact on performance across the broad set of workloads; a slightly larger group of parameters have a major impact for at least one workload; and the majority of parameters does not have a major impact on any workload.

Observation 2 (There Is a Rank Order in Parameter Importance): Fig. 7 quantifies what we just qualitatively observed: it shows (cumulative) parameter importance. This graph reconfirms our observation that `core_frequency` and `core_cta` are the most important architecture parameter with respect to GPGPU performance, with a relative importance of 20.7% and 16.2%, respectively. This observation reveals at least three insights: 1) increasing clock frequency and/or the amount of parallelism within a core is generally very effective to improve GPGPU performance; 2) among the frequency components, apart from core frequency, increasing frequency of the interconnection network and DRAM will improve performance more than increasing L2 cache frequency; and 3) among the caches, shared memory, and core registers, changing the size of L1 data cache impacts performance more than the other storage units. This rank order in parameter importance may help GPU architects to focus their design optimizations.

Observation 3 (GPGPU Parallelism Varies Widely): If the importance of `core_frequency` is much higher (>40%) compared to the importance of `core_cta` (<5%) for a given kernel, we observe the average IPS across different GPU architectures to be less than 50 giga IPS (GIPS), which is very small for GPGPU workloads, see also Fig. 8. Example

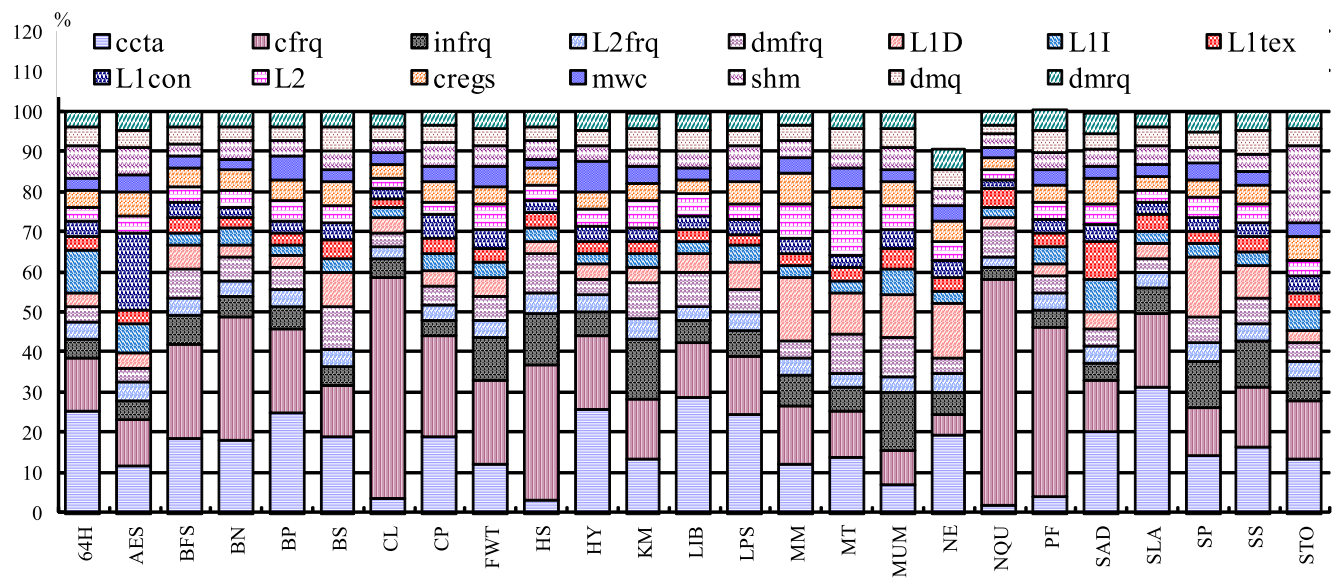

Fig. 6. Importance quantification rank of the experimented GPU architecture parameters for all experimented GPU benchmarks.

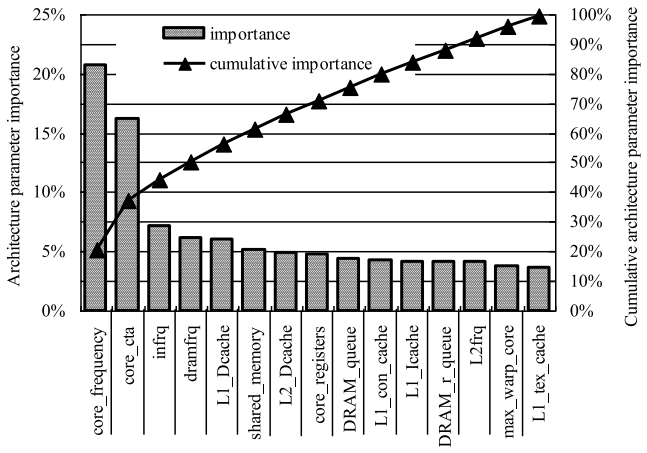

Fig. 7. Average importance of GPU architecture parameters across all the benchmarks.

benchmarks are CL, HS, MUM, NQU, and PF. The common feature for these kernels is that most of the work is done by only a few threads even when a larger number of threads are launched, indicating that it is difficult to effectively parallelize these kernels. In other words, these GPGPU kernels are unable to efficiently leverage the GPU parallel computing resources as they expose limited thread-level parallelism.

