Accelerating Architectural Simulation Via Statistical Techniques: A Survey

Qi Guo, Tianshi Chen, Yunji Chen, and Franz Franchetti

***Abstract*—In computer architecture research and development, simulation is a powerful way of acquiring and predicting processor behaviors. While architectural simulation has been extensively utilized for computer performance evaluation, design space exploration, and computer architecture assessment, it still suffers from the high computational costs in practice. Specifically, the total simulation time is determined by the simulator's raw speed and the total number of simulated instructions. The simulator's speed can be improved by enhanced simulation infrastructures (e.g., simulators with high-level abstraction, parallel simulators, and hardware-assisted simulators). Orthogonal to these work, recent studies also managed to significantly reduce the total number of simulated instructions with a slight loss of accuracy. Interestingly, we observe that most of these work are built upon statistical techniques. This survey presents a comprehensive review to such studies and proposes a taxonomy based on the sources of reduction. In addition to identifying the similarities and differences of state-of-the-art approaches, we further discuss insights gained from these studies as well as implications for future research.**

***Index Terms*—Architectural simulation, design space exploration (DSE), regression, statistical methods.**

I. Introduction

When designing a processor, designers have to estimate architectural behaviors of the design before implementation and manufacture, so that the processor can meet specific design objectives. As one of the most prevalent methodologies for addressing this problem, architectural simulation has been extensively deployed since it offers designers a balance of cost, timeliness, and flexibility [1]. Application scopes of architectural simulation include but are not limited to performance evaluation, functional validation, design space exploration (DSE), and assessment of architectural innovations. In industry, all leading processor manufacturers have devised their own cycle-accurate simulators to the design of processor. For example, IBM maintains Mambo simulation environment [2], which is designed for IBM PowerPC systems, ranging from embedded system (e.g., IBM's 32-bit embedded 405GP [3]) to supercomputer (e.g., BlueGene). AMD develops SimNow simulator [4] that emulates AMD Athlon 64 and AMD Opteron uniprocessor and multiprocessor systems. Intel also uses HAsim [5] to evaluate future processor products. Architectural simulation also plays a critical role in academic research of computer systems, since it enables validations of novel research ideas without manufacturing real chips. One piece of evidence is that more than 80% papers presented at premier conferences of computer architecture research, ISCA/HPCA/Micro (2009–2011), utilized architectural simulators to validate the proposed ideas.

While the importance of architectural simulation has been widely acknowledged, the speed of architectural simulator is notoriously slow. Specifically, the speed of a cycle-accurate simulator is typically between 1 KIPS (thousand instructions per seconds) and 1 MIPS (million instructions per second) [6], which are several orders of magnitude slower than the native execution. To improve the simulator's speed, many concrete simulators have been proposed, including simulators with high abstract level (e.g., Sniper [7], [8]), parallel simulators (e.g., P-Mambo [9], SlackSim [10], and Graphite [6]), and hardware-assist simulators (e.g., FPGA-Accelerated Simulation Technologies [11], ProtoFlex [12], research accelerator for multiple processors [13], and ScalableCore [14]). An overview of the above simulators can be found in [6] and [8]. Nevertheless, even with the help of such enhanced simulators, architects and researchers still cannot afford the total simulation time in their practice. The reason is that the total simulation time is determined not only by the simulator's speed, but also by the total number of simulated instructions.

To reduce the total number of simulated instructions, recently many approaches have been proposed. The key idea of these approaches is to extract a small subset of the representative instructions from the complete set of simulated instructions by using various statistical techniques (e.g., sampling theory, regression analysis, and machine learning). Since these approaches only need to simulate a reduced number of instructions, we called them as partial simulation approaches hereinafter. Apparently, partial simulation approaches are completely orthogonal to the afore-mentioned approaches that need to greatly modify the simulators, and they can be easily applicable to state-of-the-art simulators to further improve simulation efficiency. In the rest of this survey, we will comprehensively review the partial simulation approaches.

Formally, the total number of simulated instructions (I) can be expressed as $I = N \times T$, where N is the number of instructions per simulation run and T is the number of

Manuscript received December 20, 2014; revised May 2, 2015; accepted June 22, 2015. Date of publication September 24, 2015; date of current version February 24, 2016. This work was supported in part by the Defense Advanced Research Projects Agency PERFECT program under Grant HR0011-13-2-0007, in part by the National Science Foundation of China under Grant 61221062, Grant 61303158, Grant 61432016, and Grant 61473275, in part by the 973 Program of China under Grant 2015CB358800, and in part by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDB02040009. This paper was recommended by Associate Editor S. Kim. *(Corresponding Author: Yunji Chen.)*

Q. Guo and F. Franchetti are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA.

T. Chen and Y. Chen are with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100092, China (e-mail: cyj@ict.ac.cn).

Digital Object Identifier 10.1109/TCAD.2015.2481796

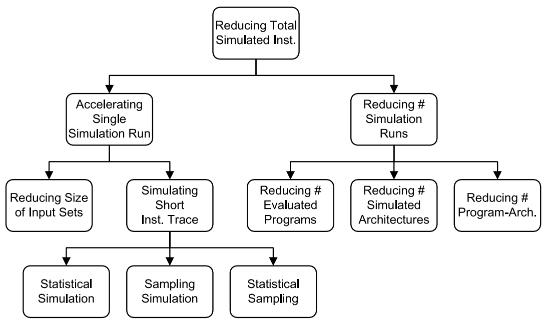

Reducing Total Simulated Inst.
Accelerating Single Simulation Run
Reducing # Simulation Runs
Reducing Size of Input Sets
Simulating Short Inst. Trace
Reducing # Evaluated Programs
Reducing # Simulated Architectures
Reducing # Program-Arch.
Statistical Simulation
Sampling Simulation
Statistical Sampling

Fig. 1. Taxonomy of partial simulation approaches.

simulation runs.[1] Based on this observation, we propose a detailed taxonomy as shown in Fig. 1, where existing partial simulation approaches are categorized into two classes. The first class focuses on reducing the number of instructions per simulation run (N), and the second class tries to reduce the number of simulation runs (T). In the first class, the approaches are further classified into two kinds, since N is mainly determined by the simulated program and its related input sets. The first kind of approaches reduce the size of input sets to reduce the simulation time, for example, the execution time of train input set could be only 25% of that of the ref input set [15]. The second kind of approaches reduce the simulation time by detailedly simulating a short instruction trace instead of the original instruction trace. There are three main approaches to achieve this goal: 1) statistical simulation; 2) sampling simulation; and 3) statistical sampling. Statistical simulation only simulates a small program synthesized from the original program based on its statistical characteristics. Sampling simulation only simulates representative traces from the original program trace. Statistical sampling simulates a small subset of instruction traces from the original program trace via statistical sampling theory. According to the reported results, these approaches can reduce the simulated instructions by about two or three orders of magnitude.

The second class of approaches focus on reducing the number of simulation runs (T). In practice, T is further determined by the number of evaluated programs and the number of simulated architectures. The main way to reduce the number of evaluated programs is called benchmark subsetting, which finds representative benchmarks among a large set of benchmarks with statistical techniques such as principle component analysis and clustering analysis. On the other hand, to reduce the number of simulated architectures, several paradigms have been proposed, such as regression modeling, ranking modeling, heuristic searching, and analytical modeling. These approaches are mainly used for DSE, i.e., finding the optimal architectures from a large architectural design space in the presence of design constraints. The key idea of these approach is to selectively simulate only a small number of architectures rather than the whole design space. In addition to the above approaches, researchers also propose several approaches to reduce both the number of evaluated programs and the number of simulated architectures simultaneously.

[1]Traditionally, the practitioners need to undergo multiple simulation runs to obtain the desired results. For example, the researchers have to run several different benchmarks on the simulators to sufficiently validate their ideas. Another more notable example is that the architects have to run thousands, even millions of simulations to explore the exponential architectural design space to find the optimal design tradeoffs.

II. Background

A. Architectural Simulator

The architectural simulators that are widely used in computer architecture research can be roughly divided into two kinds, including trace-driven simulator and execution-driven simulator.

1) Trace-Driven Simulator: In traditional trace-driven simulators, at first, traces (i.e., time-ordered records of dynamic execution stream of instructions) are collected from real applications. Then, such traces are served as inputs to drive the trace simulators to model the behaviors of the target architectures, where functional simulation is separated from detailed timing simulation. Since applications are not functionally executed in the trace-driven simulators, it can obtain a slight speed advantage over the execution-driven simulators [1]. Moreover, trace-driven simulators are relatively easier to implement than other alternatives, since they only need to consider minimal amount information that is critical to the replay of programs. However, the main drawback of trace-driven simulators is that they cannot accurately model the speculation which is common in modern superscalar architectures. Moreover, in the multicore era, trace-driven simulators suffer from the inability of capturing timing-dependent behaviors of multithreaded applications [16].

2) Execution-Driven Simulator: The execution-driven simulators combine the functional simulation and detailed timing simulation together to obtain the simulated results. Execution-driven simulators take an executable binary rather than a trace as the input, and they are extensively used in the design of modern processors. One of the most famous execution-driven simulators is SimpleScalar, which was first written by Todd Austin in 1994, and its first release was assembled, debugged, and documented by Doug Burger in 1996 at the University of Wisconsin–Madison [17]. SimpleScalar provides five execution-driven simulators in the release, ranging from the simplest (and fastest) functional simulator to a detailed, out-of-order (OOO) superscalar processor simulator. RSIM [18] is another execution-driven simulator mainly targeting shared-memory multiprocessors with instruction-level parallelism (ILP) processors. It executes the instructions OOO together with the timing model, and it supports accurate memory consistency and wrong path simulation. Actually, RSIM sacrifices simulation speed for accuracy.

To facilitate the design of multicore processors, several multicore execution-driven simulators have been developed. One of the most notable examples is SuperESCalar simulator (SESC). SESC can characterize a variety of architectures, including dynamic superscalar processors, chip-multiprocessors (CMPs), processor-in-memory, and speculative multithreading architectures. The main targets of SESC are to provide a fastest simulator, make the code understandable and extensible, and offer many flexible configurations for architects. Simics is a commercial full system simulator [19], and it can support detailed simulation of multiprocessors with the help of general execution-driven multiprocessor simulator (GEMS) [20]. Currently, GEMS is no longer maintained, and the main efforts are shifted to the development of gem5 simulator system [21], which merges the advantages of M5 [22] and GEMS. The gem5 simulator provides a highly configurable simulation framework, multiple instruction set architectures, diverse CPU models, and a flexible memory system for modeling modern processors. MARSSx86 is a fast, cycle-accurate, full-system multicore simulator for x86-64 architecture [23], and it is built upon PTLsim [24].

