Efficient Uncertainty Modeling for System Design via Mixed Integer Programming

Zichang He*, Weilong Cui†, Chunfeng Cui*, Timothy Sherwood† and Zheng Zhang*
*Department of Electrical and Computer Engineering, †Department of Computer Science
University of California, Santa Barbara, CA 93106
Emails: zichanghe@ucsb.edu, cuiwl@cs.ucsb.edu, chunfengcui@ucsb.edu, sherwood@cs.ucsb.edu, zhengzhang@ece.ucsb.edu

***Abstract*—The post-Moore era casts a shadow of uncertainty on many aspects of computer system design. Managing that uncertainty requires new algorithmic tools to make quantitative assessments. While prior uncertainty quantification methods, such as generalized polynomial chaos (gPC), show how to work precisely under the uncertainty inherent to physical devices, these approaches focus solely on variables from a continuous domain. However, as one moves up the system stack to the architecture level many parameters are constrained to a discrete (integer) domain. This paper proposes an efficient and accurate uncertainty modeling technique, named mixed generalized polynomial chaos (M-gPC), for architectural uncertainty analysis. The M-gPC technique extends the generalized polynomial chaos (gPC) theory originally developed in the uncertainty quantification community, such that it can efficiently handle the mixed-type (i.e., both continuous and discrete) uncertainties in computer architecture design. Specifically, we employ some stochastic basis functions to capture the architecture-level impact caused by uncertain parameters in a simulator. We also develop a novel mixed-integer programming method to select a small number of uncertain parameter samples for detailed simulations. With a few highly informative simulation samples, an accurate surrogate model is constructed in place of cycle-level simulators for various architectural uncertainty analysis. In the chip-multiprocessor (CMP) model, we are able to estimate the propagated uncertainties with only 95 samples whereas Monte Carlo requires 5×10^4 samples to achieve the similar accuracy. We also demonstrate the efficiency and effectiveness of our method on a detailed DRAM subsystem.**

I. Introduction

Designing a computer system in an era of rapidly evolving applications and technology nodes involves many uncertainties. Computer system design have been known to be susceptible to all sorts of uncertainties from device-level process variations [1] to variations in application characteristics inside a datacenter [2]. Of course there is a deep library of work on quantifying uncertainty in architecture and system design that has been particularly focused on device and circuit level uncertainty [3–8] for us to draw upon, but as one moves up the system stack from the device to the architecture level and above many variables (e.g., cycles to satisfy an L1 cache miss or the number of bits of error correcting to use) are constrained to a discrete (e.g., integer) domain.

Quantifying uncertainty at the system level has been demonstrated via some high-level analytic models [9], yet doing so with detailed simulation is extremely challenging. On one hand, Monte Carlo (MC) methods require a large amount of data samples of the performance outputs due to its slow convergence rate. The time cost associated with acquiring such samples renders these techniques unsuitable when detailed simulator is used instead of analytic models, where one simulation sample usually costs minutes or hours (or even up to days in some large-scale system simulations). On the other hand, many advanced uncertainty quantification methods such as generalized polynomial chaos (gPC), although highly efficient, cannot handle the unique challenge of such a mixed-domain (continuous and discrete) problem.

We demonstrate a new algorithm for accurate uncertainty analysis in the context of computer system design by using only a small number of detailed simulations. To achieve this goal, we extend the generalized polynomial chaos (gPC) method [10], which is a powerful technique developed in the uncertainty quantification community. Due to its superior convergence rate and orders-of-magnitude speedup over MC in many applications, the gPC technique has been successfully applied to electronic design automation problems. Existing work includes fast stochastic modeling, simulation and optimization for electronic integrated circuits [11–13], integrated photonic devices and circuits [14], micro-electromechanical systems [15] and so forth. However, all these techniques can only handle analog behaviors/performance and only consider continuous uncertain parameters.

Paper Contributions. In this paper, we develop a new surrogate modeling technique, named mixed-gPC (M-gPC), for accurate uncertainty analysis of computer system design. Our algorithm employs a mixed integer programming method to handle mixed continuous and discrete uncertain parameters with a small number of simulations. To verify this theory experimentally in the context of computer system design, we examine its application to both a closed-form chip-multiprocessor (CMP) model and a detailed simulation-driven DRAM subsystem. Our specific contributions include:

- We present a CAD framework for architectural uncertainty analysis. Based on the given continuous and discrete distributions of input uncertain parameters, our algorithm automatically chooses the basis functions and simulation samples to build an accurate surrogate model.
- We present the numerical algorithms of building basis functions and choosing simulation samples. The key challenge is to decide a small number of highly informative simulation samples required in the stochastic collocation framework [16]. We formulate this problem as a mixed-

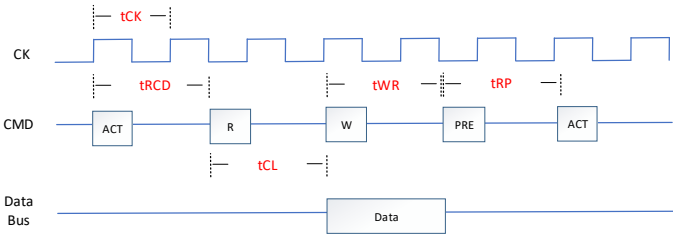

Fig. 1. All named DRAM timings here are considered under the influence of uncertainty (process variation). Note that the timings are not proportional to real number of cycles in this figure.

integer programming (MIP), and present a hierarchical decomposition method to efficiently solve it.
- We validate the proposed M-gPC framework by an analytical CMP model and a realistic DRAM subsystem. Our framework shows significant speedup and high accuracy on these two architectural analysis benchmarks.

II. Architectural Uncertainty Analysis Framework

We propose a M-gPC modeling method for computer architectural uncertainty analysis. The basic idea is to use a mixed-integer optimization method to select a few important samples for both continuous and discrete uncertain parameters. Then, an accurate stochastic surrogate model is constructed for architecture-level uncertainty analysis after a small number of detailed cycle-level simulations.