On the other hand, an importance of `core_cta` higher than 10% indicates that the respective benchmark exposes more parallelism. This is the case for all the other 20 benchmarks. Note though high levels of thread-level parallelism is not a sufficient reason for high GPGPU performance, see for example BFS, LIB, NE, and SLA which show relatively modest performance according to Fig. 8. This is due to memory intensity (e.g., DRAM frequency is relatively important for BFS and LIB), poor data locality (L1 D-cache size is relatively important for NE), and interconnection network performance (interconnect frequency is relatively important for SLA).

D. Parameter Interactions

We now quantify and analyze pairwise interactions between GPU architecture parameters. Fig. 9(a) shows that `core_frequency` and `core_cta` interact most strongly among all pairwise interactions for the BFS benchmark. Changing these two parameters at the same time affects performance more than changing only one of them at a time. On the other hand, we can definitely ignore the interactions between the other parameters because their interaction intensities are extremely weak. Across our experimented benchmarks, a number of other benchmarks also show a single dominant pairwise interaction, including BFS, *BN*, *BP*, *CP*, LIB, MUM, NE, and SLA. It is interesting to note that the most important pairwise interaction varies across benchmarks, as shown in the upper part of Table IV. Although parameter pair `core_frequency` and `core_cta` is the most important one for most benchmarks, this is not the case for MUM and NE. The most important interaction occurs between DRAM frequency and interconnect frequency for MUM; and between L1 D-cache and the maximum number of thread blocks per core for NE.

It is interesting to note that the most important pairwise interaction does not necessarily occur between the two most important parameters, which is different from CPU architectures [29]. This is the case for MUM. The most important parameters are interconnect frequency and L1 D-cache size, see Fig. 6, however, the most important interaction occurs between interconnect frequency and DRAM frequency. Note though that DRAM frequency is the third most important parameter—some reverse ordering may happen.

As noted above, only 8 out of 25 benchmarks exhibit a single dominant pairwise interaction. The other 17 benchmarks exhibit two or more dominant pairwise interactions.

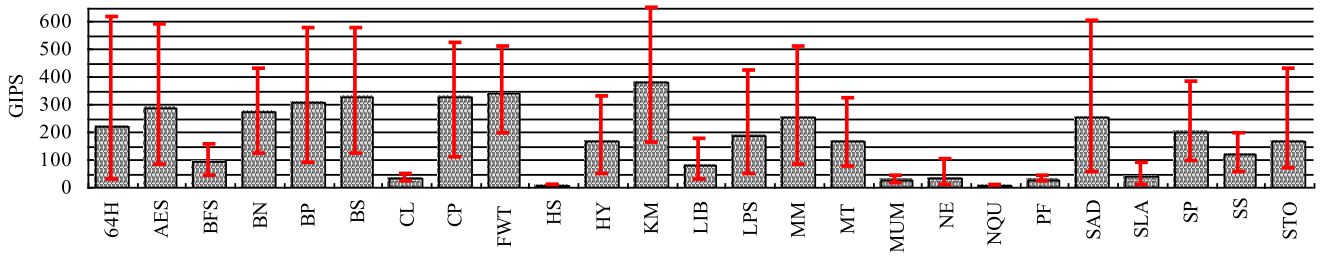

Fig. 8. Maximum, minimum, and average per-benchmark performance (measured in IPS) across the range of GPU architectures in our design space.

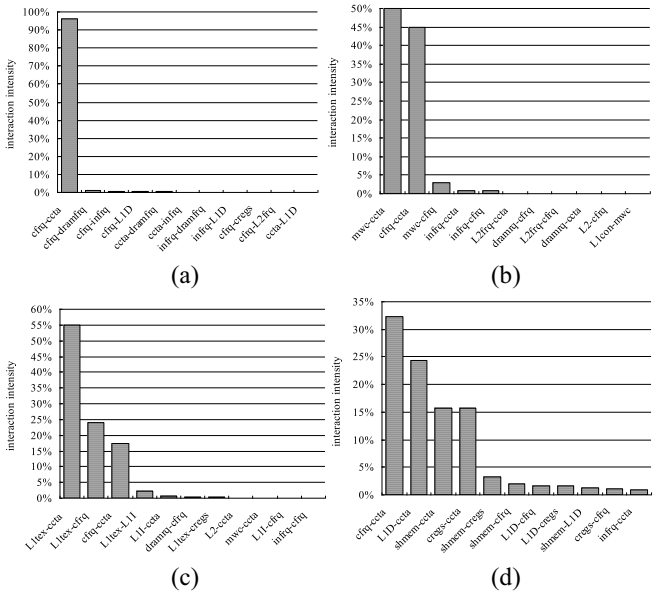

Fig. 9. Example benchmarks showing (a) one, (b) two, (c) three, and (d) four dominant pairwise interactions.

