Estimating and Understanding Architectural Risk

Weilong Cui
University of California, Santa Barbara
cuiwl@cs.ucsb.edu

Timothy Sherwood
University of California, Santa Barbara
sherwood@cs.ucsb.edu

ABSTRACT

Designing a system in an era of rapidly evolving application behaviors and significant technology shifts involves taking on risk that a design will fail to meet its performance goals. While risk assessment and management are expected in both business and investment, these aspects are typically treated as independent to questions of performance and efficiency in architecture analysis. As hardware and software characteristics become uncertain (i.e. samples from a distribution), we demonstrate that the resulting performance distributions quickly grow beyond our ability to reason about with intuition alone. We further show that knowledge of the performance distribution can be used to significantly improve both the average case performance and minimize the risk of underperformance (which we term architectural risk). Our automated framework can be used to quantify the areas where trade-offs between expected performance and the "tail" of performance are most acute and provide new insights supporting architectural decision making (such as core selection) under uncertainty. Importantly it can do this even without a priori knowledge of an analytic model governing that uncertainty.

CCS CONCEPTS

• **Computing methodologies → Uncertainty quantification**; *Modeling methodologies*; • **Computer systems organization → *Multicore architectures***;

KEYWORDS

Uncertainty, Random Variable, Core Selection, Architecture Modelling

ACM Reference format:
Weilong Cui and Timothy Sherwood. 2017. Estimating and Understanding Architectural Risk. In *Proceedings of MICRO-50, Cambridge, MA, USA, October 14–18, 2017,* 14 pages.
https://doi.org/10.1145/3123939.3124541

1 INTRODUCTION

Computer architecture has always been governed by a combination of physical laws, human creativity, and economic realities. The decision to invest the engineering hours, the design and test infrastructure, and the initial fabrication costs into a new design is never taken lightly. However, the lack of a clear forecast for both new technologies and new application domains means that this investment now involves *significant* new risks. The new focus on big data and deep learning applications [10, 45], the threat of the end of Moore's Law [51], and the emergence of new chip technologies [27, 40], new memory technologies [24, 30, 48] and new computing devices and paradigms [3, 11, 41] add up to a computing landscape wrought with uncertainty.

Dealing quantitatively with this uncertainty, and the risk it creates, requires both new concepts and new tools. Computer architects tend to focus on quantifying and optimizing performance metrics such as IPC, throughput, and power efficiency but generally fail to consider how *exposed* to risk a class of designs might be. As we explore in this paper, even with fairly simple assumptions these new trade-offs can lead to surprising relationships. The end goal of this new line of work is to help find designs more robust to the impacts of uncertainty than performance-only-optimal designs while still maintaining very strong performance in the common case.

While the true risks a company takes on when attempting to bring a new architecture to market vary from complex partnerships, to infringing intellectual property, to marketing missteps, in this paper we concentrate on ***architectural risk***. Architectural risk, intuitively, is the degree to which the *performance* of a design is fragile in the face of *unknowns*. In many industrial settings high level design decisions are made at the level of spreadsheets and other high-level analytical models or data points drawn from past experience. Most do not consider the *uncertainty* in the assumptions being made nor the fragility of the decisions with respect to those uncertainties. Here we concentrate on such analytical models of architecture; however, a significant confounding factor (for both academics wishing to study this problem and industry professionals wishing for good tools to exist) is that data for such models in this space is very closely guarded.

For a technique to be successful it must be able to *extract* useful models of uncertainty from the limited data points available to each manufacturer. For example, given some example data points relating the performance of a set of designs to the amount of resources they consume, one should be able to say something about the **impact of the uncertainty** underlying those data points **without assuming** they are drawn from a very specific distribution. We describe a technique by which fewer than 50 data points can be used (even assuming no a priori knowledge) to effectively estimate a host of established architecturally relevant distributions for the purpose of quantifying the impact of uncertainty. Building from this contribution we show how even seemingly straight-forward questions such as core selection can lead to some surprising new interactions. Being "risk unaware" can lead you to decisions that not only have less desirable distribution but can even be strictly sub-optimal even when only considering *expected* (i.e. average case)