A. Problem Formulation and Overall Workflow

Problem Formulation. Given a computer architecture design case (e.g., a DRAM subsystem, see Fig. 1), let $\mathbf{y}(\boldsymbol{\xi})$ be some uncertain performance metrics of interest, such as the bandwidth or power. We use vector $\boldsymbol{\xi} = [\xi_1, \xi_2, \ldots, \xi_d]^T \in \mathbb{R}^d$ to denote some model uncertainty parameters, such as the time of one tick of clock (tCK), time precharge/recovery period (tRP), and so forth. We assume that these parameters are mutually independent, and each parameter ξ_i admits a probability density function $\rho_i(\xi_i)$. Please note that these uncertain parameters can be *either continuous or discrete*. For instance, tCK often obeys a truncated Gaussian distribution [17], and tRP often follows a binomial distribution [18] due to the process variations. Our goal is to quantify the uncertainty of $\mathbf{y}(\boldsymbol{\xi})$ caused by $\boldsymbol{\xi}$. Because each detailed cycle accurate simulation is extremely expensive, we instead want to build a surrogate model in the following form with only *a small number of simulations*:

$$\mathbf{y}(\boldsymbol{\xi}) \approx \sum_{|\boldsymbol{\alpha}|=0}^{p} \mathbf{c}_{\boldsymbol{\alpha}} \Psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi}). \tag{1}$$

Here, $\Psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi})$ is a stochastic basis function indexed by vector $\boldsymbol{\alpha}$, and p upper bounds the total order of the basis functions.

Overall Workflow. Before proceeding to the technical details, we first describe our high-level workflow in Fig. 2.

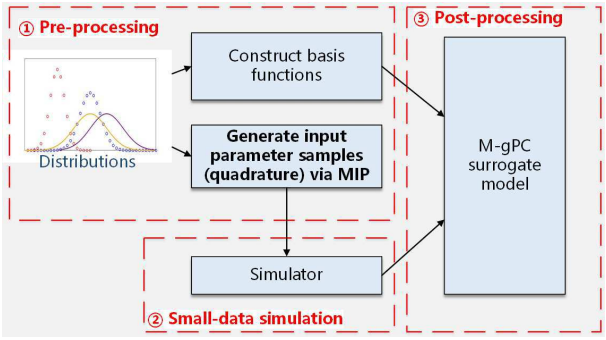

Fig. 2. Workflow of the architectural uncertainty analysis method.

- **Step 1: pre-processing.** Based on the given distributions of uncertain parameters $\boldsymbol{\xi}$ (which can be either continuous or discrete), we need to specify a class of basis functions $\{\Psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi})\}$. We also decide a few highly informative sample points and weights $\{\boldsymbol{\xi}_i, w_i\}_{i=1}^M$. Here w_i quantitatively describes the importance of sample $\boldsymbol{\xi}_i$.
- **Step 2: small-data simulations.** Given the carefully selected parameter samples, we call a high-fidelity cycle accurate simulator (e.g., the DRAMSim2 [19] for a DRAM) to simulate sample $\boldsymbol{\xi}_i$ for $i = 1, 2 \ldots, M$. The repeated simulations produce a set of output performance samples $\{\mathbf{y}(\boldsymbol{\xi}_i)\}_{i=1}^M$. Because M is very small, these detailed simulations can be done in a relatively short time.
- **Step 3: post-processing.** With $\{\mathbf{y}(\boldsymbol{\xi}_i)\}_{i=1}^M$ and the importance of each sample, we will decide the weight vector $\mathbf{c}_{\boldsymbol{\alpha}}$ for each basis function in Eq. (1).

B. The Proposed M-gPC Method

Three questions need to be answered in the above workflow:
- **Q1:** How shall we decide the basis functions?
- **Q2:** How shall we decide the parameter sample $\boldsymbol{\xi}_i$ and weight w_i, and the proper number of samples M?
- **Q3:** How shall we post-process the data to get the surrogate model (1)?

The first and third questions are already addressed in the standard gPC method [10]. Given the probability density function of each parameter, the gPC method selects a set of orthornormal polynomials as the basis functions:

$$\mathbb{E}\left[\Psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi})\,\Psi_{\boldsymbol{\beta}}(\boldsymbol{\xi})\right] = \begin{cases} 1, & \text{if } \boldsymbol{\alpha} = \boldsymbol{\beta}; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Here the index vector $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_d]^T$ denotes the polynomial degree of each uncertain parameter in the basis function. As a result, the expansion (1) employs $N_p = \binom{d+p}{p}$ basis functions in total. This choice of basis functions allows the stochastic collocation method [10] to compute each unknwon $\mathbf{c}_{\boldsymbol{\alpha}}$ by a projection procedure:

$$\mathbf{c}_{\boldsymbol{\alpha}} = \mathbb{E}\left[\mathbf{y}(\boldsymbol{\xi})\,\Psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi})\right] \approx \sum_{i=1}^{M} \mathbf{y}(\boldsymbol{\xi}_i)\,\Psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi}_i)\,w_i. \tag{3}$$

Here $\{\boldsymbol{\xi}_i, w_i\}_{i=1}^M$ are the quadrature samples and weights that we need to determine. Once the surrogate model is

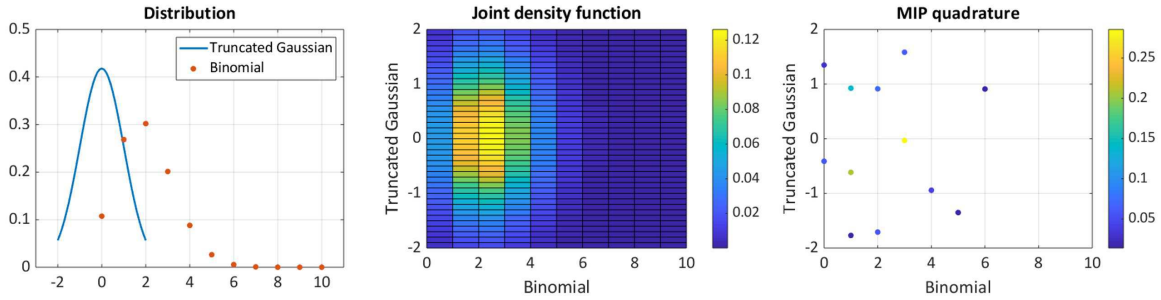

Fig. 3. Example of mixed-integer programming-based quadrature for a discrete variable with distribution Binomial$(10, 0.2)$ and another continuous one with a truncated Gaussian distribution in the range $(-2, 2)$. The color bar of MIP quadrature represents the weights of all samples.

constructed, we can easily obtain the mean value and standard deviation of the output performance via Eq. (4):

$$\mathbb{E}[\mathbf{y}(\boldsymbol{\xi})] \approx \mathbf{c_0}, \quad \sigma[\mathbf{y}(\boldsymbol{\xi})] \approx \sqrt{\sum_{|\boldsymbol{\alpha}|=1}^{p} \mathbf{c}_{\boldsymbol{\alpha}}^2}, \tag{4}$$

where all calculations are performed component-wise. We can also obtain the histogram or probability distribution of $\mathbf{y}(\boldsymbol{\xi})$ without calling the detailed cycle accurate simulator again. The details of basis functions are given in Appendix A.