Another kind of simulators heavily rely on dynamic binary instrumentation tools (e.g., Pin [25]) to feed information to the timing models. For example, CMP$im [26] employs the Pin tool to generate memory information on-the-fly to feed the cache model for the investigation of the cache system on multiprocessors. Other examples include Graphite [6], Sniper [8], and ZSim [27]. Since the applications are functionally emulated/executed on the host natively, these simulators are relatively fast. For example, ZSim can achieve up to 300 MIPS to model a chip with 1024 OOO cores.

B. Usage of Architectural Simulation

The application scopes of architectural simulation are very broad, including performance evaluation, functional validation, DSE, assessment of architectural innovations, as well as software performance tuning.

1) Design Space Exploration: DSE is widely considered as one of the fundamental problems during the design of computer systems. The silicon advances significantly increase the complexity of processors, which leads to a large number of design parameters (e.g., cache size, reorder buffer size, and number of cores) to decide. This problem is further exacerbated for heterogeneous architecture [28]. In DSE, cycle-accurate architectural simulators are indispensable tools for evaluating complicated and subtle design tradeoffs with respect to large design spaces, and handling various design constraints (e.g., power/area/thermal constraints and quality of service requirements). In order to achieve efficient and effective DSE, architectural simulators should be portable, flexible, accurate, and fast.

2) Assessment of Architectural Innovations: Assessment of architectural innovations relies heavily on architectural simulation during the early design phase. In the absence of a real system, architectural simulation offers a cost-effective way of evaluating architectural innovations. For instance, photonic interconnection network has been acknowledged as one of the most promising techniques to provide low-latency, ultrahigh-bandwidth, and low-power network for intracore communications. However, silicon-photonic integration is still expensive and has to face several challenges in manufacture. Under this circumstance, several photonic on-chip interconnect network simulators, e.g., PhoenixSim [29], have been developed to evaluate the effectiveness of such architectures.

3) Software Performance Debugging and Tuning: Although the architectural simulation is mainly employed for the architectural design of processors, it can also support various tasks in software engineering, including performance analysis, debugging, and testing of software. For example, Albertsson and Magnusson [31] proposed to leverage full system simulation (i.e., Simics simulation environment [19]) to build a temporal debugger to analyze real-time properties of software [30], [31]. In the above debugger, the predictable, nonintrusive debugging environment for checking temporal errors should be attributed to the deterministic characteristics of deployed architectural simulators.

C. Statistical Methods

1) Statistical Concepts: Here, we introduce several basic statistical concepts that are closely related with partial simulation approaches. Sampling theory and hypothesis testing are two of the core concepts in statistic. Both of them are utilized to provide statistically rigorous approximation to the estimated results offered by simulators.

TABLE I
SAMPLING VARIABLES

	Population Variables	**Sample Variables**
size	N	n
mean	μ	$\bar{x}$
standard deviation	σ	s
variation coefficient	c_v	$\widehat{c}_v$
confidence level	-	C
confidence interval	-	$\pm\theta$

a) Sampling theory: In practice, statistical analysis is conducted on a chosen subset of the entire population, since it is usually impossible to get all data from the entire population. A subset of the entire population is also called a sample, and how to choose a sample to estimate the property of the entire population is investigated by sampling theory. In our application, sampling theory is used to determine a proper sample of instructions that can best characterize the property of all simulated instructions [32].

As a basic type of sampling techniques, simple random sample (SRS) is widely used, where each individual of the population has the same possibility to be chosen. Table I gives the statistical variables that are used to characterize the property of population and sample in SRS. In SRS, a sample of n individuals are selected at random from a population of N individuals. The analysis of the entire population is usually conducted on the n sampled individuals, and the true population mean μ is estimated by the sample mean $\bar{x}$. The variation coefficient c_v is the standard deviation σ normalized by mean, that is, $c_v = \sigma/\mu$ and $\widehat{c}_v = s/\bar{x}$ are calculated for the population and the sample, respectively. Similar to the standard deviation, variation coefficient can also be used to measure the dispersion of a probability distribution. However, the standard deviation can be directly used to construct confidence interval, which is expressed as an interval to indicate the reliability of a mean estimate. In more detail, confidence interval can be calculated as $\theta = z(\sigma/\sqrt{n})$, where z is the upper $(1-C)/2$ critical value for the standard normal distribution. Here C could be interpreted in such a way that there is a C probability that $\bar{x}$ is with the range of $\pm\theta$ of μ. Thus, for a sample with given standard deviation σ and size n, the achieved confidence interval varies along with specified confidence level.

b) Statistical hypothesis testing: Another important concept in statistics is statistical hypothesis testing (SHT), which is a systematic method to make decisions using experimental data. The most representative usage of SHT in computer architecture is to evaluate the performance comparisons of different simulation runs [33], even native computer systems [34].

There are several components to define when conducting the hypothesis testing, as shown in Table II. Hypothesis testing always begins with a statement of the value of a population statistic containing the condition of equality, e.g., $\mu = 0.5$, and such statement is called as null hypothesis, denoted as H_0. On contrary, the alternative hypothesis is the statement that must be accepted if the null hypothesis H_0 is rejected (i.e., false). Based on null hypothesis, one can calculate the probability of the observation under H_0, which is called p-value. Once p-value is less than a user specified significance level,

TABLE II
TERMS IN SHT

	Term	Notes
null hypothesis	H_0	A statement of a population parameter
alternative hypothesis	H_1	The statement accepted if H_0 rejected
significance level	α	degree of confidence
p-value	p	determine whether or not reject H_0

α (the most commonly used values are 5% and 1%), the null hypothesis will be rejected at the given level of significance. In this condition, the alternative hypothesis will be accepted to reach a statistically rigorous conclusion.

2) Regression Analysis: Regression analysis focuses on characterizing the relationship (i.e., regression function) between a dependent variable and one or more independent variables. It is mainly used for prediction and forecasting, e.g., predicting the performance (dependent variable) of architectural parameters (independent variables) in architectural simulation, which may partially overlap with the field of machine learning that will be detailed later. There are two kinds of regression techniques: 1) parametric and 2) nonparametric regression. Parametric regression implies that the form of regression function is predefined, only the corresponding parameters should be estimated from the observed data. On contrary, nonparametric regression does not give assumptions on the concrete form of the regression functions, thus, large sample size is required to construct the model structure and model estimates. Actually, both parametric (e.g., linear regression) and nonparametric regression (e.g., spline function) have been utilized for performance/power modeling to accelerate architectural simulation.

Linear regression is one of the most notable parametric regression techniques, where the dependent variable is a linear combination of the parameters (rather than the independent variables). A general linear regression model can be written as

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i$$

where x_{ij} is the ith observation on the jth independent variable, and ϵ is an error term. The main task of regression analysis is to determine the concrete value of $\beta_j (1 \leq j \leq p)$ via, typically, least square approaches.

Interpolation is a technique to create new data points within the range of a set of already known data points, which is basically a nonparametric regression technique. Spline interpolation is a special interpolation technique where smooth piecewise polynomial function, called spline, is utilized to predict the value of dependent variable. Spline is divided into polynomial intervals, and such intervals are connected by knots. Actually, the number and location of knots are critical to the approximation between the spline function and real data.

3) Machine Learning: Machine learning focuses on the study of algorithms that can automatically improve the performance through experience [35]. It has already been widely used in the community of computer architecture, e.g., resource allocation [36], task scheduling [37], hardware reconfiguration [38], and architectural optimization [39]. Here, we present several basic but critical concepts of machine learning.

In a traditional machine learning problem, given a training set consisting of n training data, each data is a pair of objects denoted as $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i \in \Re^d$ is the training example and $y_i \in \Re$ is the corresponding label of $\mathbf{x}_i$, learning algorithms (learners) try to find an appropriate inferred function (or model) $f : \Re^d \rightarrow \Re$ that implements the optimal mapping. Then, for a new example $\mathbf{x}_j$, learned function could generate the predicted output y_j as $f(\mathbf{x}_j)$. In particular, for a binary classification task the label y_i is a binary variable such as $y_i \in \{-1, 1\}$. In contrast, in a regression task the label y_i is a continuous variable. Actually, such a learning problem is a typical supervised learning problem since the learning process is performed on supervised training data, i.e., training examples and their corresponding labels. On the other hand, unsupervised learning indicates that the training set only contains training examples without labels, and the learning process is conducted directly on the original example $\mathbf{x}_i$. According to these concepts, it is easy to derive the definition of semisupervised learning, that is, the training set contains both labeled and unlabeled training examples. Actually, all three kinds of learning techniques stated above have been leveraged to accelerate the architectural simulation by researchers in the community of computer architecture.

III. Accelerate Single Simulation Run

As stated, the simulation cost of a single simulation run is primarily determined by the evaluated program and its input sets. Thus, there are two categories of approaches to accelerate a single simulation run, that is, reducing size of input sets and simulating a short instruction trace. We will detail each category in the following sections.

A. Reducing Size of Input Sets

The first category of approaches to accelerate single simulation run is to reduce the size of input sets. MinneSpec [40] is one of the earliest investigations that fall in this category, which is a reduced input set for standard SPEC CPU2000 benchmark suite. SPEC CPU2000 benchmark suite contains three standard data sets: 1) test; 2) train; and 3) ref, whose scales increase in order. Among these data sets, programs running with the ref data set exhibit the most similar behaviors to real-life applications, and MinneSpec tries to provide a small data set to reasonably mimic the behavior of the ref data set. MinneSpec is developed by modifying the input commands, providing new input files, or modifying/truncating/replacing ref files for all benchmarks in SPEC CPU2000. By comparing several statistical characteristics, such as the instruction mixture and memory behaviors, of MinneSpec with the original ref data set, the authors found that MinneSpec can reasonably mimic the behavior of ref data sets of SPEC CPU2000 benchmarks. However, although MinneSpec has been recognized by SPEC and distributed with Version 1.2 of SPEC CPU2000, it is not widely used for evaluating existing modern architectures. Eeckhout *et al.* [41] later validated that such reduce input sets are only representative for some programs, but not for others.

Hsu *et al.* [42] compared several statistical characteristics, e.g., instruction per cycle (IPC), execution paths, and path coverage, among test, train, and ref, and they found that both test and train input sets have significant different characteristics to that of ref data sets for several benchmarks. Thus, such input data sets may not be suitable for profile-based optimization or validation of research ideas, since the results of performance evaluation conducted on these data sets could be misleading.