MICRO-50, October 14–18, 2017, Cambridge, MA, USA

ACM ISBN 978-1-4503-4952-9/17/10…$15.00
https://doi.org/10.1145/3123939.3124541

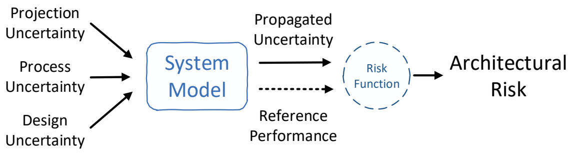


Figure 1: Relationship between uncertainty and architectural risk.

performance. Our methodology is encapsulated in a new tool, available on github[1], which performs this analysis automatically through a mix of statistics, symbolic algebra, and simulation.

To provide some background to this new risk-aware analysis, we begin in Section 2 with more precise definitions of uncertainty and risk in connection with computer architecture performance modeling. We also define an example system to which we will later apply our analysis. In Section 3 we describe how we use a hybrid analytic and sampling based framework to automatically propagate uncertainties to calculate architectural risk in high-level performance models. In Section 4 we then apply our framework to perform a risk-aware CMP design space exploration and examine a series of related questions. With this understanding of our analysis in place, we then discuss the relationship of this work to the other system modeling and performance evaluation efforts in Section 5. Finally, in Section 6, we conclude our analysis and discuss future directions for this new approach.

2 ARCHITECTURAL RISK

Uncertainty and risk are commonly used terms in many fields including economic and financial analytics, but it is important to be clear about their meaning here. In general risk is a *function* of the impact of *uncertainty* on the return of the system. In economics, uncertain events may include hikes in the price for raw materials, emergence of a serious competitor on the market, the loss of key personnel, and so on. Each event has some impact on the system (e.g. loss of sales, failure to recover payment). These impacts are then typically unified, by some function, to a common metric. Decisions are then made with an understanding of the risk in conjunction with expected outcome. These trade-offs can lead to the development of entirely new financial instruments.

In computer architecture our default metric is performance (or some combination of performance and energy). We typically talk about the expected performance without discussing the tail of the distribution of performance. Risk here maps one-to-one with the economic notion of risk [28]: we have uncertainty in what we know; those uncertainties manifest as changes in the performance of the system; the impact of those changes can then be quantified as risk. Figure 1 captures this idea graphically.

For example, in Amdahl's Law, the "return" is the speedup over performance of unit core. The inputs are f, which is the parallelizable portion of the program, and s, which is the speedup over the parallelizable code. When considering uncertainty in this case, the "uncertain events" are unexpected values in f and s. The probability of these unexpected events can be modeled by some underlying distribution. The cost in terms of performance variations associated with the unexpected input values can be depicted as some cost function (or risk function) C.

Definition. In this work, we define **architectural risk** in Equation 1 and 2.

$$R_e = C(P_e, \hat{P}), \quad P_e < \hat{P} \tag{1}$$

$$ArchR = \frac{\sum_{e \in E_1 \times E_2 \times \ldots \times E_n} R_e}{\|E_1 \times E_2 \times \ldots \times E_n\|} \tag{2}$$

The above equation 1 captures the architectural risk under the impact of some unexpected event e. $\hat{P}$ is the reference performance (or target performance of the design). If the reference performance is guaranteed to be achieved there is defined to be no risk. P_e is the real system performance under the impact of event e. C is the cost function (or risk function) and is usually subjective to the system designer or project manager. One might be interested in the probability of any unexpected event happening and thus defining a step risk function; Another might be interested in some certain events and their impacts and thus defining a piece-wise risk function; A third may only be interested in the monetary loss due to performance difference (e.g. less performant chips may be binned and sold at lower prices) and thus defining a mapping between performance and dollars. Equation 2 aggregates the risk captured by Equation 1 and takes the average across all possible event combinations, where E_1 through E_n are the sets of unexpected events for each type i of uncertainty. The performance P in the definition is not limited to execution time but is a broad term and can be any metric depending on what the system model is trying to evaluate.

We propose that computer architectures are exposed to three major sources of uncertainties: projection uncertainty, process uncertainty, and design uncertainty; most of which can be tracked at different levels of the system stack.