Sample and weight selection in M-gPC. However, addressing the second question is difficult. According to (3), we need to choose a small number of samples and weights to accurately estimate the numerical integration. Numerous numerical integration rules (e.g., Gauss quadrature rule and sparse grids [10]) are available for continuous variables, but they cannot handle discrete variables. A naive choice is to use all possible samples of a discrete uncertain variable, but this leads to a huge number of simulation samples. In order to address this issue, we develop a novel mixed-integer programming method to generate a few high-quality quadrature samples and weights. We defer the details of this approach to Section III, and we first demonstrate this approach by a simple example. We consider a problem with two uncertain variables: the continuous one is truncated Gaussian in the range of $(-2, 2)$, and the discrete one obeys a binomial distribution. When choosing quadrature samples, the first variable can be any value in the range of $(-2, 2)$, but the second one can only take some integer values below 10. The results of our mixed integer programming are shown in Fig. 3: only 12 samples with different weights are generated.

III. Proposed MIP-Based Quadrature Rule

This section presents our mixed-integer programming solver to determine the quadrature points and weights $\{\boldsymbol{\xi}_i, w_i\}_{i=1}^M$ required in Eq. (3). The quadrature samples are provided to a cycle-level simulator for repeated simulations. Therefore, we hope to make M as small as possible.

A. Optimization Formulation

Let $\mathcal{I} = \{i \,|\, \xi_i \in \mathbb{Z}\}$ denote the index set of integer parameters in $\boldsymbol{\xi}$. We propose to compute the quadrature points and weights by an optimization-based method. Our method differs from [20, 21]: our proposed method can handle both discrete and continuous uncertain parameters, whereas [20, 21] can only handle continuous uncertain variables.

Eq. (3) requires the integration of polynomials up to order $2p$ if $y(\boldsymbol{\xi})$ can be well approximated by an order-p expansion. Therefore, our optimization problem is set up by matching the integration of basis functions up to order $2p$. If $|\boldsymbol{\alpha}| \le 2p$, then there is a one-to-one correspondence between $\boldsymbol{\alpha}$ and integer $k \in [1, N_{2p}]$ where $N_{2p} = (d + 2p)!/(d!)(2p!)$. We use the scalar index for simplicity, and seek for $\{\boldsymbol{\xi}_i, w_i\}_{i=1}^M$ such that

$$\mathbb{E}[\Psi_k(\boldsymbol{\xi})] = \sum_{i=1}^{M} \Psi_k(\boldsymbol{\xi}_i)\, w_i, \ \forall\, k \in [N_{2p}]$$
$$\text{where } w_i \ge 0 \text{ and } \boldsymbol{\xi}_{i,\mathcal{I}} \in \mathbb{Z}^{|\mathcal{I}|}. \tag{5}$$

Here, $\boldsymbol{\xi}_{i,\mathcal{I}} = \{\boldsymbol{\xi}_{i,l}\}_{l\in\mathcal{I}}$ denotes all integer elements in sample $\boldsymbol{\xi}_i$. Due to the orthonormal condition, we know $\mathbb{E}[\Psi_k(\boldsymbol{\xi})] = \mathbb{E}[\Psi_k(\boldsymbol{\xi})\Psi_1(\boldsymbol{\xi})] = \delta_{1k}$.

As a result, we solve $\{\boldsymbol{\xi}_i, w_i\}_{i=1}^M$ via the following nonlinear least squares

$$\min_{\bar{\boldsymbol{\xi}}, \mathbf{w}} \quad \|\boldsymbol{\Phi}(\bar{\boldsymbol{\xi}})\mathbf{w} - \mathbf{e}_1\|_2^2, \quad \text{s.t.} \quad \mathbf{w} \ge 0, \ \bar{\boldsymbol{\xi}}_{\mathcal{I}} \in \mathbb{Z}^{M|\mathcal{I}|}. \tag{6}$$

where $\bar{\boldsymbol{\xi}} = [\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_M]^T \in \mathbb{R}^{M\times d}$, $\bar{\boldsymbol{\xi}}_{\mathcal{I}} = \{\boldsymbol{\xi}_{i,\mathcal{I}}\}_{i=1}^M$, $\boldsymbol{w} = [w_1, w_2, \ldots, w_M]^T \in \mathbb{R}^M$, $\mathbf{e}_1 = [1, 0, \ldots, 0]^T \in \mathbb{R}^{N_{2p}}$, $\boldsymbol{\Phi}(\bar{\boldsymbol{\xi}}) \in \mathbb{R}^{N_{2p}\times M}$ with each elements $\left[\boldsymbol{\Phi}(\bar{\boldsymbol{\xi}})\right]_{ki} = \Psi_k(\boldsymbol{\xi}_i)$. There are $M \times (d+1)$ unknown variables in total, which can be a large-scale optimization problem as d increases.

B. Hierarchical BCD for Solving Problem (6)

Eq. (6) is a large-scale mixed integer nonlinear programming problem (MINLP) with $M|\mathcal{I}|$ integers. We solve this hard problem by a hierarchical decomposition method. Our key idea is to simplify the original large scale MINLP into several easier sub-problems.

We intend to employ a block coordinate descent (BCD) method to solve the quadrature points and weights separately. However, directly applying it to our mixed-integer case is inefficient since the complexity of mixed-integer programming grows exponentially with the number of integer variables [22]. In order to further speed up the algorithm, we separate the

Algorithm 1: Hierarchical BCD for solving (6)

Input: Initial points and weights, maximal outer iteration n_{max}, and solver error tolerance ε.
Output: Optimized points and weights $\{\boldsymbol{\xi}_i, w_i\}_{i=1}^{M}$;
for $t = 1, 2, \ldots, n_{\max}$ **do**
  for $i = 1, 2, \ldots, M$ **do**
    Solve $\Delta\boldsymbol{\xi}_i$ via Eq. (7);
    Update ith point via solving Eq. (8);
    Update all weights via solving Eq. (9);
  if $\left\|\Phi\left(\bar{\boldsymbol{\xi}}^t\right)\boldsymbol{w}^t - \mathbf{e}_1\right\|_2^2 \leq \varepsilon$ **then**
    break; % converge
  else
    if $\left\|\Delta\bar{\boldsymbol{\xi}}\right\|_F \leq 10^{-8}$ **then**
      break; % not converge

Algorithm 2: MIP-based stochastic collocation.