We define a pairwise interaction to be dominant if its importance is larger than 10%. Fig. 9(b) shows HY as an example benchmark with two dominant pairwise interactions; Fig. 9(c) and (d) shows *SAD* and *LP* with three and four dominant pairwise interactions, respectively. Across the experimented benchmarks, AES, FWT, HS, HY, MT, NQU, and STO have two dominant pairwise interactions; 64*H*, CL, *KM*, *MM*, PF, and *SAD* have three; *BS*, *LPS*, and *SP* have four; *SS* has five. The top most important pairwise interaction per benchmark is listed in Table IV.

GPU architects can leverage this information to more efficiently explore the design space. Interactions indicate to the architect that changing one parameter may have limited impact on overall performance as long as the other parameter is kept constant. In other words, the architect needs to consider exploring the effect of simultaneously changing both parameters to see the largest effect on overall performance. Without the analysis provided by QIG, the architect would not know which interactions to consider, and may therefore need to explore and consider parameter sweeps for all possible parameter interactions, which is obviously infeasible. QIG on the other hand identifies the dominant parameter interactions, which is invaluable for the architect to focus on a few parameter interactions, greatly simplifying the design space exploration.

TABLE IV
STRONGEST PAIRWISE INTERACTIONS FOR OUR EXPERIMENTED BENCHMARKS

Benchmark	Parameter 1	Parameter 2	Importance (%)
BFS	cfrq	ccta	96
BN	cfrq	ccta	99.4
BP	cfrq	ccta	84.6
CP	cfrq	ccta	84
LIB	cfrq	ccta	94.5
MUM	dramfrq	infrq	87.8
NE	L1D	ccta	93.5
SLA	cfrq	ccta	89.7
AES	L1con	cfrq	38.1
FWT	cfrq	ccta	42.4
HS	infrq	cfrq	58.9
HY	mwc	ccta	50
MT	L2	ccta	28.4
NQU	L1tex	cfrq	54.9
STO	shmem	ccta	57.5
64H	shmem	ccta	48.3
CL	cregs	cfrq	29
KM	infrq	cfrq	33.3
MM	L1D	ccta	40.2
PF	dramfrq	cfrq	29.7
SAD	L1tex	ccta	54.8
BS	dramfrq	ccta	36.8
LPS	cfrq	ccta	32.2
SP	L1D	ccta	46.7
SS	infrq	ccta	27.2

E. Case Study

We illustrate this further using a case study for the *BP* benchmark. BP is a machine-learning algorithm that trains the weights of connected nodes in a layered neural network. BP is widely used because it is a common algorithm in a variety of areas, such as face recognition, medication, and deep learning, and therefore it may be worth optimizing the GPU architecture for this particular workload.

The two most important GPU architecture parameters for *BP* are `core_cta` and `core_frequency`. Changing the value of `core_cta` from 1 to 8 while keeping all other GPU architecture parameters unchanged to their default value, increases performance from 123 GIPS to 408 GIPS. Changing `core_frequency` from 0.5 GHz to 1 GHz, again while keeping all other parameters constant to their default value, increases performance from 258 GIPS to 487 GIPS. However, changing both parameters at the same time improves performance up to 571 GIPS, which is a

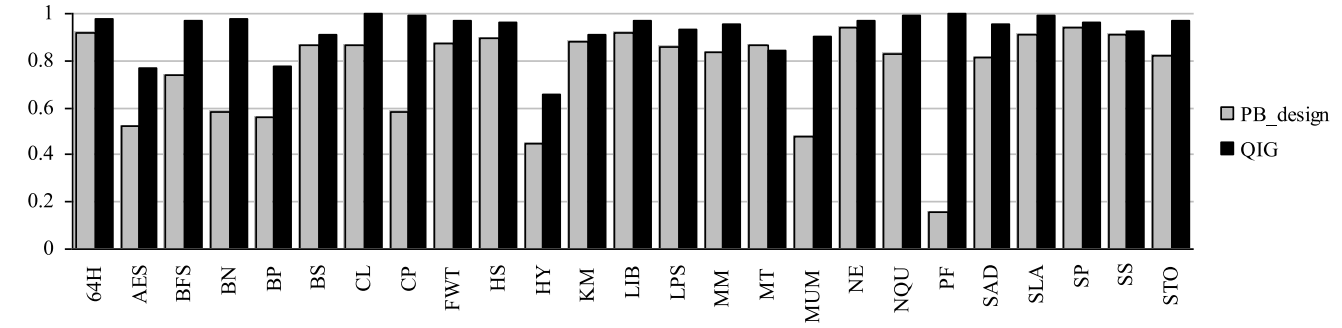

Fig. 10. Rank correlation coefficient between the ranking obtained using QIG and PB versus a simulation reference—higher is better.

significant improvement (17%–40%) over optimizing only a single parameter. More importantly, this case study illustrates that QIG can assist the GPU architect to identify the most important architecture parameters and interactions and accelerate the design process by having the architect focus on the few most important parameters and interactions.