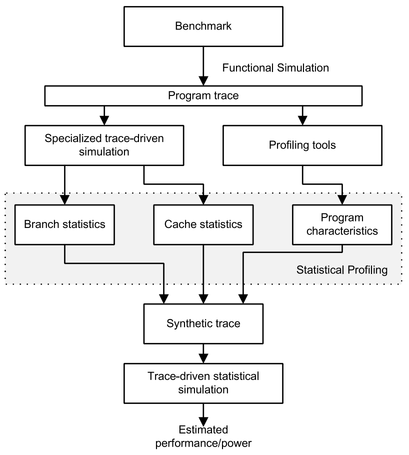


Fig. 2. General framework of statistical simulation process.

B. Simulating Short Instruction Trace

This category of approaches focuses on simulating a shorter instruction trace than the original instruction trace of a given program and its inputs. There are three different approaches, i.e., statistical simulation, sampling simulation, and statistical sampling.

1) Statistical Simulation: The basic idea of statistical simulation [43], [44] is to reconstruct a synthetic, small program based on the statistical profiles (e.g., distribution of instruction types and branch behaviors) that are extracted from the detailed simulation of the original benchmark. Thus, simulation efficiency can be improved by replacing the original large-scale program with a small synthetic version. Typically, statistical simulation consists of four steps: 1) program trace generation; 2) statistical profiling; 3) synthetic-trace generation; and 4) trace-driven simulation [44], as shown in Fig. 2. At first, the program traces of given benchmarks can be generated by functional simulation. Then, the profiling tools extract a set of statistical characteristics from such traces, and there are two kinds of statistical profiling tools to accomplish this task. The first is the microarchitecture-independent profiling tools that only analyze the functional operations of the program instructions and produce microarchitecture-independent characteristics, such as instruction mix and instruction dependencies. The second tool is specialized cache/branch predictor simulators to collect the cache and branch behaviors from the program trace, and these characteristics are closely related to concrete architectures. The complete set of such statistical characteristics are then used to generate a synthetic trace with the same statistical properties. Finally, such synthetic traces can be efficiently executed on a trace-driven simulator to obtain corresponding performance/power results. Obviously, the effectiveness of statistical simulation should be measured by comparing the simulated performance/power results of the original program traces and the corresponding synthetic traces. For SPECint95 benchmarks, statistical simulation results in about 10% and 5% relative error for IPC and power, respectively, while the number of simulated instructions can be reduced by several orders of magnitude.

The selection of statistical characteristics is very crucial to the accuracy of statistical simulation. Initially, basic block size, instruction mixture, cache hit rate, etc., are treated as statistical characteristics in a statistical simulator high-level synthesized (HLS) [45]. In addition to such statistical characteristics, Eeckhout *et al.* [46] proposed to utilize control flow graph to characterize the control flow behaviors to enhance the accuracy of statistical simulation. Besides, they also showed that delayed update[2] should be considered when characterizing branch behavior. Based on the experiments conducted on an eight-way superscalar processor, the average error is 6.6% and 4% for predicting performance and energy, respectively, using SPECint 2000 benchmarks.

To facilitate the design of CMPs, Genbrugge and Eeckhout [48], [49] further extended statistical simulation to multithreaded programs. They proposed several statistical characteristics of the cache access behaviors, for example, the probabilities of set access and per-set least recently used stack distance. Besides, they also showed that it is important to model the time-varying behavior to accurately capture the conflict behavior in accessing shared resources. Experimental results demonstrate that the average IPC prediction error is less than 5.5% while the simulation speed-up can achieve 40× to 70×.

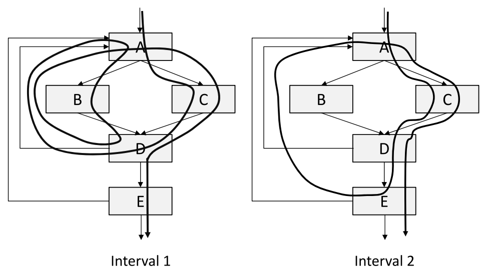
Hughes and Li [50] also extended statistical simulation to multithreaded programs for CMPs. They proposed to build synchronized statistical control flow graphs with the behaviors of interthread synchronization and sharing patterns, so as to capture the interactions between threads. Moreover, the memory access is captured by thread-aware data reference models, and the branch behavior is captured by wavelet-based branching models. For the evaluated SPLASH-2 benchmarks, this approach results in IPC errors from 3.8% to 9.8% and the simulation time has been reduced by more than an order of magnitude.

Recently, graphics processing units (GPUs) have been employed to speed-up the general-purpose applications, leading to GPGPU computing. However, as GPGPU architectures have many parallel hardware threads, current sequential cycle-accurate GPU simulators such as GPGPU-Sim [51] take a long time for simulation. To speed-up the simulation of GPGPU architectures, Yu *et al.* [52] proposed to synthesize small benchmarks with a reduced number of iterations compared against the original workloads. To generate such small benchmarks, several static and dynamic characteristics are collected from the original workloads. Experimental results show that the speed-up of this approach is 88× on average compared with GPGPU-Sim.

2) Sampling Simulation: Different from statistical simulation that reconstructs new small synthetic programs, sampling simulation directly extracts a small fraction of representative instructions from the original instruction trace. Compared with the full simulation with billions of instructions, this approach only needs to simulate several millions of instructions. The most well-known approach is SimPoint [53].

SimPoint is proposed by Sherwood *et al.* [53], [54] based on the concept of basic block vectors (BBVs). For a given interval of execution (e.g., 100 million instructions), a BBV is an array where each element represents a basic block in the original program. More specifically, each element in a BBV is the weighted count of how many times the related basic block has been executed in this interval. Thus, BBV is

[2]It refers to the timing to update the predictor with branch results [47]. In a typical pipelined processor, the branch is predicted in the fetch stage, but the predictor is updated in the commit stage.



	A B C D E
Interval 1	<3, 1, 2, 3, 1>
Interval 2	<2, 0, 2, 2, 2>
Manh. Dist	<1, 1, 0, 1, 1> = 4

Fig. 3. Illustrative example to compute the similarity between two execution intervals.

an architecture-independent metric that can characterize the behavior of arbitrary execution intervals in a program. To compare the behaviors of execution intervals, the Manhattan distance is utilized to measure the similarity between different BBVs. An illustrative example to demonstrate the computation of similarity between two execution intervals is shown in Fig. 3. There are two execution intervals in this example. For interval 1, since basic block A has been executed for three times, the value of its corresponding element in BBV is 3. We can easily determine the value of other elements in the BBVs of intervals 1 and 2. Then, the similarity (i.e., Manhattan distance) of these two execution intervals is calculated as 4.

Based on the similarity metric of different intervals, K-means clustering analysis is conducted to find the representative intervals that are closest to the center of corresponding cluster for simulation, and these simulation intervals are called simulation points. After conducting detailed simulations on the set of simulation points, a weighted average for IPC can be computed to approximate the performance of the entire program's execution. Experimental results show that by creating multiple simulation points (≤ 10), the average IPC error is only 3% compared with the entire execution. Moreover, the detailed tuning of SimPoint is also discussed in [55].

Van Biesbrouck *et al.* [56] further extended the SimPoint methodology to estimate the performance of simultaneous multithreading machines. The key idea is using the co-phase matrix to represent the combinations of execution phases from different programs that are running simultaneously, and each entry in the matrix is the combination of the phase-IDs, called as co-phase identifier. Then, the co-phase matrix is used to determine fast-forwarding of threads between samples, and estimate the performance of detailed simulation as well. On the evaluated program pairs, this approach results in an error rate of 4% on average, at the costs of only 1% of the full simulation.

The profiled information of SimPoint is collected via simulation techniques, which may not be able to reproduce the complex execution environment required by some real applications. Besides, it is often hard to port the latest version of complicated applications to simulators. To address such problems, a toolkit called PinPoint is created by Patil *et al.* [57]. PinPoint is built upon SimPoint methodology and a dynamic instrumentation tool, Pin [25], to automatically find the representative execution intervals at run time on commodity computers.

Huang *et al.* [58] proposed to employ the sampling technique for reducing the simulation cost of GPGPU architectures. The proposed sampling technique includes interlaunch sampling and intralaunch sampling, to select representative kernel launches and instructions within one kernel launch, respectively. The interlaunch sampling is achieved by selecting a kernel launch from a group of kernels having similar characteristics. The intralaunch sampling is achieved by selecting the thread blocks with approximate performance. On the evaluated kernels, the sample size of TBPoint is 2.6% with a sampling error as 0.47%.

3) Statistical Sampling: In contrast to SimPoint that uses clustering analysis to find representative traces, there are also several studies employing statistical sampling theory to reduce simulation traces. Conte *et al.* [59] proposed the state-reduction method to statistically sample the simulation traces, which is, to the best of our knowledge, the first piece of work that applies statistical sampling to processor simulation. The accuracy of the approach is guaranteed by reducing the sampling bias (e.g., standard error) and nonsampling bias (e.g., branch predictor state).

Sampling microarchitecture simulation (SMARTS) selects a small subset of instruction traces from the original instruction trace with a specified confidence interval via statistical sampling theory [32], [60]. In SMARTS, a sampling unit is defined as U consecutive instructions in the program's instruction trace. When the total number of instructions is L, the number of total sampling units N equals L/U. According to statistical sampling theory, the required confidence on the estimation of cycle per instruction (CPI) of the entire program execution can be determined by the number of samples (recall n in Table I) from N sampling units, and number of samples n can be dynamically adjusted according to the coefficient of variation of CPI ($\widehat{V}_{\text{CPI}}$). In SMARTS, suppose sampling begins at offset j, that is, detailed simulation is performed at unit j and lasts for U instructions. Then, SMARTS fastforwards $U(k-1) - W$ instructions with only functional simulation, and subsequently W instructions for warming-up are executed via detailed simulation. Experimental results show that for typical applications, a sample size of $n = 10\,000$ units with unit size $U = 1000$ can achieve 99.7% confidence of $\pm 3\%$ error.

In SMARTS, to produce unbiased estimates, the functional warming period is the dominant part of entire simulation. To eliminate the functional warming bottleneck, Wenisch *et al.* [61] further proposed TurboSMARTS, which chooses a minimal subset of warmed states and stores them in checkpoints. Therefore, subsequent experiments can directly load the states from such checkpoints to improve the simulation efficiency. TurboSMARTS maintains the same accuracy of existing simulation sampling approaches while achieves over 250$\times$ speedup.