Output: The M-gPC coefficients $\{\mathbf{c}_{\boldsymbol{\alpha}}\}_{|\boldsymbol{\alpha}|=0}^{p}$
1: Initialize the quadrature points and weights;
2: **Increase Phase.** Update points and weights via Alg. 1. If algorithm fails to converge, increase the number of points and go back to last line.
3: **Decrease Phase.** Update points and weights via Alg. 1. If algorithm converges, decrease the number of points until it fails.
4: Call the simulator to obtain $\{\mathbf{y}(\boldsymbol{\xi}_i)\}_{i=1}^{M}$
5: Calculate the M-gPC weight vectors via Eq. (3).

quadrature points into M blocks. Consequently, we only need to solve a MINLP with $|\mathcal{I}|$ integer variables for each sub-problem. Our detailed framework is shown in Alg. 1.

In Alg. 1, we have an outer iteration t and an inner iteration i. In this case, only one sample is optimized in each inner iteration, which only has d unknown variables. By applying the Gauss Newton method, the original problem is converted to a mixed-integer quadrature program (MIQP), which is much easier to solve. Considering that the Jacobian matrix J can be ill-conditioned, we adopt the Tikhonov regularization [23] here to make the MIP solver more stable:

$$\begin{aligned} \min_{\Delta\boldsymbol{\xi}_i} \quad & \left\|\hat{\mathbf{J}}_i^{t\times i}\Delta\boldsymbol{\xi}_i + \mathbf{r}^{t\times i}\right\|_2^2 + \lambda\left\|\Delta\boldsymbol{\xi}_i\right\|_2^2 \\ \text{s.t.} \quad & \Delta\boldsymbol{\xi}_{i,\mathcal{I}} \in \mathbb{Z}^{|\mathcal{I}|}, \end{aligned} \tag{7}$$

where $\mathbf{r}^{t\times i} = \boldsymbol{\Phi}\left(\bar{\boldsymbol{\xi}}^{t\times i-1}\right)\boldsymbol{w}^{t\times i} - \mathbf{e}_1$ denotes the residual given the ith points under the $(t\times i)$th iteration and $\mathbf{J}_i^{t\times i}$ denotes the Jacobian matrix of $\mathbf{r}^{t\times i}$, and λ is a regularization parameter. Then we can update each quadrature point as

$$\boldsymbol{\xi}_i^t = \boldsymbol{\xi}_i^{t-1} + \Delta\boldsymbol{\xi}_i. \tag{8}$$

After one sample is optimized, we fix all the points and update the weights via solving a linear least square problem:

$$\boldsymbol{w}^{t\times i} = \arg\min_{\boldsymbol{w}} \left\|\boldsymbol{\Phi}\left(\bar{\boldsymbol{\xi}}^{t\times i}\right)\boldsymbol{w} - \mathbf{e}_1\right\|_2^2 \tag{9}$$

In summary, we decompose the original large-scale MINLP problem [Eq. (6)] to many linear least square problems [Eq. (9)] and small-scale MIQP problems [Eq. (7)], which are much more efficient to solve. The comprehensive algorithm is summarized in Alg. 1. With the proposed MIP-based quadrature, we can select a small number of samples to calculate the M-gPC coefficients via Eq. (3).

C. Initialization and Number of Quadrature Points

In practice, a global optimal solution to problem (6) is unnecessary: any solution with a high accuracy and leading to a small number of quadrature points can be used to build a M-gPC surrogate model with good performance. However, choosing a good initial guess is important to ensure high accuracy. We employ the weighted complete linkage clustering method [20] to generate the initial guess. Firstly, many candidate samples with weights are randomly generated by Monte Carlo. Then, they are clustered to a smaller number of points. In this process, any two points with a minimal weighted distance can be merged into one point sequentially until the number of points achieves the initial setting. The weighted distance between two sample points is defined as follows:

$$D_{ij} = (w_i + w_j)\left(\max_{\boldsymbol{\xi}_1\in C_1, \boldsymbol{\xi}_2\in C_2} d\left(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2\right)\right). \tag{10}$$

In order to properly determine the number of quadrature points M, an increase-decrease module [21] is adopted. Firstly, we increase M to ensure that (6) can be solved with a high accuracy. Then, we decrease M and solve (6) to check if an accurate quadrature rule can be found with fewer samples. Integrating this module with our framework, the overall MIP-based stochastic collocation method is summarized in Alg. 2.

We have the following remarks. Firstly, the CPU time of the current hierarchical algorithm is negligible compared to the CPU time cost by sample simulations, especially for low-dimensional problems. Secondly, the theoretical guarantees on the approximation error and on the number of quadrature points in [21] still hold in our proposed MIP-based framework.

IV. Case Studies

A. Analytical CMP Models

1) Uncertainties in Analytical CMP Models: Here, an analytical CMP model in [9] is used to verify the proposed architectural uncertainty analysis framework. A more general heterogeneous core selection problem based on that of Hill and Marty [24] is illustrated in Table I. In terms of uncertainty description, the inputs parallelism of the application (f), communication overhead among cores (c) and designed number of each core on chip (N_{core_i}) are discrete, while performance of each type of core (P_{core_i}) and $yield_{\text{core}_i}$ are continuous. We use the same uncertainty models described in previous work [9] shown in Table II. Its parameter setting in a case of two cores is shown in Table II.

TABLE I
CLOSED FORM OF CMP MODEL.

$$\text{Speedup} = 1/\left(T_{\text{sequential}} + T_{\text{parallel}}\right)$$
$$T_{\text{sequential}} = (1 - f + c \times N_{\text{core}})/P_{\text{serial}}$$
$$T_{\text{parallel}} = f/P_{\text{parallel}}$$
$$P_{\text{serial}} = \max\{P_{\text{core}_i} | N_{\text{core}_i} > 0\}$$
$$P_{\text{parallel}} = \sum_{i \in \text{core types}} N_{\text{core}_i}$$
$$P_{\text{core}_i} = \sqrt{A_{\text{core}_i}}$$
$$A_{\text{total}} = \sum_{i \in \text{core types}} N_{\text{core}_i} \times A_{\text{core}_i}$$

TABLE II
UNCERTAIN PARAMETERS SETTING IN THE CMP MODEL.