F. Validation and Comparison Against PB

We now validate whether QIG is indeed able to identify the most important GPU architecture parameters. The models underlying to QIG, while accurate, do not provide perfect accuracy, hence some validation is needed. This is done as follows. QIG provides a parameter importance rank per benchmark. We pick the most important parameter according to QIG and run a simulation when changing this one parameter between its maximum and minimum value, while keeping the other parameters unchanged to their default value. We compute the performance delta between the maximum and minimum value. This is done for all parameters in the ranking. We then compute the rank correlation coefficient between the QIG ranking and the ranking obtained through simulations. A rank correlation coefficient close to one indicates that QIG is indeed able to accurately rank the architecture parameters.

This is verified in Fig. 10 which reports this rank correlation coefficient. The rank correlation coefficient exceeds 0.9 for 21 out of 25 benchmarks. The lowest rank correlation coefficient is observed for HY (0.66); the reason is that this benchmark consists of seven kernels which are interdependent, and which QIG does not take into account. The average rank correlation coefficient across all benchmarks equals 0.93. In other words, QIG is indeed able to accurately rank and identify the most important GPU architecture parameters.

The PB design of experiment [30], previously proposed for CPU design space exploration [12], is not as accurate as QIG, with an average rank correlation coefficient of 0.76. For some benchmarks the rank correlation coefficient is as low as 0.16 (PF), and for a handful benchmarks we observe a rank correlation coefficient around 0.5. This indicates that PB can be misleading, which may lead the architect to waste valuable time and effort in exploring uninteresting areas of the design space.

TABLE V
GPU CARDS CONSIDERED IN THE HARDWARE VALIDATION SETUP

GPU	GTX 480	GTX 580	GTX 680	GTX 780
# Core	15	16	8	12
Core Clock (MHz)	700	825	1200	1100
Core V_{DD} (V)	1.06	1.05	1.18	1.15
Architecture	Fermi		Kepler	
Per Core				
# Warp Schedulers	2		4	
# Single Precision Units	32		192	
Registers (KB)	128		256	
L1 Cache (KB)	48/16		48/16	
L2 Cache (KB)	768	768	512	1536
Memory Clock (MHz)	1846	2004	3004	3004
Memory Contollers	6	6	4	6
TDP (W)	250	250	195	250
Technology (nm)	40		28	

VI. HARDWARE VALIDATION

So far, we considered a simulation-based validation of the proposed model. This setup was instigated by the purpose of the model, which is to guide GPU architects at early stages during the design cycle. In this section, we validate the model on real hardware. Because we are unable to consider many hardware configurations for training and evaluating the model, we consider a different setup in which we consider many benchmarks for a single GPU architecture. Instead of building a performance model to predict performance across different GPU architectures for a single benchmark, we now consider a *power* model to predict power consumtpion *across different benchmarks for a single GPU architecture*. In addition to validating QIG on real hardware, this case study also illustrates QIG's versatility, as we now use the QIG methodology to predict power rather than performance, and we do so across benchmarks rather than across architectures.

A. Experimental Setup

We consider four NVIDIA GPU cards including GTX 480, 580, 680 and 780, see Table V. These four cards cover two GPU architectures, namely Fermi (GTX 480 and 580) and Kepler (GTX 680 and 780). GPU cards with the same architecture differ from each other in the number of cores and clock frequency. [A core refers to a streaming multiprocessor (SM) in Fermi, and an SMX in Kepler.] In particular, GTX 480 has 15 cores, with each core running at 700 MHz frequency. While GTX 580 has 16 cores and each core runs at 825 MHz.

TABLE VI
GPU HARDWARE PERFORMANCE COUNTERS

Param	Unit	Comments
gstore_eff	percent	global store efficiency
iss_slot	count	# issued slots
exe_inst	count	# instructions executed
iss_slotu	percent	issue slot utilization
inst_pwp	count	# instructions per warp
active_wp	count	# active warps
inst_rplyohd	percent	instruction replay overhead
wp_occp	percent	achieved warp occupancy
gload_eff	percent	global load efficiency
sm_eff	percent	SM efficiency
active_cyc	count	# active cycles
thd_laun	count	# threads launched
L1_gloadh	count	# L1 cache global load hit
wp_laun	count	# warps launched
sm_elapsedcyc	count	# elapsed SM cycles
L1_gloadm	count	# L1 cache global load miss

We track hardware performance counters using CUPTI [31]. Hardware performance counters vary across GPU architectures. Table VI lists performance counters that are common across the four architectures.