4) Multiprocessor Sampling: SimFlex [62], [63], which is a fast and accurate full-system simulator built upon Simics simulation environment, applies SMARTS methodology to rapidly choose a representative sample of each workload. A key innovation of SimFlex is that it applies statistical sampling techniques to multiprocessor programs. A multiprocessor program execution consists of multiple instruction streams with nondeterministic communications among them, which makes it hard to find the optimal sample of the full program. SimFlex tackles this problem by focusing on the critical path of multiprocessor execution. In more detail, in order to estimate the execution time of the full program, it is unnecessary to consider none-critical-paths since they do not contribute to the overall execution time. By only considering the program sections on the critical path, the sampling process can be treated as a (uniprocessor-like) interleaving of executions across different processors.

In traditional instruction-based sampling such as SimPoint and SMARTS, the original execution is sampled based on a fixed length of instructions, leading to the divergence of execution progress among threads. In this case, the overlap of different threads may not be the representation of the actual behavior of a multithread application. On contrary, the time-based sampling (TBS) samples the original execution based on a fixed number of cycles, preserving the progressed time of the original execution [64]. Therefore, TBS is able to sample the simulation of multicore processors with no limitation in terms of application type (e.g., multithreaded, multiprogrammed, or both) and architecture heterogeneity. TBS is further implemented as an open source simulator, called enhanced SESC or ESESC. Similarly, Carlson *et al.* [65] proposed to track simulated time rather than instruction count for multithreaded applications. Besides, they also consider the application synchronization events during the fast-forwarding to improve the prediction of application execution. Recently, Carlson *et al.* [66] further proposed BarrierPoint to accelerate sampling simulation by using globally synchronizing barriers in multithreaded applications. The key idea of BarrierPoint is that it only simulates a selected number of representative interbarrier regions, from which the total application execution time can be predicted. BarrierPoint automatically identifies most representative regions by conducting clustering analysis on the microarchitectural independent characteristics of all regions. Compared with prior TBS techniques, BarrierPoint is more efficient since it eliminates full-application functional simulation and it can be simulated in parallel.

IV. Reducing Total Simulation Runs

In the design phase of processors, the total number of simulation runs can be determined by the number of evaluated programs and the number candidate architectures. Thus, the reduction of total simulation runs can be considered from these two aspects.

A. Reducing Evaluated Programs

In the recent years, the number of computer applications increases significantly, which results in a large number of benchmarks to evaluate during the design phase of computer systems. For example, the designers of general-purpose processors often need to consider SPEC CPU benchmark suits (e.g., SPEC CPU2000/2006), MiBench, and PARSEC in performance evaluation. It is quite time-consuming to simulate all these programs due to extremely slow simulation speed. The situation is further exacerbated with the ever-increasing scale of the input sets of modern programs. Therefore, many investigations try to reduce the number of evaluated programs during architectural design based on the observation that there always exists statistical redundancy among benchmarks.

Eeckhout *et al.* [67] proposed to use principal components analysis (PCA) as a statistically rigorous way to select a representative subset from a benchmark suite. In their paper, several program characteristics are proposed to characterize each benchmark. These characteristics include instruction mix, branch prediction accuracy, *D*-/*I*-cache miss rate, sequential flow breaks, and ILP. Such characteristics are extracted via profiling tools at first. Then, PCA is employed to obtain the uncorrelated principal components, which can reduce the dimensionality of the data and generate better clustering results. Finally, different benchmarks are clustered based on the Euclidean distances. Through choosing one benchmark from each cluster, a representative subset of the benchmark suit can be constructed.

Eeckhout *et al.* [68] further proposed to use several microarchitecture-independent characteristics to find a set of representative program phases for simulation. Such microarchitecture-independent program characteristics are also used to measure the similarity among benchmarks, and then clustering analysis is utilized to group similar benchmarks together [69]. Following the basic idea of benchmark subsetting, only representative programs in each group are used to subset the original benchmark suite for efficient simulation. For the SPEC CPU2000 benchmark suite, by using eight representative programs to subset the entire benchmark suite, the error in average IPC is less than 5%, and by using five representative programs to subset all the 32 MiBench and MediaBench benchmarks, the error in average IPC is less than 3.9%. Actually, the proposed microarchitecture-independent characteristics can be used not only for finding representative program subset but also for performance prediction [70], performance optimization [71], and compiler optimization [72].

Yi *et al.* [73] proposed to use the Plackett and Burman (P&B) design to determine the critical parameters of the performance. The effects of parameters are determined by the magnitudes that are computed by using the P&B matrix and the simulated results (e.g., execution time). Then, the similarity among benchmarks is measured by the rank of effects of all parameters. Finally, such similarity is utilized to determine the representative benchmarks for simulation.

Phansalkar *et al.* [74], [75] analyzed the redundancy and application balance in the SPEC CPU2006 benchmark suite. They applied multivariate statistical analysis techniques such as PCA and hierarchical clustering analysis (HCA) to identify the similarity among SPEC CPU2006 programs. According to their evaluation, using four and six benchmarks to subset the entire CINT2006 benchmarks lead to 5.8% and 3.8% average performance errors, respectively. Also, using six and eight benchmarks to subset the entire CFP2006 benchmarks achieve 10.8% and 7% average performance errors, respectively.

Jin and Cheng [76] also proposed a benchmark subsetting methodology and conducted a case study using a bioinformatic benchmark suite called ImplantBench. Their methodology contains two steps. First, factor analysis is employed to reduce the dimensionality of benchmark characteristics and HCA is utilized to demonstrate the inherent correlation and similarities among programs. In the second step, a distance-based program selection strategy is proposed to select the subset of benchmarks given desired workspace variance coverage. Moreover, Jin and Cheng [77], [78] further proposed a general benchmark subsetting framework based on evolutionary algorithms, where a subset of given benchmark suite can be generated based on the microarchitecture-independent characteristics, desired workload space coverage, and total execution time.

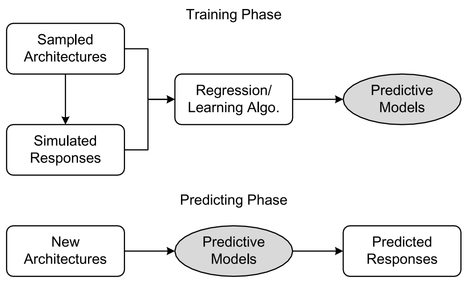


Fig. 4. Reducing simulated architectures via predictive modeling techniques.

B. Reducing Simulated Architectures

1) Predictive Modeling: In the design phase of processors, there always exists a large number of candidate architectures for evaluation. To reduce the candidate architectures, Yi *et al.* [73] proposed to use the P&B design to determine the critical parameters of the performance. To further reduce the simulated architectures, predictive modeling techniques have been proposed recently. The basic idea of predictive modeling technique is to simulate a subset of architectures to build predictive models that can rapidly estimate the performance/power of all candidate architectures.

The basic flow of applying predictive models for DSE can be divided into two phases: 1) the training phase and 2) the predicting phase as illustrated in Fig. 4. In the training phase, a small fraction of architectures in the entire design space are sampled for simulation to obtain the simulated processor responses (e.g., performance, power, and area). Then, the simulated architectures will be treated as the training data to build predictive models via various learning/regression algorithms. In the predicting phase, such predictive models can be utilized to predict the responses of any architectures that are not involved in the training phase without additional simulation runs. Since the responses of all candidate architectures can be estimated by the predictive models, it is relatively trivial to find the optimal architecture via directly comparing the predicted results of all architectures.

Joseph *et al.* [79] first proposed to use linear regression models to characterize the relationship between design parameters and processor performance. Since linear regression models can directly quantify different impacts of design parameters on performance, it can provide more insights and opportunities for performance optimization. To validate this approach, they conduct experiments on a design space containing 26 different variable design parameters. According to the built linear model, it can be concluded that the depth of pipeline, reorder buffer, and the size of issue queue are three most critical design parameters on the processor performance for the evaluated design space.

Although linear models are shown to be capable to provide accurate estimation of the significance of design parameters and their combination, they cannot accurately capture the nonlinear relationship between design parameters and processor responses. Thus, a widely used nonlinear model, radial basis function (RBF) networks, is used to construct accurate predictive models at low simulation costs [80]. According to experiments on SPEC CPU2000 benchmark suite, such nonlinear models only result in 2.8% average error in CPI across the design space. Lee and Brooks [81] also observed that there exists nonlinear relationship between architectural parameters and performance. They proposed to use spline functions for building predictive models. More specifically, restricted cubic splines are utilized to capture the complex, highly curved relationships between parameters and performance/power. They conducted experiments on a design space that contains nearly 1 billion architectures, and by sampling 4000 architectures for training model, the mean prediction errors for performance and power are 4.1% and 4.3%, respectively. In fact, spline function is also used for Pareto frontier analysis, pipeline depth analysis, and processor heterogeneity analysis [82], [83]. Recently, spline function is also used by Wu and Lee [84] to construct predictive models to explore the hardware–software co-spaces.

Lee *et al.* [85] further proposed the composable performance regression (CPR) to efficiently build scalable models for multiprocessors. This paper focuses on efficiently predicting the performance of multiworkloads running on multiprocessors, while previous predictive modeling techniques only estimate the performance of a single (either single-threaded or multithreaded) workload. The CPR model consists of uniprocessor, contention, and penalty models to produce the performance estimation of multiprocessors at the costs of a small number of simulations. In more detail, given a set of benchmarks $B = B_1, \ldots, B_n$ running on a n-core multiprocessor, at first CPR iteratively estimates the performance of each benchmark in B. For one specific benchmark B_i, CPR framework also predicts its contention indicator when it contends with other benchmarks in the set B for shared resources (such as LLC, memory controller). Finally, a penalty model is used to combine the baseline performance of B_i and the contention indicator together to obtain the estimated performance of B_i when running with other benchmarks in the set B. Actually, the uniprocessor model and contention models are also trained by cubic spline function on the full parameter space and shared resource space, respectively.

Artificial neural networks (ANNs) are considered as one of the most powerful and popular learning algorithms in real applications. İpek *et al.* [86], [87] proposed to utilize ANNs to capture the relationship between architectural parameters and performance/power. Besides, to reduce the number of sampling architectures to build a model meeting specific accuracy constraints, intelligent sampling is used to achieve an efficient training process. The experiments on the design spaces of a memory subsystem and a CMP system validate that ANN-based regression models generally predict IPC with 1%–2% error, while reducing the required simulation runs by two orders of magnitude compared with the full simulation.