Uncertainty model	Parameter setting	
$f \sim \frac{\text{Binomial}(M,p)}{M}$	$M = 60,\ p = 0.6$	
$c \sim \frac{\text{Binomial}(M,p)}{M}$	$M = 80,\ p = 0.7$	
$N_{\text{core}_i} \sim \text{Binomial}\left(M, yield_{\text{core}_i}\right)$	$M = 20,\ yield_{\text{core}_0} = 0.7432$	
	$M = 20,\ yield_{\text{core}_1} = 0.5739$	
$P_{\text{core}_i} \sim \text{Truncated Gaussian}\ (\mu, \sigma, 0)$	$\mu_0 = 5.6569,\ \sigma_0 = 1.1314$	
	$\mu_1 = 8,\ \sigma_1 = 1.6$	
$yield_{\text{core}_i} = \left(1 + \frac{d \times A_{\text{core}_i}}{\alpha}\right)^{-\alpha}$	$A_{\text{core}_0} = 32$	$A_{\text{core}_1} = 64$

Under such uncertain parameters, our task is to approximate the moments and distribution of Speedup as close as possible to estimate the propagated uncertainty in the model.

2) Numerical Results: The results of a 2nd-order M-gPC model under different thresholds ε are shown in Table III. A higher order leads to more accurate approximation, but order 2 already works well in this case. We can find that ε can control the number of M-gPC samples and accuracy of the performance. The results show that the algorithm performance may be limited by a too large or too small ε, but it performs well with small samples in capturing the mean value and standard deviation when ε ranges from 10^{-3} to 10^{-6}. In these cases, compared with the relative root mean square error (RMSE) and relative mean absolute error (MAE) between the M-gPC and MC are all around 0.03 and 0.02 respectively. Especially, to achieve the same level of accuracy, the sample number needed in a M-gPC model is much smaller than MC methods. For example, if we take the results of 10^5 MC as the ground truth, then 95 M-gPC simulation samples can already achieve the same accuracy of 5×10^4 MC samples in capturing the mean value The histograms of $\varepsilon = 10^{-5}$ case under 10^5 MC samples are illustrated in Fig. 4.

B. Uncertainties in Detailed Memory Subsystems

1) Experiment Setup: To faithfully capture the uncertainties in a DRAM subsystem, we use the detailed DRAM simulator DRAMSim2 [19]. We model the uncertainty parameter after previous works [17, 18] as shown in Table IV (also see Fig. 1). We run a collection of memory traces from SPEC 2017 benchmark suit [25] shown in Table V.

We also simulate a memory system with 1 memory channel under the JEDEC standards with a 32-entry command queue and a 32-entry transaction queue. For the DRAM itself, we experiment on a collection of different DDR3 devices shown in Table VI.

TABLE III
SPEEDUP PERFORMANCE OF DIFFERENT MODELS.

	Sample	Mean	Std	RMSE	MAE	ε
M-gPC	84	0.4353	0.1021	0.0418	0.0311	1e-2
	85	0.4382	0.0974	0.0338	0.0231	1e-3
	87	0.4380	0.0992	0.0306	0.0208	1e-4
	95	0.4376	0.0986	0.0306	0.0228	1e-5
	123	0.4376	0.0987	0.0314	0.0233	1e-6
	179	0.4386	0.0982	0.0289	0.0205	1e-7
	182	0.4387	0.0975	0.0294	0.0214	1e-8
MC	1e3	0.4369	0.1011	N/A	N/A	N/A
	5e3	0.4370	0.1002			
	1e4	0.4383	0.0995			
	5e4	0.4375	0.099			
	1e5	0.4377	0.0987			

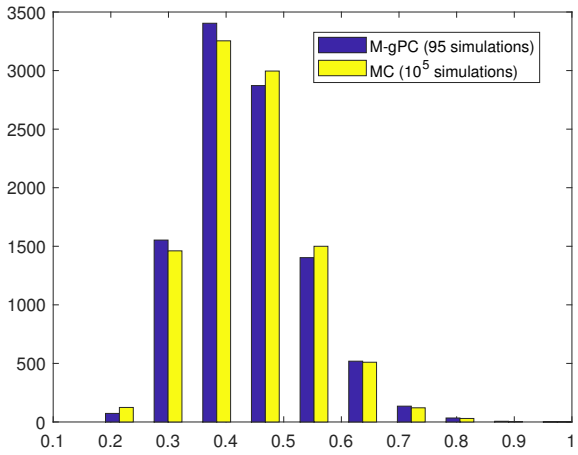

Fig. 4. Histograms of speedup performance with $\varepsilon = 10^{-5}$.

TABLE IV
UNCERTAIN PARAMETERS SETTING IN THE DRAM SUBSYSTEM.

$$\text{tCK} \sim \text{Truncated Gaussian}\ (\mu, \sigma, 0)$$
$$\text{tRCD} \sim \text{Binomial}\ (M, p)$$
$$\text{tCL} \sim \text{Binomial}\ (M, p)$$
$$\text{tRP} \sim \text{Binomial}\ (M, p)$$
$$\text{tWR} \sim \text{Binomial}\ (M, p)$$

TABLE V
WORKLOADS AND SIMULATION TIME COMPARISON.

Workload	Length of trace	MC time	M-gPC time
600.peribench	46.8M	∼15.1h	∼4.5h + 12.5m
602.gcc	35.7M	∼8.8h	∼2.6h + 12.5m
605.mcf	43.5M	∼14.7h	∼4.4h + 12.5m
623.xalancbmk	42.9M	∼12.6h	∼3.8h + 12.5m
625.x264	30.5M	∼9h	∼2.7h + 12.5m
631.deepsjeng	37.6M	∼12.3h	∼3.7h + 12.5m
641.leela	36.1M	∼11.6h	∼3.5h + 12.5m
998.specrand	32.9M	∼10.6h	∼3.2h + 12.5m

TABLE VI
DDR3 DEVICES USED IN THE DRAM SUBSYSTEM.

Label	Width	Capacity	Internal frequency
1	4b	64MB	667MHz
2	8b	32MB	400MHz
3	8b	32MB	667MHz
4	4b	32MB	800MHz
5	4b	32MB	667MHz
6	8b	16MB	667MHz
7	16b	8MB	667MHz

2) Numerical Results: We use a 2nd-order M-gPC with threshold $\varepsilon = 10^{-3}$ to build a surrogate model for the DRAM simulator. Due to the practical timing issue, we only run 200 MC samples to verify the effectiveness of the M-gPC model. We solve (6) to get 60 quadrature samples and weights for M-gPC in MATLAB with a 3.4GHz 8GB memory desktop, which takes 12.5 min. The time comparison between M-gPC modeling and MC for the first device configuration under 20% uncertainty is shown in Table V. Clearly, the cost of solving (6) is negligible compared with the total simulation time.