GPU power is measured as follows. All four GPU cards have two sources of power supply: 1) a PCIe slot 12 V power supply and 2) an ATX 12 V power supply. Both power sources have to be counted to accurately measure GPU power consumption. The PCIe power supply contributes over 40% of the total GPU power consumption for the GTX 480, but only less than 10% for the other three cards. We measure instantaneous current and voltage to compute power of each source (PCIe and ATX power supply). We sense the current draw by measuring the voltage drop across a current sensing resistor. The current sensing resistors are inserted in a PCIe riser card for measuring PCIe power and ATX power supply lines for measuring ATX power. This setup allows us to easily switch the target GPU for measuring its power. We use an NI DAQ which is a general-purpose data acquisition card to sample voltage and current at a rate of 2 million samples per second.

We consider the 25 benchmarks listed in Table II. To obtain sufficient training examples, we launch each benchmark with five different thread counts: n the default thread count, as well as $2n$, $4n$, $8n$, and $16n$. This yields 125 benchmark runs in total. We next choose 100 of these as training examples to build a power consumption model for that GPU card, and we employ the remaining 25 items as the testing examples to evaluate the accuracy of the model. We use an equation similar to (8) to calculate the model error for each GPU card.

B. Results

Fig. 11 shows the average power prediction errors for QIG across different benchmarks for the four GPU cards. Our models are fairly accurate for the GTX 480, 580, and 780 with average prediction errors of 6.9%, 5.4%, and 8.9%, respectively. However, the error appears to be higher for the GTX 680—an average error of 18.8%. We believe the reason is that GTX 680 is the first-generation product of the Kepler architecture which employs very aggressive power optimizations, substantially reducing its power consumption

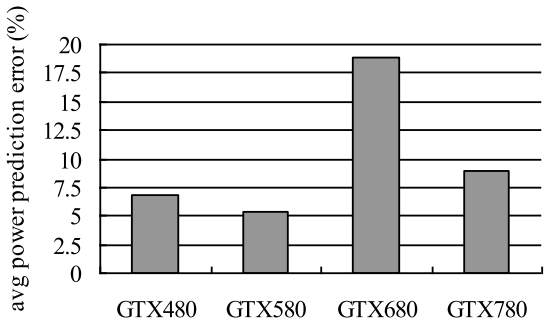

Fig. 11. Average power prediction error of QIG for the four GPU cards.

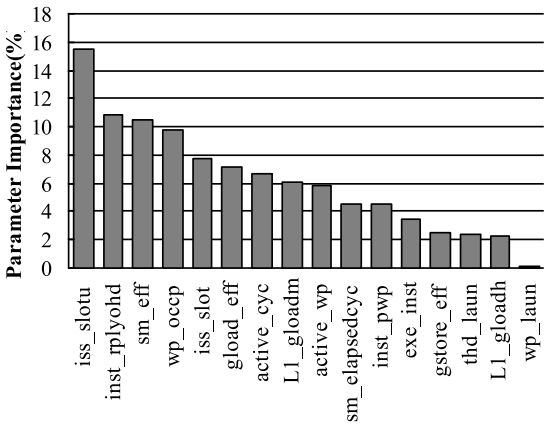

Fig. 12. Importance rank of performance counters on real hardware with respect to power consumption.

over GTX 580. This, in its turn, leads to wide variations in power consumption across the different benchmarks. (For example, the performance and power usage of GTX 680 had to be capped to prevent overly demanding synthetic benchmarks from damaging GPU cards [32].) In the GTX 780, NVIDIA developed GPU Boost, which continuously monitors and adjusts the clock speed and voltage level, leading to less variation in power consumption across different workloads. This leads to a lower prediction error.

Fig. 12 quantifies parameter importance for the various hardware performance counters with respect to power consumption. The *iss_slotu* performance counter appears to be the most important factor. This can be understood intuitively as the utilization of issue slots is a measure for the amount of activity within a core, and hence it has a high impact on dynamic power consumption. Interestingly, QIG reveals that the *wp_laun* performance counter is not important to power consumption. This suggests that the number of warps launched does not indicate high activity in the core.

Fig. 13 quantifies the important pairwise interactions. Interestingly, the most important parameter iss_slotu and the eighth most important parameter *L*1_gloadm interact the most strongly. This can be explained by the observation that L1 cache global load misses incur long latencies, which significantly affects issue slot utilization and logic activity. This also aligns with our observation for the simulation-based experiments: the most important pairwise interaction does not

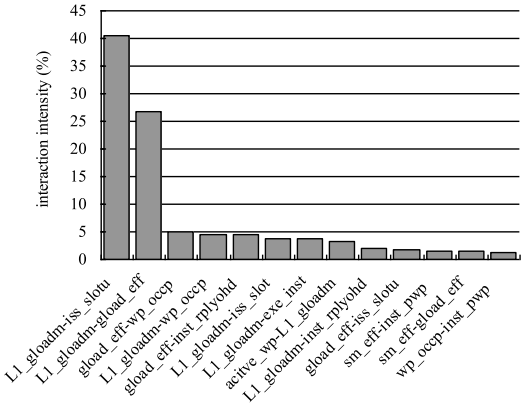

Fig. 13. Interaction intensity rank of performance counters on real hardware with respect to power consumption.

necessarily occur between the two most important architectural parameters. Interestingly, the most important parameter interaction is consistent across all four GPU cards.