Cook and Skadron [89] advocated to use genetic programming (GP) to create polynomial functions (or response surface) to characterize the relationship between the performance and architectural parameters. The original GP is used to optimize a population of programs based on evolutionary biology. A distinguish feature of this approach is that the predictive function is modeled as a expression tree, where each node represents a user-defined operator (e.g., simple arithmetic operators, a square operator, and a logarithmic operator) or a design parameter. Then GP imposes evolutionary process (i.e., selection, crossover, and mutation) to this expression tree to obtain the predictive functions. Thus, the predictive functions are built automatically and explicitly. They conducted experiments on the data sets investigated in [81] and [86], and experimental results demonstrate that this approach can obtain highly accurate predictive functions at the cost of a few detailed simulation runs.

In contrast to previous predictive modeling techniques that are only evaluated by the prediction accuracy on different architectures, another kind of work focuses on improving the quality of predicted Pareto set that represents the architectures with best performance/energy tradeoffs. Palermo *et al.* [90] investigated four regression models, i.e., linear regression, shepard-based interpolation, ANNs, and RBFs, to predict the performance and energy. The Pareto sets found by these predictive models are compared with the actual Pareto set with the metric average distance from reference set.[3] Experimental results demonstrate that ANNs achieve the best performance among investigated four models in their proposed DSE framework.

Actually, the afore-mentioned approaches only consider supervised learning techniques to model the relationship between design parameters and processor responses, that is, only simulated architectures are utilized for model construction. Inspired by recent advances on semisupervised learning,[4] Guo *et al.* [91] proposed the co-training model tree (COMT) approach to exploit unlabeled architectures to improve the accuracy of predictive models. To be specific, COMT works in a co-training style, where two learning models label unlabeled architectures for each other. According to their experiments, COMT outperforms ANN-based model given the same number of simulated architectures via random sampling. Moreover, Chen *et al.* [92] proposed the COAL approach that combines semisupervised learning and active learning[5] together to further improve the performance of predictive models.

In addition to the above regression-based approaches, recently Chen *et al.* [93] proposed a ranking-based approach for DSE. The key observation is that architects mostly need the relative ranking of two architectures, rather than accurately estimating the performance of each architecture. Thus, by formulating the DSE as a ranking problem, ArchRanker is proposed to train ranking models to predict the relative ranking of architectures. Experimental results show that ArchRanker can not only more accurately predict the relative ordering of two architectures, but also require much fewer simulation runs to obtain the same accuracy compared with ANN-based regression models. To facilitate the design of heterogeneous systems, Mariani *et al.* [94] proposed the DRuiD framework to rank different heterogeneous architectures for a target functionality. Specifically, DRuiD uses machine learning approaches (i.e., random forests and genetic algorithms) to determine the most suitable computational element (e.g., the hardware accelerator implemented on the FPGA) to be used for a certain application kernel.

With the developing of machine learning techniques, it is expected that more opportunities could be provided to further reduce the number of simulated architectures for training accurate predictive models.

2) Heuristic Searching: In contrast to predictive modeling techniques that explicitly construct approximate functions between design parameters and processor responses, another methodology uses heuristic searching algorithms (e.g., evolutionary algorithms) to directly find the most promising architectures under design constraints. In other words, such methodology treats DSE as a multiobjective optimization (MOO) problem, and by solving this problem, the number of indispensable simulated architectures can be significantly reduced.

In order to identify the promising architectures with best performance/energy tradeoffs for a superscalar architecture, Palermo *et al.* [95] formed this problem as an MOO, and proposed a DSE framework based on heuristics algorithms. Moreover, the authors compared the effectiveness of three heuristic algorithms: 1) random search; 2) simulated annealing; and 3) tabu search, to reduce the overall simulation runs. Experimental results show that the proposed framework can reduce the number of simulation runs up to three orders of magnitude compared to an exhaustive search strategy.

Ascia *et al.* [96] combined multiobjective evolutionary algorithms and fuzzy systems together to reduce the simulation costs of an SoC platform. Multiobjective evolutionary algorithms are used as exploration heuristic to approximate the Pareto set, and fuzzy systems are used to accelerate the evaluation of each configuration via predicting the performance rather than simulation. They conducted experiments on a highly parameterized SoC platform, and experimental results demonstrate that this approach can improve the quality of Pareto set and reduce simulation costs for a given set of multimedia applications.

Recently, Mariani *et al.* [97] proposed an iterative DSE approach to derive the most promising architectures via correlation-based models. In each iteration, only the architecture with the maximal expected improvement over present Pareto set is considered for simulation. This architecture with maximal expected improvement is selected by using a single objective genetic algorithm. Such an evolutionary process continues until the Pareto set is unchanged or a maximum number of simulation runs has been reached. Comparing with several mature MOO algorithms (including the notable NSGA-II [98]), this approach can speed up the overall exploration phase up to 65%.

Mariani *et al.* [99] proposed the DeSpErate++ framework to exploit predictive models to improve the DSE efficiency on a parallel computing platform. More specifically, an estimation of distribution algorithm is applied to heuristically identify the distribution of optimal design configurations in the design space. Orthogonally, a predictive model such as ANN is exploited to predict the simulation time of architectural configurations to organize their schedule over a parallel computing system.

Actually, in [90], [96], [97], and [99], predictive modeling (such as those described in Section IV-B1) is exploited in orthogonality to heuristic optimization approaches (such as the ones described in [95] and [98]) to achieve high DSE efficiency.

3) Analytical Modeling: In contrast to afore-mentioned empirical models such as ANNs that treat the processor as a "black box," mechanistic analytical models are derived from the internal mechanisms of processors to estimate the performance without detailed full simulation. Karkhanis and Smith [100] proposed a first-order model to predict the performance of superscalar processors. The basic idea is to first count the miss events (e.g., branch misprediction

[3]It measures the distance to the actual Pareto set of entire design space.

[4]As stated in Section II-C3, semisupervised learning techniques leverage unlabeled training examples to enhance the prediction accuracy given limited labeled training examples.

[5]In active learning, selected unlabeled architectures are actively sent for simulation to improve the modeling accuracy.

and cache misses) through relatively simple trace-driven simulation, and then the performance penalties caused by such miss events are added to the ideal CPI.

Eyerman *et al.* [101] proposed interval analysis to first divide the program execution into discrete intervals by miss events, then determine the performance of each interval based on the corresponding miss events, and finally aggregate the performance of all intervals to obtain the overall performance. In contrast to prior studies that model the performance based on the issue rate, Eyerman *et al.* [102] further presented a simplified interval model that is built upon the dispatch width.

As in-order processors gain more attentions due to the energy concern, Breughe *et al.* [103] built an analytical model for scalar in-order processors by considering both miss events and hazards due to dependencies. After that, Breughe *et al.* [104] further built a mechanistic analytical model for superscalar in-order processors. The model also takes the profiled program characteristics (e.g., instruction mix, cache miss rates, and branch misprediction rates) as the input, while it is greatly enhanced with modeling functional unit contention and interinstruction dependences.

The above work requires profiling to obtain architecture-dependent characteristics such as cache misses, and the corresponding profiling cost cannot be neglected for a large design space. To reduce the profiling cost, Van den Steen *et al.* [105] constructed an analytical model based on architecture independent (while program dependent) profiles, including instruction mix and dependences, memory behavior, and branch behavior. Such profiles are sent to the analytical model, together with the architecture parameters such as pipeline depth, to obtain both the performance and power estimation.

Processor queueing model [106] is another example of analytical model. Although the queuing model also relies on processor parameters (e.g., memory latency) and program characteristics (e.g., loads per instruction), it mainly focuses on the interactions between the pipeline, buffers, and caches in the memory hierarchy. The queuing model is validated against cycle-accurate simulators, and it can be easily integrated into a multiprocessor system model.

C. Reducing Both Programs and Architectures

The aforementioned approaches manage to cut down the total number of simulation runs by reducing either the number of evaluated programs or the number of simulated architectures. To further reduce the number of simulation runs, Dubach *et al.* [107] and Khan *et al.* [108] independently proposed signature-based approaches. The key of these approaches is incorporating several simulated responses on a small number (e.g., 8 or 32) of typical architectures, named as signatures, for each program during the training process. Then, when encountering a new program, only a small number of simulation runs are required to obtain its signature. With the help of such signature, the responses of the new program can be predicted without additional simulation runs. Although the basic ideas of these two work are similar, Dubach *et al.* [107] used linear models to combine several models trained from different programs, while Khan *et al.* [108] directly treated the signatures as the inputs of the learning algorithm.

Fig. 5 compares the number of simulation runs of the signature-based approaches and traditional predictive modeling approaches. Conventionally, the total number of simulation runs is $m \times N$, where m is the number of simulated

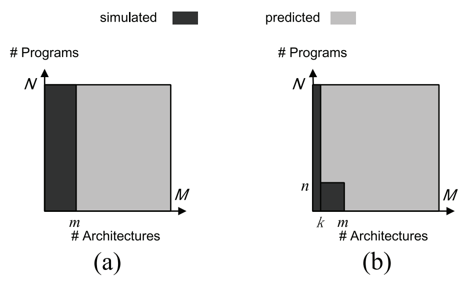


Fig. 5. Comparison of (a) traditional predictive modeling techniques and (b) signature-based approaches.

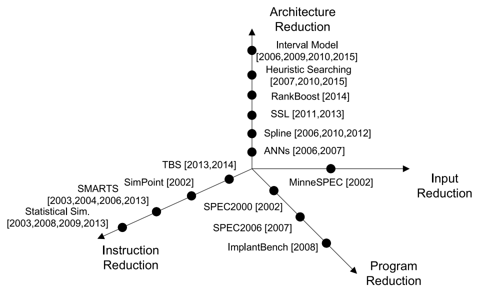


Fig. 6. Comparison of different fast simulation methodology via statistical techniques.

architectures and the N is the number of evaluated programs. In comparison, the total number of simulation runs of the signature-based approaches is $m \times n + (N - n) \times k$, where m is the number of simulated architecture, n is the number of trained programs, N is the number of evaluated programs, and k is the size of signature (e.g., 8). Thus, it can be observed that the total number of simulation runs could be significantly reduced, especially when more programs should be evaluated.

In addition to exploring the program and architecture co-design space, researchers also manage to explore the joint space consisting of not only architectural options but also various compiler and hardware circuit options. Dubach *et al.* [109] utilized predictive modeling (i.e., support vector machines) to explore the joint space of architecture and compiler options. Azizi *et al.* [110] proposed to use spline functions to explore the joint space of architecture and circuits. The joint space of program and processor architecture is also explored by spline functions in [84].