Performance under different uncertainty levels: in our setting, the mean value is set as the configuration without uncertainty, and the standard deviation σ is set as $\sigma = \alpha \cdot \mu$, where α is defined as the uncertainty level. For binomial distributions, we can also use Gaussian parameters to approximate it. The performance of aggregate average bandwidth, average power, average latency and data bus (DBUS) utilization are illustrated in Fig. 5. It is expected that the standard deviation will increase when the uncertainty level increases, which can be well captured by the M-gPC model with around only 60 samples. The sample number may vary since different input distributions lead to different M-gPC samples. The histograms of these four metrics are shown in Fig. 6. The M-gPC model can capture the performance distributions well. For the bandwidth, the approximating error may be more sensitive to uncertainty levels. This is because the bandwidth performance function appears to be an inverse curve, which is relatively harder for a 2nd-order M-gPC to approximate. In this case, we can increase the order to get better approximation. These results verify that the M-gPC model is able to handle the cases under different levels of uncertainty.

Performance under different device configurations: we run different device configurations with 20% uncertainties on the 602.gcc trace. The results are illustrated in Fig. 7, which show that, for different DRAM devices, M-gPC model can approximate the moments well with only around 60 samples. The errors for approximating the distribution are also small: RMSE varies in 1%-4%, MAE varies in 0.8%-2.4%.

Performance under different workloads: we run the first device configuration with 20% uncertainty on different workloads. The results are illustrated in Table VII. The moments and distribution are also approximated with a high accuracy, which shows the M-gPC model with only 66 simulation samples work well on different workloads too.

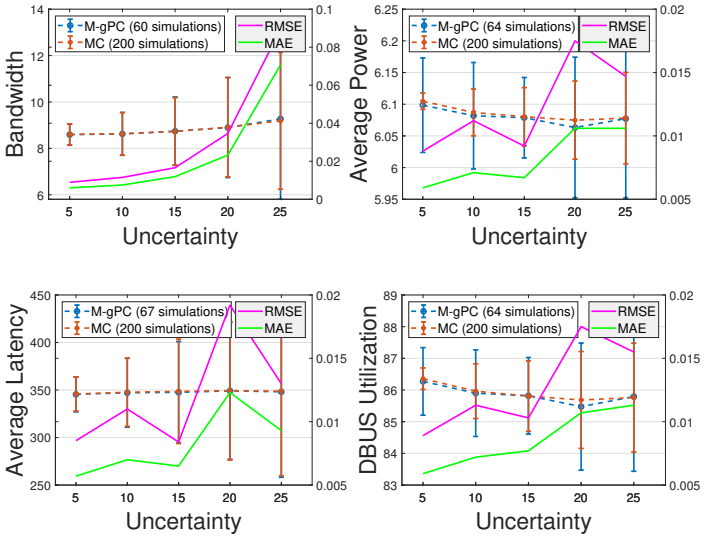

Fig. 5. Performance under different uncertainty levels: M-gPC model (around 60 simulation samples) vs MC method (200 simulation samples).

V. CONCLUSION

Uncertainty analysis for system design is an increasingly important concern especially when we navigate through the vast design space with many emerging and immature technologies. The general MC method to analyze uncertainty is limited due to the expensive simulations in a practical architecture, while efficient modeling methods such as generalized polynomial chaos (gPC) cannot handle the unique challenge when we move from the analog device world to a discrete architecture design. Many parameters and configurations are inherently discrete/integers at the system level, hence the uncertainty analysis becomes a mixed-domain problem. To address these challenges, we propose a novel mixed-integer programming method to find a quadrature, and an M-gPC model that can handle both continuous and discrete inputs. The results of an analytical CMP model have shown that our framework with 95 samples can approximate the results of a MC method with 5×10^4 samples. We have also verified the proposed uncertainty analysis framework on detailed DRAM subsystems. There are still many open problems for architectural uncertainty analysis, such as the high-dimensional cases and non-smooth output performance. The proposed architectural uncertainty analysis framework can be used in many application cases in the future.

ACKNOWLEDGMENT

This work was supported by NSF CCF No. 1763699 and NSF CAREER Award No. 1846476.

APPENDIX A
BASIS FUNCTION CONSTRUCTION

When the uncertain parameters are mutually independent, a multivariate basis function can be constructed based on the