VII. Related Work

There exists an extensive body of prior work in empirical performance modeling for CPUs. Joseph *et al.* [33] builded linear regression models from simulation data that relate microarchitectural parameters and their mutual interactions to overall processor performance. In their follow-on work [34] they explore nonlinear regression modeling. Similarly, Lee and Brooks [35] proposed regression models for both performance and energy using splines. They leverage spline-based regression modeling to build multiprocessor performance models [36] and explore the huge design space of adaptive processors [37]. Ïpek *et al.* [38] and Dubach *et al.* [39] builded performance models using artificial neural networks. Lee *et al.* [40] compared spline-based regression modeling against artificial neural networks and conclude that both approaches are equally accurate; regression modeling provides better statistical understanding, while neural networks offer greater automation. Vaswani *et al.* [41] incorporated the interaction between the compiler and the architecture, and build empirical application-specific performance models that capture the effect of compiler optimization flags and microarchitecture parameters. Ould-Ahmed-Vall *et al.* [42] builded tree-based empirical models: the model chooses a path at each node in the tree and finds a linear regression model in the leaves. None of these prior works target predicting parameter importance and their interactions; nor did these works use ensemble learning.

Yi *et al.* [12] used the PB design of experiment to identify the most important architecture parameters in the CPU design space. Our results show that, at least for the GPU design space, PB is not as accurate as QIG. Moreover, PB quantifies the importance of select pairwise interactions only, by construction; QIG on the other hand analyzes all possible pairwise interactions.

A number of empirical models have been proposed for GPUs as well. Stargazer [7] and Starchart [8], [9] use a statistical tree-based partitioning approach to automatically explore the workload optimization space to auto-tune GPGPU applications. Wu *et al.* [10] employed artificial neural networks to estimate performance and power of GPU architectures. Our results show that the performance models constructed by QIG are more accurate than the machine learning models proposed in these prior works. Moreover, these prior works do not rank architecture parameters based on importance and interaction intensity, which is of critical importance for efficient design space exploration.

In addition, principal component analysis (PCA) has been widely used to characterize CPU [43] and GPU architecture performance characteristics [44]. PCA transforms the possibly correlated architectural parameters into uncorrelated variables, called principal components, which are linear combination of the original parameters. The first component captures the highest variance in the data set, followed by the second, and so forth. PCA provides useful insight by analyzing the dominating architecture parameters in the most dominant principal components. Unfortunately, PCA does not readily identify the most important architectural parameters and interactions; QIG on the other hand provides this information by design. Moreover, PCA assumes that the data follows a Gaussian distribution which may require nontrivial preprocessing if the data is non-Gaussian distributed.

All the above studies explore the design space at the architecture level. Several studies explore cross-layer design spaces. In particular, Sarma and Dutt [45] employed a statistical reasoning technique, namely response surfaces, to construct performance models while exploring the architecture design space along with device-level parameters. Response surfaces, unlike QIG, do not readily identify the important variables and their interactions in the design space. Exploring cross-layer design spaces using QIG is subject for future work.

VIII. Conclusion

In this paper, we propose QIG, an ensemble-learning-based approach to quantify the importance of GPU architecture parameters and their interactions. We show that QIG outperforms prior work in the area by a significant margin: QIG's average prediction error is as low as 4.2% for a 15-D GPU architecture design space whereas prior work, such as Starchart and ANN/SVM-based approaches yield an average error of 23+% on average, for the same (relatively small) training set consisting of 240 examples.

We leverage QIG to identify the most important architecture parameters and interactions, which is critical for a GPU architect to efficiently and effectively explore the design space toward the optimum design. We find the PB design of experiment, previously proposed to rank parameter importance, to yield misleading parameter rankings for a number of workloads in our setup; its accuracy in ranking parameters equals 76% on average. QIG on the other hand ranks parameters with 93% accuracy. Moreover, PB can only rank select pairwise parameter interactions, whereas QIG ranks all pairwise interactions, which yields a more accurate and complete view of the design space.

QIG provides a number of interesting insights regarding the GPU design space. GPGPU performance is predominantly determined by a handful architecture parameters including core frequency and the maximum number of thread blocks per core. Some workloads are sensitive to other architecture parameters, such as L1 data cache size and interconnect frequency. Although some workloads are sensitive to a single predominant pairwise parameter interaction, the majority of workloads are sensitive to several (and up to a handful) pairwise parameter interactions, which reinforces the observation that GPU design space is complex and therefore requires efficient techniques, such as QIG to efficiently explore the design space and not waste valuable architect effort and time in uninteresting parts of the design space.

ACKNOWLEDGMENT

The authors are grateful to J. Leng and V. J. Reddi for sharing the power measurement data. The authors would also like to thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] *CUDA Programming Guide, Version 3.0*, NVIDIA Corp., Santa Clara, CA, USA, 2010.