V. Summary and Comparison

Fig. 6 presents the above partial simulation approaches via statistical techniques from another perspective. It is notable that all those approaches can be classified in four orthogonal dimensions. In the first dimension, the approaches improve simulation efficiency via reducing the input size, where the most representative approach is MinneSPEC [40]. In the second dimension, the approaches speed-up simulation by reducing the number of simulated dynamic instructions of each program. Statistical simulation, sampling simulation, and statistical sampling are three main paradigms in this category. The approaches in the third dimension manage to reduce the number of programs for evaluation, where clustering analysis technique is most widely used to select a small but representative subset from the original benchmark suite. Finally, in the fourth dimension, the approaches reduce the number of simulated architectures via predictive modeling

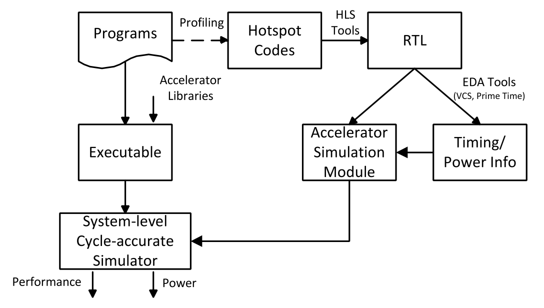


Fig. 7. Design and evaluation flow of accelerator-centric architecture.

techniques. To construct such predictive models, various statistical regression techniques, including linear regression, spline function, and machine learning techniques, have been extensively studied by researchers. Moreover, the number of simulated architectures can also be reduced by heuristic searching and analytical modeling.

It is possible to combine these afore-mentioned approaches from different dimensions to achieve a more efficient fast simulation methodology. For example, given a DSE task for N target programs, the total number of simulated instructions should be $M \times N \times P$, where M is the number of architectures in the entire design space and P is the average number of dynamic instructions of each program. To accelerate the architectural simulation, in the first step, architects can enforce benchmark subsetting techniques on all target programs, and only reserve $n(n < N)$ programs for evaluation. Then, statistical simulation or statistical sampling techniques can be used to reduce the number of simulated instructions by several orders of magnitude [i.e., the average number of instructions is only $p(p << P)$]. Finally, only a small proportion of all possible architectures [e.g., $m(m << M)$] need to simulate for constructing predictive models. With the help of such predictive models, it is no necessary to conduct simulations on remaining architectures. Therefore, the total number of simulated instructions can be reduced from $M \times N \times P$ to only $m \times n \times p$, which can improve the simulation efficiency by several orders of magnitude.

VI. Implications for Future Accelerator-Centric Architectures

As the transistor densities continue to scale exponentially, a chip cannot be fully powered at one time given limited chip-level power budget, which results in the dark silicon problem. To address this problem, customized and specialized hardware accelerators would be very promising and have been investigated by many researchers. Actually, heterogeneous chips containing application-specific accelerators are becoming increasingly common in mobile system, server, and desktop systems [111], [112]. Fig. 7 shows a typical design and evaluation flow of accelerator-centric architectures [113], [114]. The first step is profiling the target program to get its hotspot codes (e.g., the most time-consuming functions). Then, several HLS tools (e.g., AutoPilot [115]) are used to directly convert the C code to synthesizeable Verilog. After that, the Verilog designs of the accelerators are simulated by register transfer level (RTL) simulation tools to get their timing and power characteristics. With such timing and power information, cycle accurate simulation modules of the accelerators can be generated and plugged into the system-level cycle-accurate simulator such as Simics [19]. Once the system-level simulator is built, the executable binary of the target program, which is compiled with the accelerator libraries, is evaluated on such simulator to generate the overall performance and power estimation.

According to the design flow as shown in Fig. 7, there are several challenges to efficiently design accelerator-centric architectures. The first one is how to efficiently generate an accelerator given a target program. In addition to using tools, such as AutoPilot and Spiral [116], to automatically generate the Verilog designs of accelerators, many researchers also designed the application-specific accelerators manually, such as thin servers with smart pipes [117] for memcached and Q100 [118] for database applications. The second challenge is to determine which programs should be accelerated, since there always exists a tradeoff between generality and efficiency of processors. The third challenge is how to accurately and efficiently evaluate different design options of accelerators, so as to find optimal designs under specific performance/power/area constraints. Here, we discuss the implication of reviewed partial simulation approaches on addressing the later two challenges.

Although specialized architectures gain significant efficiency at the cost of generality, it is still expected that more applications can benefit from such architectures given limited area and power budget. This problem can be formulated as to find a representative subset of many widely used applications, which has been extensively investigated by the benchmark subsetting techniques. Inspired by the benchmark subsetting techniques, we may use several statistical techniques (e.g., PCA and clustering analysis) to guide the selection of target applications for hardware acceleration. A typical work that uses this methodology to design accelerators is the 10×10 project, which exploits workload analysis to drive the design of heterogeneous architecture containing customized accelerators [119], [120]. The project is built upon detailed analysis of program characteristics, such as operations, data types, and control flows. By conducting clustering analysis on such extracted characteristics from various benchmark suites, several clusters are carefully selected and supported by customized microengines.

Similar to the design of general-purpose processors, there also exists many tunable parameters for the design of accelerators. For example, during the design of DRAM-aware fast Fourier transform (FFT) accelerator, there are many different parameters such as FFT radix and streaming width [121]. Conventionally, RTL simulation is indispensable to evaluate different design options. However, the simulation costs are intractable due a large number of design options for evaluation. To address this issue, Shao *et al.* [122] proposed a pre RTL approach to efficiently evaluate the performance and power of accelerator designs. In addition to such high-level models for fast DSE, we also believe that several partial simulation approaches, such as predictive modeling, can also be deployed to further cut down the RTL simulation costs. As shown in Fig. 4, only a sampled number of accelerator configurations are analyzed by the RTL-based synthesis flow to generate corresponding performance and power estimation. Then, predictive models can be constructed with various statistical techniques. Finally, the performance/power of all other accelerator configurations can be directly predicted by such models without any costly simulation runs.

VII. Conclusion

In this paper, we review recent advances on partial simulation techniques. The basic idea of these techniques is to simulate a small subset of the representative instructions from the complete set of simulated instructions by using various statistical techniques, such as basic statistical concepts (sampling theory and non-parametric hierarchical performance testing [34]), statistical analysis techniques (PCA and clustering analysis), regression techniques (linear regression and spline function), and machine learning techniques (RBF, ANNs, and evolutionary algorithms). Technically, these techniques tradeoff accuracy for speed, and they can be categorized into four orthogonal dimensions, that is, input size reduction, dynamic instruction reduction, program reduction, and architecture reduction. Also, these approaches can be combined together to obtain a more efficient simulation methodology. We believe that partial simulation methodology can also play an important role during the design of future accelerator-centric architectures.

Acknowledgment

The content, views, and conclusions presented in this paper do not necessarily reflect the position or the policy of Defense Advanced Research Projects Agency or the U.S. Government, and no official endorsement should be inferred.

References

[1] J. J. Yi *et al.*, "The future of simulation: A field of dreams," *Computer*, vol. 39, no. 11, pp. 22–29, Nov. 2006.

[2] P. Bohrer *et al.*, "Mambo: A full system simulator for the PowerPC architecture," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 4, pp. 8–12, Mar. 2004.

[3] *PowerPC 405GP Embedded Processor Users Manual*, IBM Corp., Research Triangle Park, NC, USA, 2000.

[4] R. Bedicheck, "SimNow: Fast platform simulation purely in software," in *Proc. Hot Chips*, Aug. 2004. [Online]. Available: http://www.hotchips.org/wp-content/uploads/hc_archives/hc16/2_Mon/15_HC16_Sess4_Pres1_bw.pdf

[5] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer, "HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing," in *Proc. HPCA*, San Antonio, TX, USA, 2011, pp. 406–417.

[6] J. E. Miller *et al.*, "Graphite: A distributed parallel simulator for multicores," in *Proc. HPCA*, Bengaluru, India, 2010, pp. 1–12.

[7] D. Genbrugge, S. Eyerman, and L. Eeckhout, "Interval simulation: Raising the level of abstraction in architectural simulation," in *Proc. HPCA*, Bengaluru, India, 2010, pp. 1–12.

[8] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. SC*, Seatle, WA, USA, 2011, pp. 1–12.

[9] K. Wang, Y. Zhang, H. Wang, and X. Shen, "Parallelization of IBM mambo system simulator in functional modes," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 1, pp. 71–76, Jan. 2008.

[10] J. Chen, M. Annavaram, and M. Dubois, "SlackSim: A platform for parallel simulations of CMPs on CMPs," *ACM SIGARCH Comput. Archit. News*, vol. 37, no. 2, pp. 20–29, May 2009.

[11] D. Chiou *et al.*, "FPGA-accelerated simulation technologies (FAST): Fast, full-system, cycle-accurate simulators," in *Proc. MICRO*, Chicago, IL, USA, 2007, pp. 249–261.

[12] E. S. Chung *et al.*, "ProtoFlex: Towards scalable, full-system multiprocessor simulations using FPGAs," *ACM Trans. Reconfig. Tech. Syst.*, vol. 2, no. 2, Jun. 2009, Art. ID 15.

[13] Z. Tan *et al.*, "RAMP gold: An FPGA-based architecture simulator for multiprocessors," in *Proc. DAC*, Anaheim, CA, USA, 2010, pp. 463–468.

[14] S. Takamaeda-Yamazaki, R. Sasakawa, Y. Sakaguchi, and K. Kise, "An FPGA-based scalable simulation accelerator for tile architectures," *SIGARCH Comput. Archit. News*, vol. 39, no. 4, pp. 38–43, Sep. 2011.

[15] M. Wong, "C++ benchmarks in SPEC CPU2006," *SIGARCH Comput. Archit. News CAN*, vol. 35, no. 1, pp. 77–83, Mar. 2007.

[16] A. Rico *et al.*, "Trace-driven simulation of multithreaded applications," in *Proc. ISPASS*, Austin, TX, USA, 2011, pp. 87–96.

[17] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," Dept. Comput. Sci., Univ. Wisconsin–Madison, Madison, WI, USA, Tech. Rep. 1342, Jun. 1997.

[18] C. J. Hughes, V. S. Pai, P. Ranganathan, and S. V. Adve, "Rsim: Simulating shared-memory multiprocessors with ILP processors," *Computer*, vol. 35, no. 2, pp. 40–49, Feb. 2002.