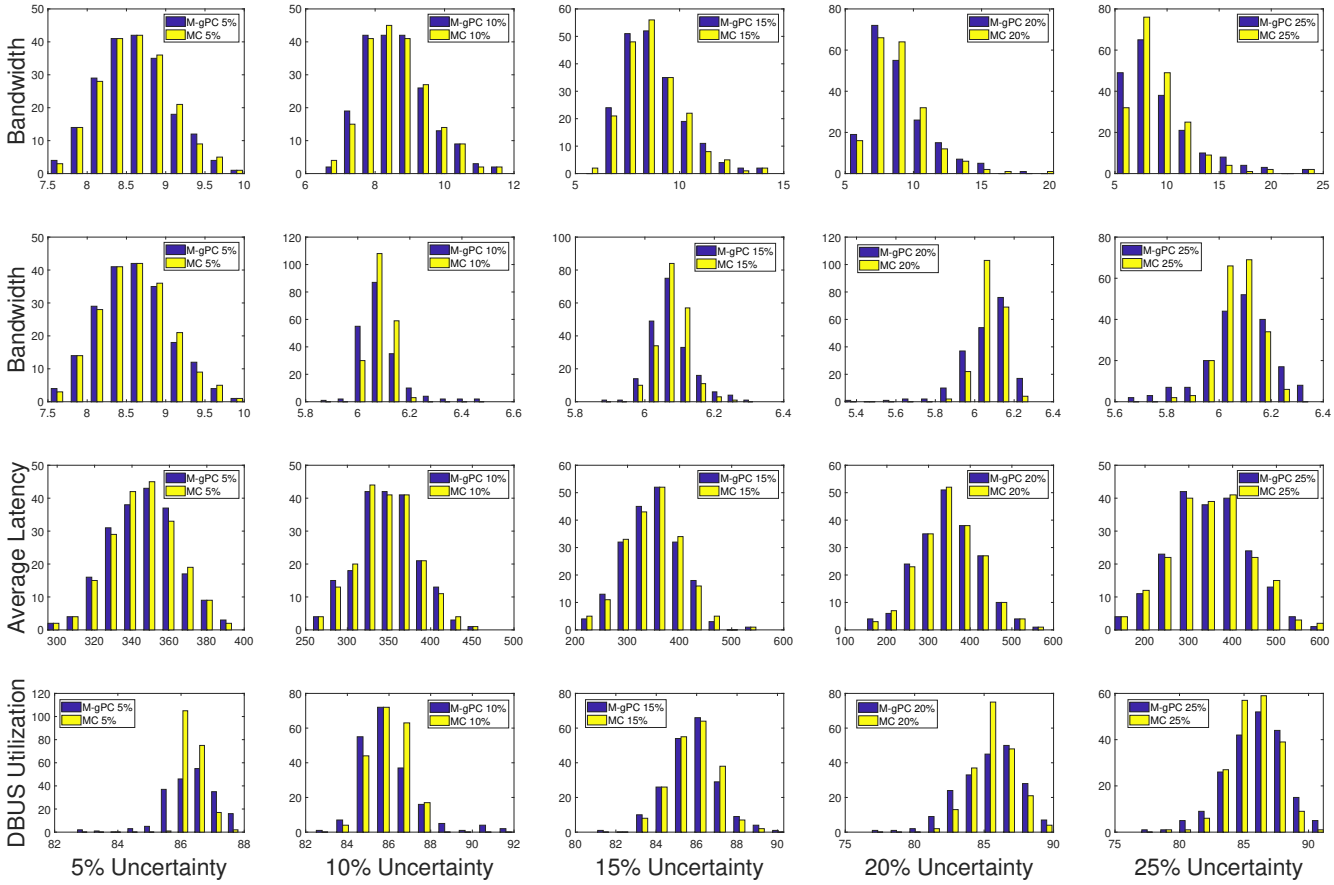

Fig. 6. Histograms of different performances under different uncertainty levels.

TABLE VII
PERFORMANCE UNDER DIFFERENT WORKLOADS: M-GPC MODEL (66 SIMULATION SAMPLES) VS MC METHOD (200 SIMULATION SAMPLES).

	Workloads	peribench	gcc	mcf	xalancbmk	x264	deepsjeng	leela	specrand
Bandwidth (GB/s)	Mean (M-gPC)	7.2847	8.9289	6.4874	7.8169	7.3144	7.1424	7.1413	7.2011
	Mean (MC)	7.2834	8.9332	6.4868	7.8201	7.3146	7.1434	7.1422	7.2047
	Std (M-gPC)	1.7545	2.0969	1.5718	1.8619	1.747	1.7194	1.7142	1.7318
	Std (MC)	1.7875	2.1479	1.6026	1.9002	1.7794	1.7538	1.7503	1.772
	RMSE	0.0318	0.0322	0.0319	0.0321	0.032	0.0319	0.0319	0.0319
	MAE	0.0186	0.0185	0.0185	0.0185	0.0187	0.0185	0.0185	0.0187
Average Power (watts)	Mean (M-gPC)	5.6782	6.0804	5.3127	5.7486	5.3717	5.5257	5.4308	5.5089
	Mean (MC)	5.6796	6.0831	5.3149	5.7517	5.3731	5.5281	5.4333	5.5112
	Std (M-gPC)	0.2159	0.1362	0.2168	0.1832	0.1871	0.2073	0.1934	0.2113
	Std (MC)	0.18	0.0817	0.1858	0.1428	0.1486	0.1732	0.1605	0.1773
	RMSE	0.0123	0.0124	0.0121	0.0122	0.0123	0.0121	0.0121	0.0125
	MAE	0.0094	0.0096	0.0093	0.0093	0.0094	0.0093	0.0093	0.0096
Average Latency (ns)	Mean (M-gPC)	423.5401	347.5645	462.5911	393.1547	431.4108	438.715	424.1744	428.8241
	Mean (MC)	424.0171	348.0141	463.0697	393.5325	432.0178	439.1198	424.5305	429.1871
	Std (M-gPC)	90.7912	71.9639	99.6712	82.6582	91.3793	93.9994	90.5722	91.6698
	Std (MC)	89.3849	71.4477	97.9099	81.544	90.3919	92.4259	89.0638	90.7059
	RMSE	0.0129	0.013	0.0129	0.013	0.0131	0.0129	0.013	0.0132
	MAE	0.0101	0.0101	0.01	0.0103	0.0102	0.0101	0.0102	0.0103
DBUS Utilization (%)	Mean (M-gPC)	70.0493	85.8639	62.3914	75.1677	70.3477	68.6842	68.6735	69.2553
	Mean (MC)	70.0251	85.8725	62.3752	75.1851	70.333	68.6808	68.669	69.2661
	Std (M-gPC)	4.0931	2.5388	4.1026	3.5501	3.7875	4.0072	3.8193	4.1252
	Std (MC)	3.7132	1.9014	3.7994	3.0927	3.3418	3.6469	3.4791	3.7686
	RMSE	0.0125	0.0128	0.0124	0.0123	0.0125	0.0122	0.0122	0.0128
	MAE	0.0097	0.0098	0.0096	0.0094	0.0096	0.0094	0.0094	0.01

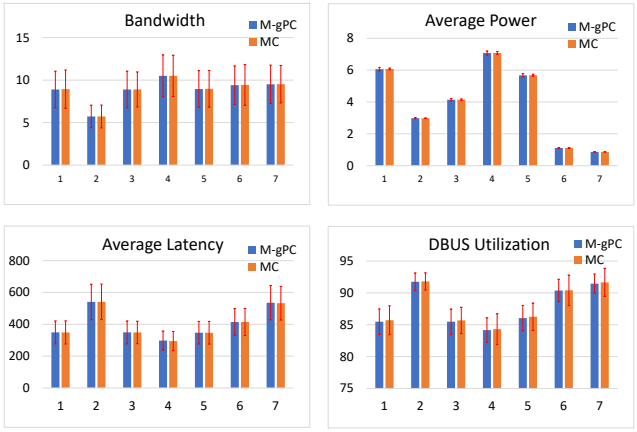

Fig. 7. Performance under different device configurations: M-gPC model (around 60 simulation samples) vs MC method (200 simulation samples).

product of some univariate ones:

$$\Psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi}) = \prod_{i=1}^{d} \phi_{\alpha_i}^{(i)}(\xi_i). \tag{11}$$

Here $\phi_{\alpha_i}^{(i)}$ is a degree-α_i polynomial basis function for parameter ξ_i. Given the marginal probability density function $\rho_i(\xi_i)$ for each variable ξ_i, a set of uni-variate orthornormal polynomials $\left\{\phi_m^{(i)}, m \in \mathbb{N}\right\}$ can be constructed by the well-known three-term recurrence relation [26], which satisfy:

$$\mathbb{E}[\phi_m^{(i)} \phi_n^{(i)}] = \delta_{m,n}, \ \forall\, i = 1, \ldots, d. \tag{12}$$

Here $\delta_{m,n}$ is a Delta function. The basis functions for discrete variable are also defined in a discrete or integer domain.