[2] *ATI Stream Technology*, Advanced Micro Devices, Sunnyvale, CA, USA, 2011. [Online]. Available: http://www.amd.com/stream

[3] AMD, *OpenCL*, Khronos Group, Beaverton, OR, USA, 2012. [Online]. Available: http://www.khronos.org/opencl

[4] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Austin, TX, USA, Jun. 2009, pp. 152–163.

[5] J. W. Sim, A. Dasgupta, H. Kim, and R. Vuduc, "A performance analysis framework for identifying potential benefits in GPGPU applications," in *Proc. 17th ACM SIGPLAN Symp. Principles Pract. Parallel Program.*, New Orleans, LA, USA, Feb. 2012, pp. 11–22.

[6] S. Hong and H. Kim, "An integrated GPU power and performance model," in *Proc. ISCA*, St-Malo, France, Jun. 2010, pp. 280–289.

[7] W. Jia, K. A. Shaw, and M. Martonosi, "Stargazer: Automated regression-based GPU design space exploration," in *Proc. IEEE ISPASS*, New Brunswick, NJ, USA, Apr. 2012, pp. 2–13.

[8] W. Jia, K. A. Shaw, and M. Martonosi, "Starchart: Hardware and software optimization using recursive partitioning regression trees," in *Proc. 22nd Int. Conf. Parallel Archit. Compilation Tech. (PACT)*, Sep. 2013, pp. 257–267.

[9] W. Jia, K. A. Shaw, and M. Martonosi, "GPU performance and power tuning using regression trees," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 2, p. 13, May 2015.

[10] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "GPGPU performance and power estimation using machine learning," in *Proc. Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015, pp. 564–576.

[11] J. H. Friedman, "Stochastic gradient boosting," *Comput. Stat. Data Anal.*, vol. 38, no. 4, pp. 367–378, Feb. 2002.

[12] J. J. Yi, D. J. Lilja, and D. M. Hawkins, "A statistically rigorous approach for improving simulation methodology," in *Proc. 9th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2003, pp. 281–291.

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

[14] L. Breiman, "Stacked regressions," *Mach. Learn.*, vol. 24, no. 1, pp. 49–64, Jul. 1996.

[15] J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky, "Bayesian model averaging: A tutorial," *Stat. Sci.*, vol. 14, no. 4, pp. 382–417, Nov. 1999.

[16] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[17] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Mach. Learn.*, vol. 13, no. 1, pp. 1–22, Mar. 1999.

[18] J. H. Friedman and J. J. Meulman, "Multiple additive regression trees with application in epidemiology," *Stat. Med.*, vol. 22, no. 9, pp. 1365–1381, May 2003.

[19] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Boston, MA, USA, 2009, pp. 163–174.

[20] *The NVIDIA CUDA SDK Code Samples*, NVIDIA Corp., Santa Clara, CA, USA, 2013.

[21] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization*, 2009, pp. 44–54.

[22] I. Group. (2007). *The Parboil Benchmark Suite*. [Online]. Available: http://impact.crhc.illions.edu/parbiol.php

[23] L. Žaloudek, L. Sekanina, and V. Šimek, "GPU accelerators for evolvable cellular automata," in *Proc. IEEE Comput. World Future Comput. Service Comput. Cogn. Adapt. Content Pattens*, Nov. 2009, pp. 533–537.

[24] M. Giles and S. Xiaoke. *Notes on Using the NVIDIA 8800 GTX Graphics Card*. [Online]. Available: https://people.maths.ox.ac.uk/gilesm/codes/libor_old/report.pdf

[25] M. Giles. *Jacobi Iteration for a Laplace Discretisation on a 3D Structured Grid*. [Online]. Available: https://people.maths.ox.ac.uk/gilesm/codes/laplace3d/laplace3d.pdf

[26] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," in *Proc. IEEE CVPR Workshop Comput. Vis. GPU*, Jun. 2008, pp. 1–6.

[27] R. Development Team. (2013). *R Language Definition*. [Online]. Available: http://cran.r-project.org/doc/manual/R-lang.pdf

[28] Cran. R Project. (May 2013). *gbm*. [Online]. Available: http://cran.r-project.org/web/packages/gbm/gbm.pdf

[29] J. J. Yi, D. J. Lilja, and D. M. Hawkins, "A statistically rigorous approach for improving simulation methodology," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Anaheim, CA, USA, Feb. 2003, pp. 281–291.

[30] R. L. Plackett and J. P. Burman, "The design of optimum multifactorial experiments," *Biometrika*, vol. 33, no. 4, pp. 305–325, 1946.

[31] *CUDA Tookit Documentation*, NVIDIA Corp., Santa Clara, CA, USA, 2008.

[32] *Introducing The GeForce GTX780*, NVIDIA Corp., Santa Clara, CA, USA, 2013.

[33] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "Construction and use of linear regression models for processor performance analysis," in *Proc. 12th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Austin, TX, USA, Feb. 2006, pp. 99–108.

[34] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "A predictive performance model for superscalar processors," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Dec. 2006, pp. 161–170.