[19] P. S. Magnusson *et al.*, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[20] M. M. K. Martin *et al.*, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, Nov. 2005.

[21] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, May 2011.

[22] N. L. Binkert *et al.*, "The M5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, Jul./Aug. 2006.

[23] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: A full system simulator for multicore x86 CPUs," in *Proc. DAC*, New York, NY, USA, 2011, pp. 1050–1055.

[24] M. T. Yourst, "PTLsim: A cycle accurate full system x86-64 microarchitectural simulator," in *Proc. ISPASS*, San Jose, CA, USA, 2007, pp. 23–34.

[25] C.-K. Luk *et al.*, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proc. PLDI*, Chicago, IL, USA, 2005, pp. 190–200.

[26] A. Jaleel, R. S. Cohn, C.-K. Luk, and B. Jacob, "CMP$im: A pin-based on-the-fly multi-core cache simulator," in *Proc. MOBS*, Beijing, China, Jun. 2008. [Online]. Available: http://eng.umd.edu/∼blj/papers/mobs2008.pdf

[27] D. Sanchez and C. Kozyrakis, "ZSim: Fast and accurate microarchitectural simulation of thousand-core systems," in *Proc. ISCA*, Tel Aviv, Israel, 2013, pp. 475–486.

[28] N. Chitlur *et al.*, "QuickIA: Exploring heterogeneous architectures on real prototypes," in *Proc. HPCA*, New Orleans, LA, USA, 2012, pp. 1–8.

[29] J. Chan, G. Hendry, A. Biberman, K. Bergman, and L. P. Carloni, "PhoenixSim: A simulator for physical-layer analysis of chip-scale photonic interconnection networks," in *Proc. DATE*, Dresden, Germany, 2010, pp. 691–696.

[30] P. S. Magnusson and J. Montelius, "Performance debugging and tuning using an instruction-set simulator," Swedish Inst. Comput. Sci., Kista, Sweden, Tech. Rep. T97:02, 1997.

[31] L. Albertsson and P. S. Magnusson, "Using complete system simulation for temporal debugging of general purpose operating systems and workload," in *Proc. MASCOTS*, San Francisco, CA, USA, 2000, pp. 191–198.

[32] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling," in *Proc. ISCA*, San Diego, CA, USA, 2003, pp. 84–95.

[33] A. R. Alameldeen and D. A. Wood, "Variability in architectural simulations of multi-threaded workloads," in *Proc. HPCA*, Anaheim, CA, USA, 2003, pp. 7–18.

[34] T. Chen *et al.*, "Statistical performance comparisons of computers," in *Proc. HPCA*, New Orleans, LA, USA, 2012, pp. 1–12.

[35] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.

[36] R. Bitirgen, E. Ipek, and J. F. Martinez, "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach," in *Proc. MICRO*, Lake Como, Italy, 2008, pp. 318–329.

[37] J. Li *et al.*, "Machine learning based online performance prediction for runtime parallelization and task scheduling," in *Proc. ISPASS*, Boston, MA, USA, 2009, pp. 89–100.

[38] C. Dubach, T. M. Jones, E. V. Bonilla, and M. F. P. O'Boyle, "A predictive model for dynamic microarchitectural adaptivity control," in *Proc. MICRO*, Atlanta, GA, USA, 2010, pp. 485–496.

[39] E. Ipek, O. Mutlu, J. F. Martinez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *Proc. ISCA*, Beijing, China, 2008, pp. 39–50.

[40] A. J. KleinOsowski and D. J. Lilja, "MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research," *Comput. Arch. Lett.*, vol. 1, no. 1, pp. 1–4, Jan./Dec. 2002.

[41] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Designing computer architecture research workloads," *Computer*, vol. 36, no. 2, pp. 65–71, Feb. 2003.

[42] W. C. Hsu, H. Chen, P. C. Yew, and H. Chen, "On the predictability of program behavior using different input data sets," in *Proc. INTERACT*, Cambridge, MA, USA, 2002, pp. 45–53.

[43] S. Nussbaum and J. E. Smith, "Modeling superscalar processors via statistical simulation," in *Proc. PACT*, Barcelona, Spain, 2001, pp. 15–24.

[44] L. Eeckhout, S. Nussbaum, J. E. Smith, and K. De Bosschere, "Statistical simulation: Adding efficiency to the computer designer's toolbox," *IEEE Micro*, vol. 23, no. 5, pp. 26–38, Sep./Oct. 2003.

[45] M. Oskin, F. T. Chong, and M. Farrens, "HLS: Combining statistical and symbolic simulation to guide microprocessor designs," in *Proc. ISCA*, Vancouver, BC, Canada, 2000, pp. 71–82.

[46] L. Eeckhout, R. H. Bell, Jr., B. Stougie, K. De Bosschere, and L. K. John, "Control flow modeling in statistical simulation for accurate and efficient processor design studies," in *Proc. ISCA*, Munchen, Germany, Jun. 2004, pp. 350–361.

[47] K. Skadron, M. Martonosi, and D. W. Clark, "Speculative updates of local and global branch history: A quantitative analysis," *J. Instr. Level Parallel.*, vol. 2, pp. 589–598, Jan. 2000.

[48] D. Genbrugge and L. Eeckhout, "Statistical simulation of chip multiprocessors running multi-program workloads," in *Proc. ICCD*, Lake Tahoe, CA, USA, 2007, pp. 464–471.

[49] D. Genbrugge and L. Eeckhout, "Chip multiprocessor design space exploration through statistical simulation," *IEEE Trans. Comput.*, vol. 58, no. 12, pp. 1668–1681, Dec. 2009.

[50] C. Hughes and T. Li, "Accelerating multi-core processor design space evaluation using automatic multi-threaded workload synthesis," in *Proc. IISWC*, Seattle, WA, USA, 2008, pp. 163–172.

[51] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proc. ISPASS*, Boston, MA, USA, 2009, pp. 163–174.

[52] Z. Yu *et al.*, "Accelerating GPGPU architecture simulation," in *Proc. SIGMETRICS*, Pittsburgh, PA, USA, 2013, pp. 331–332.

[53] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proc. ASPLOS*, San Jose, CA, USA, 2002, pp. 45–57.

[54] T. Sherwood, S. Sair, and B. Calder, "Phase tracking and prediction," in *Proc. ISCA*, San Diego, CA, USA, 2003, pp. 336–349.

[55] G. Hamerly, E. Perelman, J. Lau, B. Calder, and T. Sherwood, "Using machine learning to guide architecture simulation," *J. Mach. Learn. Res.*, vol. 7, pp. 343–378, Feb. 2006.

[56] M. Van Biesbrouck, T. Sherwood, and B. Calder, "A co-phase matrix to guide simultaneous multithreading simulation," in *Proc. ISPASS*, Austin, TX, USA, 2004, pp. 45–56.

[57] H. Patil *et al.*, "Pinpointing representative portions of large Intel Itanium programs with dynamic instrumentation," in *Proc. MICRO*, Portland, OR, USA, 2004, pp. 81–92.

[58] J.-C. Huang, L. Nai, H. Kim, and H.-H. S. Lee, "TBPoint: Reducing simulation time for large-scale GPGPU kernels," in *Proc. IPDPS*, Phoenix, AZ, USA, 2014, pp. 437–446.

[59] T. M. Conte, M. A. Hirsch, and K. N. Menezes, "Reducing state loss for effective trace sampling of superscalar processors," in *Proc. ICCD*, Austin, TX, USA, 1996, pp. 468–477.

[60] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "Statistical sampling of microarchitecture simulation," *ACM Trans. Model. Comput. Simulat.*, vol. 16, no. 3, pp. 197–224, Jul. 2006.

[61] T. F. Wenisch, R. E. Wunderlich, B. Falsafi, and J. C. Hoe, "TurboSMARTS: Accurate microarchitecture simulation sampling in minutes," in *Proc. SIGMETRICS*, Banff, AB, Canada, 2005, pp. 408–409.

[62] N. Hardavellas *et al.*, "SimFlex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 4, pp. 31–35, 2004.

[63] T. F. Wenisch *et al.*, "SimFlex: Statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, Jul./Aug. 2006.

[64] E. K. Ardestani and J. Renau, "ESESC: A fast multicore simulator using time-based sampling," in *Proc. HPCA*, Shenzhen, China, 2013, pp. 448–459.

[65] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sampled simulation of multi-threaded applications," in *Proc. ISPASS*, Austin, TX, USA, 2013, pp. 2–12.

[66] T. E. Carlson, W. Heirman, K. Van Craeynest, and L. Eeckhout, "BarrierPoint: Sampled simulation of multi-threaded applications," in *Proc. ISPASS*, Monterey, CA, USA, 2014, pp. 2–12.

[67] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Workload design: Selecting representative program-input pairs," in *Proc. PACT*, Charlottesville, VA, USA, 2002, pp. 83–94.

[68] L. Eeckhout, J. Sampson, and B. Calder, "Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation," in *Proc. IISWC*, Austin, TX, USA, 2005, pp. 2–12.

[69] A. Joshi, A. Phansalkar, L. Eeckhout, and L. K. John, "Measuring benchmark similarity using inherent program characteristics," *IEEE Trans. Comput.*, vol. 55, no. 6, pp. 769–782, Jun. 2006.

[70] K. Hoste *et al.*, "Performance prediction based on inherent program similarity," in *Proc. PACT*, San Francisco, CA, USA, 2006, pp. 114–122.

[71] D. Shelepov *et al.*, "HASS: A scheduler for heterogeneous multicore systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 66–75, 2009.

[72] Y. Chen *et al.*, "Evaluating iterative optimization across 1000 datasets," in *Proc. PLDI*, Toronto, ON, Canada, 2010, pp. 448–459.

[73] J. J. Yi, D. J. Lilja, and D. M. Hawkins, "A statistically rigorous approach for improving simulation methodology," in *Proc. HPCA*, Anaheim, CA, USA, 2003, pp. 281–291.

[74] A. Phansalkar, A. Joshi, and L. K. John, "Subsetting the SPEC CPU2006 benchmark suite," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 69–76, 2007.

[75] A. Phansalkar, A. Joshi, and L. K. John, "Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite," in *Proc. ISCA*, San Diego, CA, USA, 2007, pp. 412–423.

[76] Z. Jin and A. C. Cheng, "Improve simulation efficiency using statistical benchmark subsetting: An implantBench case study," in *Proc. DAC*, Anaheim, CA, USA, 2008, pp. 970–973.