The three term recurrence [26] is performed as follows:

$$\begin{aligned} \pi_{i+1}(x) &= (x - \alpha_i)\,\pi_i(x) - \beta_i(x)\,\pi_{i-1}(x) \\ \pi_{-1}(x) &= 0, \quad \pi_0(x) = 1, \quad i = 0, 1, \ldots, n \end{aligned} \tag{13}$$

where

$$\alpha_i = \frac{E\left[x\pi_i^2(x)\right]}{E\left[\pi_i^2(x)\right]}, \quad \beta_{i+1} = \frac{E\left[\pi_{i+1}^2(x)\right]}{E\left[\pi_i^2(x)\right]}, \quad i = 0, 1, \ldots, n \tag{14}$$

and $\beta_i = 1$. Then the orthonormal polynomials are obtained via normalizing the above obtained polynomials:

$$\phi_i(x) = \frac{\pi_i(x)}{\sqrt{\beta_0 \beta_1 \cdots \beta_i}}, \quad i = 0, 1, \ldots, n. \tag{15}$$

REFERENCES

[1] S. Mittal, "A survey of architectural techniques for managing process variation," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 54:1–54:29, Feb. 2016.

[2] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: Insights from google compute clusters," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 4, pp. 34–41, Mar. 2010.

[3] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proc. DAC*, 2003, pp. 338–342.

[4] S. Bhardwaj and S. B. K. Vrudhula, "Leakage minimization of nano-scale circuits in the presence of systematic and random variations," in *Proc. Design Autom. Conf.*, 2005, pp. 541–546.

[5] L. Zhang, L. S. Bai, R. P. Dick, L. Shang, and R. Joseph, "Process variation characterization of chip-level multiprocessors," in *Proc. Design Autom. Conf.*, 2009, pp. 694–697.

[6] H.-Y. Wong, L. Cheng, Y. Lin, and L. He, "FPGA device and architecture evaluation considering process variations," in *Proc. ICCAD*, 2005, pp. 19–24.

[7] A. Das, S. Ozdemir, G. Memik, J. Zambreno, and A. Choudhary, "Mitigating the effects of process variations: Architectural approaches for improving batch performance," in *Workshop on Arch. Support for Gigascale Integr.*, 2007.

[8] G. Yan, F. Sun, H. Li, and X. Li, "Corerank: Redeeming "dark silicon" by dynamically quantifying core-level healthy condition," *IEEE Trans. Computers*, vol. 65, no. 3, pp. 716–729, March 2016.

[9] W. Cui and T. Sherwood, "Estimating and understanding architectural risk," in *Proc. MICRO*, 2017, pp. 651–664.

[10] D. Xiu, *Numerical methods for stochastic computations: a spectral method approach*. Princeton university press, 2010.

[11] Z. Zhang, T. A. El-Moselhy, I. M. Elfadel, and L. Daniel, "Stochastic testing method for transistor-level uncertainty quantification based on generalized polynomial chaos," *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 32, no. 10, pp. 1533–1545, 2013.

[12] Z. Zhang, I. A. M. Elfadel, and L. Daniel, "Uncertainty quantification for integrated circuits: Stochastic spectral methods," in *Proc. Int. Conf. Computer-Aided Design*, 2013, pp. 803–810.

[13] Z. Zhang, T.-W. Weng, and L. Daniel, "Big-data tensor recovery for high-dimensional uncertainty quantification of process variations," *IEEE Trans. Compon. Packag. Manuf. Techol.*, vol. 7, no. 5, pp. 687–697, 2016.

[14] C. Cui and Z. Zhang, "Uncertainty quantification of electronic and photonic ICs with non-Gaussian correlated process variations," in *Proc. ICCAD*, 2018, pp. 1–8.

[15] N. Agarwal and N. R. Aluru, "A domain adaptive stochastic collocation approach for analysis of mems under uncertainties," *J. Comp. Phys.*, vol. 228, no. 20, pp. 7662–7688, 2009.

[16] D. Xiu and J. S. Hesthaven, "High-order collocation methods for differential equations with random inputs," *SIAM Journal on Scientific Computing*, vol. 27, no. 3, pp. 1118–1139, 2005.

[17] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Varius: A model of process variation and resulting timing errors for microarchitects," *IEEE Trans. Semiconductor Manufacturing*, vol. 21, no. 1, pp. 3–13, 2008.

[18] K. Chandrasekar, S. Goossens, C. Weis, M. Koedam, B. Akesson, N. Wehn, and K. Goossens, "Exploiting expendable process-margins in DRAMs for run-time performance optimization," in *Proc. Design, Autom. & Test in Europe*, 2014, p. 173.

[19] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.

[20] C. Cui, M. Gershman, and Z. Zhang, "Stochastic collocation with non-Gaussian correlated parameters via a new quadrature rule," in *Proc. IEEE Conf. EPEPS*, 2018, pp. 57–59.

[21] C. Cui and Z. Zhang, "Stochastic collocation with non-Gaussian correlated process variations: Theory, algorithms, and applications," *IEEE Trans. Compon. Packag. Manuf. Techol.*, vol. 9, no. 7, pp. 1362–1375, July 2019.

[22] I. E. Grossmann, "Review of nonlinear mixed-integer and disjunctive programming techniques," *Optimization and engineering*, vol. 3, no. 3, pp. 227–252, 2002.

[23] P. C. Hansen, "Analysis of discrete ill-posed problems by means of the l-curve," *SIAM review*, vol. 34, no. 4, pp. 561–580, 1992.

[24] M. Hill and M. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[25] "Spec cpu 2017," https://www.spec.org/cpu2017/, accessed: 2019-04-08.

[26] W. Gautschi, "On generating orthogonal polynomials," *SIAM J. Sci. Stat. Comp.*, vol. 3, no. 3, pp. 289–317, 1982.