[35] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *Proc. 12th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, San Jose, CA, USA, Oct. 2006, pp. 185–194.

[36] B. C. Lee, J. Collins, H. Wang, and D. Brooks, "CPR: Composable performance regression for scalable multiprocessor models," in *Proc. 41st Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Nov. 2008, pp. 270–281.

[37] B. C. Lee and D. M. Brooks, "Efficiency trends and limits from comprehensive microarchitectural adaptivity," in *Proc. 13th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, Seattle, WA, USA, Mar. 2008, pp. 36–47.

[38] E. İpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz, "Efficiently exploring architectural design spaces via predictive modeling," in *Proc. 12th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, San Jose, CA, USA, Oct. 2006, pp. 195–206.

[39] C. Dubach, T. M. Jones, and M. F. P. O'Boyle, "Microarchitectural design space exploration using an architecture-centric approach," in *Proc. IEEE/ACM Annu. Int. Symp. Microarchit. (MICRO)*, Dec. 2007, pp. 262–271.

[40] B. Lee *et al.*, "Methods of inference and learning for performance modeling of parallel applications," in *Proc. 12th ACM SIGPLAN Symp. Principles Pract. Parallel Program. (PPOPP)*, San Jose, CA, USA, Mar. 2007, pp. 249–258.

[41] K. Vaswani, M. J. Thazhuthaveetil, Y. N. Srikant, and P. J. Joseph, "Microarchitecture sensitive empirical models for compiler optimizations," in *Proc. Int. Symp. Code Gener. Optim. (CGO)*, San Jose, CA, USA, Mar. 2007, pp. 131–143.
[42] E. Ould-Ahmed-Vall, J. Woodlee, C. Yount, K. A. Doshi, and S. Abraham, "Using model trees for computer architecture performance analysis of software applications," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2007, pp. 116–125.
[43] L. Eeckhout, H. Vandierendonck, and K. D. Bosschere, "Quantifying the impact of input data sets on program behavior and its applications," *J. Inst. Level Parallelism*, vol. 5, no. 2, pp. 1–33, 2003.
[44] N. Goswami, R. Shankar, M. Joshi, and T. Li, "Exploring GPGPU workloads: Characterization methodology, analysis and microarchitecture evaluation implications," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, 2010, pp. 1–10.
[45] S. Sarma and N. Dutt, "Cross-layer exploration of heterogeneous multicore processor configurations," in *Proc. 28th Int. Conf. VLSI Design 14th Int. Conf. Embedded Syst.*, Bengaluru, India, 2015, pp. 147–152.

Zhibin Yu (M'07) received the Ph.D. degree in computer science from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2008.

He is currently a Professor with the Shenzhen Institutes of Advanced Technology, Chinese Academy of Science, Shenzhen, China. His current research interests include computer architecture, workload characterization and generation, GPGPU architecture, and big data processing.

Dr. Yu was a recipient of the Outstanding Technical Talent Program of Chinese Academy of Science in 2014, the "Peacock Talent" Program of Shenzhen City in 2013, the First Award in Teaching Contest of HUST Young Lectures in 2005, and the Second Award in Teaching Quality Assessment of HUST in 2003. He serves for ISCA 2013, MICRO 2014, ISCA 2015, and HPCA 2015. He is a member of ACM.

Jing Wang received the Ph.D. degree from Peking University, Beijing, China, in 2011.

From 2011 to 2013, she was a Post-Doctoral Fellow with the Department of Computer Science, Peking University. She is currently an Assistant Professor with the Electrical and Computer Engineering Department, Capital Normal University, Beijing. Her current research interests include computer architecture, energy-efficient computing, high-performance computing, and hardware reliability and variability.

Lieven Eeckhout (SM'02) received the Ph.D. degree from Ghent University, Ghent, Belgium, in 2002.

He is a Professor with Ghent University. He published a Morgan and Claypool synthesis lecture monograph in 2010 on performance evaluation methods. His current research interests include computer architecture with a specific emphasis on performance evaluation methodologies and dynamic resource management.

Dr. Eeckhout was a recipient of the two IEEE Micro Top Pick Awards and a Best Paper Award at ISPASS 2013. He was the Program Chair for HPCA 2015, CGO 2013, and ISPASS 2009. He currently acts as the Editor-in-Chief of IEEE MICRO, and as Associate Editor for the IEEE TRANSACTIONS ON COMPUTERS and *ACM Transactions on Architecture and Code Optimization*

Chengzhong Xu (F'95) received the Ph.D. degree from the University of Hong Kong, Hong Kong, in 1993.

He is currently the Director with the Institute of Advanced Computing and Data Engineering, Shenzhen Institute of Advanced Technology, Chinese Academy of Science, Beijing, China. His current research interests include parallel and distributed systems, and cloud computing.

Dr. Xu was a recipient of the Outstanding Overseas Scholar Award of NSFC. He serves on a number of journal editorial boards, including the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON CLOUD COMPUTING, the *Journal of Parallel and Distributed Computing*, and *China Science Information Sciences*.