[77] Z. Jin and A. C. Cheng, "Evolutionary benchmark subsetting," *IEEE Micro*, vol. 28, no. 6, pp. 20–36, Nov./Dec. 2008.

[78] Z. Jin and A. C. Cheng, "SubsetTrio: An evolutionary, geometric, and statistical benchmark subsetting framework," *ACM Trans. Model. Comput. Simulat.*, vol. 21, no. 3, pp. 1–23, 2011.

[79] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "Construction and use of linear regression models for processor performance analysis," in *Proc. HPCA*, Austin, TX, USA, 2006, pp. 99–108.

[80] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "A predictive performance model for superscalar processors," in *Proc. MICRO*, Orlando, FL, USA, 2006, pp. 161–170.

[81] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *Proc. ASPLOS*, San Jose, CA, USA, 2006, pp. 185–194.

[82] B. C. Lee and D. M. Brooks, "Illustrative design space studies with microarchitectural regression models," in *Proc. HPCA*, Scottsdale, AZ, USA, 2007, pp. 340–351.

[83] B. C. Lee and D. Brooks, "Applied inference: Case studies in microarchitectural design," *ACM Trans. Archit. Code Optim.*, vol. 7, no. 2, pp. 1–37, 2010.

[84] W. Wu and B. C. Lee, "Inferred models for dynamic and sparse hardware-software spaces," in *Proc. MICRO*, Vancouver, BC, Canada, 2012, pp. 413–424.

[85] B. C. Lee, J. Collins, H. Wang, and D. Brooks, "CPR: Composable performance regression for scalable multiprocessor models," in *Proc. MICRO*, Lake Como, Italy, 2008, pp. 270–281.

[86] E. İpek, S. A. McKee, R. Caruana, B. R. De Supinski, and M. Schulz, "Efficiently exploring architectural design spaces via predictive modeling," in *Proc. ASPLOS*, San Jose, CA, USA, 2006, pp. 195–206.

[87] E. Ipek *et al.*, "Efficient architectural design space exploration via predictive modeling," *ACM Trans. Archit. Code Optim.*, vol. 4, no. 4, pp. 1–34, 2008.

[88] C.-B. Cho, W. Zhang, and T. Li, "Informed microarchitecture design space exploration using workload dynamics," in *Proc. MICRO*, Chicago, IL, USA, 2007, pp. 274–285.

[89] H. Cook and K. Skadron, "Predictive design space exploration using genetically programmed response surfaces," in *Proc. DAC*, Anaheim, CA, USA, 2008, pp. 960–965.

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

[91] Q. Guo *et al.*, "Effective and efficient microprocessor design space exploration using unlabeled design configurations," in *Proc. IJCAI*, Barcelona, Spain, 2011, pp. 1671–1677.

[92] T. Chen *et al.*, “Effective and efficient microprocessor design space exploration using unlabeled design configurations,” *ACM Trans. Intell. Syst. Technol.*, vol. 5, no. 1, pp. 20:1–20:18, 2013.

[93] T. Chen *et al.*, “ArchRanker: A ranking approach to design space exploration,” in *Proc. ISCA*, Minneapolis, MN, USA, 2014, pp. 85–96.

[94] G. Mariani *et al.*, “DRuiD: Designing reconfigurable architectures with decision-making support,” in *Proc. ASP-DAC*, Singapore, 2014, pp. 213–218.

[95] G. Palermo, C. Silvano, and V. Zaccaria, “Multi-objective design space exploration of embedded systems,” *J. Embedded Comput.*, vol. 1, no. 3, pp. 305–316, 2005.

[96] G. Ascia, V. Catania, A. G. Di Nuovo, M. Palesi, and D. Patti, “Efficient design space exploration for application specific systems-on-a-chip,” *J. Syst. Archit.*, vol. 53, no. 10, pp. 733–750, 2007.

[97] G. Mariani *et al.*, “A correlation-based design space exploration methodology for multi-processor systems-on-chip,” in *Proc. DAC*, Anaheim, CA, USA, 2010, pp. 120–125.

[98] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[99] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano, “DeSpErate++: An enhanced design space exploration framework using predictive simulation scheduling,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 2, pp. 293–306, Feb. 2015.

[100] T. S. Karkhanis and J. E. Smith, “A first-order superscalar processor model,” in *Proc. ISCA*, Munich, Germany, 2004, pp. 338–349.

[101] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, “A performance counter architecture for computing accurate CPI components,” in *Proc. ASPLOS*, San Jose, CA, USA, 2006, pp. 175–184.

[102] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, “A mechanistic performance model for superscalar out-of-order processors,” *ACM Trans. Comput. Syst.*, vol. 27, no. 2, pp. 1–37, 2009.

[103] M. Breughe *et al.*, “How sensitive is processor customization to the workload’s input datasets?” in *Proc. ASAP*, San Diego, CA, USA, 2011, pp. 1–7.

[104] M. B. Breughe, S. Eyerman, and L. Eeckhout, “Mechanistic analytical modeling of superscalar in-order processor performance,” *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 1–26, 2015.

[105] S. Van den Steen *et al.*, “Micro-architecture independent analytical performance and power modeling,” in *Proc. ISPASS*, Philadelphia, PA, USA, 2015, pp. 32–41.

[106] T.-F. Tsuei and W. Yamamoto, “Queuing simulation model for multiprocessor systems,” *Computer*, vol. 36, no. 2, pp. 58–64, Feb. 2003.

[107] C. Dubach, T. M. Jones, and M. F. P. O’Boyle, “Microarchitectural design space exploration using an architecture-centric approach,” in *Proc. MICRO*, Chicago, IL, USA, 2007, pp. 262–271.

[108] S. Khan, P. Xekalakis, J. Cavazos, and M. Cintra, “Using predictivemodeling for cross-program design space exploration in multicore systems,” in *Proc. PACT*, Brasov, Romania, 2007, pp. 327–338.

[109] C. Dubach, T. M. Jones, and M. F. P. O’Boyle, “Exploring and predicting the architecture/optimising compiler co-design space,” in *Proc. CASES*, Atlanta, GA, USA, 2008, pp. 31–40.

[110] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel, and M. Horowitz, “Energy-performance tradeoffs in processor architecture and circuit design: A marginal cost analysis,” in *Proc. ISCA*, Saint-Malo, France, 2010, pp. 26–36.

[111] J. D. Brown, S. Woodward, B. M. Bass, and C. L. Johnson, “IBM power edge of network processor: A wire-speed system on a chip,” *IEEE Micro*, vol. 31, no. 2, pp. 76–85, Mar./Apr. 2011.

[112] R. Golla and P. Jordan, “T4: A highly threaded server-on-a-chip with native support for heterogeneous computing,” in *Proc. Hot Chip Symp.*, 2011. [Online]. Available: http://www.hotchips.org/wp-content/uploads/hc_archives/hc23/HC23.19.7-Server/HC23.19.731-T4-Golla-Oracle-hotchips_corrected.pdf

[113] G. Venkatesh *et al.*, “Conservation cores: Reducing the energy of mature computations,” in *Proc. ASPLOS*, Pittsburgh, PA, USA, 2010, pp. 205–218.

[114] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman, “Architecture support for accelerator-rich CMPs,” in *Proc. DAC*, San Francisco, CA, USA, 2012, pp. 843–849.

[115] J. Cong *et al.*, “High-level synthesis for FPGAs: From prototyping to deployment,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 473–491, Apr. 2011.

[116] M. Puschel *et al.*, “Spiral: Code generation for DSP transforms,” *Proc. IEEE*, vol. 93, no. 2, pp. 232–275, Feb. 2005.

[117] K. Lim, D. Meisner, A. G. Saidi, P. Ranganathan, and T. F. Wenisch, “Thin servers with smart pipes: Designing SoC accelerators for memcached,” in *Proc. ISCA*, Tel Aviv, Israel, 2013, pp. 36–47.

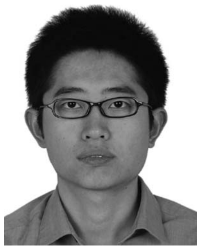
[118] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross, “Q100: The architecture and design of a database processing unit,” in *Proc. ASPLOS*, Salt Lake City, UT, USA, 2014, pp. 255–268.

[119] S. Borkar and A. A. Chien, “The future of microprocessors,” *ACM Commun.*, vol. 54, no. 5, pp. 67–77, 2011.

[120] A. Guha, Y. Zhang, R. Ur Rasool, and A. A. Chien, “Systematic evaluation of workload clustering for extremely energy-efficient architectures,” *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 2, pp. 22–29, 2013.

[121] B. Akin, F. Franchetti, and J. C. Hoe, “Understanding the design space of DRAM-optimized hardware FFT accelerators,” in *Proc. ASAP*, Zurich, Switzerland, 2014, pp. 248–255.

[122] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, “Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures,” in *Proc. ISCA*, Minneapolis, MN, USA, 2014, pp. 97–108.



Qi Guo received the B.S. degree in computer science from the Department of Computer Science and Technology, Tongji University, Shanghai, China, in 2007, and the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2012.

He is currently a Post-Doctoral Research Associate with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA. His current research interests include computer architecture and high performance computing.

Tianshi Chen received the B.S. degree in mathematics from the Special Class for the Gifted Young, University of Science and Technology of China (USTC), Hefei, China, in 2005, and the Ph.D. degree in computer science from the Department of Computer Science and Technology, USTC, in 2010.

He is currently an Associate Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His current research interests include computer architecture, parallel computing, and computational intelligence.

Yunji Chen graduated from the Special Class for the Gifted Young, University of Science and Technology of China, Hefei, China, in 2002, and the Ph.D. degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences, Beijing, China, in 2007.

He is currently a Professor with ICT. He was a Chief Architect of Godson-3 processor. His current research interests include computer architecture and computational intelligence.

Franz Franchetti received the Dipl.-Ing. (M.Sc.) degree in technical mathematics and the Dr.Techn. (Ph.D.) degree in computational mathematics from the Vienna University of Technology, Vienna, Austria, in 2000 and 2003, respectively.

He is an Associate Research Professor with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA. His current research interests include automatic performance tuning and program generation for emerging parallel platforms and algorithm/hardware co-synthesis.

Prof. Franchetti was a recipient of the Carnegie Institute of Technology Dean’s Early Career Fellowship by the College of Engineering of Carnegie Mellon University in 2013. In 2006, he was a member of the team winning the Gordon Bell Prize (Peak Performance Award), and in 2010, he was a member of the team winning the HPC Challenge Class II Award (most productive system).