Projection uncertainty comes from assumptions one has to make about the future. At the application level, a system design may target a specific set of applications, but those target applications may shift or change based on our understanding of the problem or new optimization techniques. We often implicitly estimate future workload behavior with measurements of existing workloads. At the device level, systems may target underlying technologies still working their way out of the research labs. The performance of these future technologies is predicted by their physical models and there is usually some degree of uncertainty on how well they perform.

Process uncertainty comes from the manufacturing process itself. While semiconductor manufacturing is an incredibly precise process, when the probability of any fault is integrated over billions of transistors we are left with a distribution of devices. Some will work exceedingly well, others will under perform, while still others will fail to work at all. However, unlike projection uncertainty, under process uncertainty each chip is a new "roll of the dice".

Design uncertainty comes from the hardware design process itself. Components (e.g. cores and accelerators) with unresolved critical errors or introducing significant security vulnerabilities may be prevented (through a variety of means) from being accessible in an initial roll outs of a product. This class of uncertainty is

[1] https://github.com/UCSBarchlab/Archrisk.git.

a growing concern in the more heterogeneous and accelerator-dominated architectural design regime we are now faced with, and mechanisms for "partitioning out" features is an increasingly common practice.

Note that there is a *philosophical* distinction between the uncertainties we discussed above (leading to architectural risk) and the *inaccuracy* of an analytic model or simulator of a real system. It is possible to reduce the measurement inaccuracy with better modelling, more comprehensive workloads, increasingly detailed simulation, etc. — but in the end measurement inaccuracy *could* be eliminated given enough resources. However, even if one had infinite resources, the uncertainties we proposed above will still exist. It is much harder (or even impossible) to remove such uncertainties without a fundamentally new understanding of the future[2]. From a more practical standpoint, measurement error and the techniques one may use to reduce it [15, 34, 47], might either be grouped in with projection error or stand alone as a fourth category. However, we do not explore that trade-off in this work.

Importantly, it does not take many of the above interacting forms of uncertainty to make the complexity of such interactions impossible to intuit. A successful system addressing this issue must meet the following constraints:

1. it needs to work well on the types of uncertainty present in architectural analysis as described above
2. it needs to require as few as possible assumptions about the distribution governing the behaviors observed for each aspect of uncertainty examined
3. it needs to work with the very sparse amounts of data available to characterize such distributions; and
4. it needs to be automated to a degree that architects are not bogged down with the algebraic management and equation solving in exploring these design spaces.

2.1 An Example System Under Analysis

While there are many times that architects make analytic estimates of system performance, one of the most well studied is the heterogeneous core selection problem of Hill and Marty [23] as described succinctly in Table 1. Under this model one chooses the best performing core design to execute the serial code (Equation 6) and uses the aggregated performance of all cores to execute the parallel code (Equation 7). Pollack's Rule [6] is used to model core performance as a function of resources consumed (Equation 9). Designs are bounded by the total area/resource available on chip (Equation 10). In addition to these classic assumptions, we also take communication overhead among different cores into account, denoted as c in Equation 4, which is some fraction of the sequential workload. The amount of communication overhead is proportional to the total number of cores on chip (Equation 8). This overhead is extensively studied in [53] and can be setup/tear down time for the parallel computation, synchronization during parallel execution, or any other overhead introduced along with parallelization.

Table 1: Closed form model of performance.

$$Speedup = \frac{1}{T_{sequential} + T_{parallel}} \tag{3}$$

$$T_{sequential} = \frac{1 - f + c \times N_{core}}{P_{serial}} \tag{4}$$

$$T_{parallel} = \frac{f}{P_{parallel}} \tag{5}$$

$$P_{serial} = \max\{P_{core_i} \mid N_{core_i} > 0\} \tag{6}$$

$$P_{parallel} = \sum_{i \in core\_types} N_{core_i} \times P_{core_i} \tag{7}$$

$$N_{core} = \sum_{i \in core\_types} N_{core_i} \tag{8}$$

$$P_{core_i} = \sqrt{A_{core_i}} \tag{9}$$

$$A_{total} = \sum_{i \in core\_types} N_{core_i} \times A_{core_i} \tag{10}$$

Although real processor design is far more complicated, we show that even this simple model is significantly confounded by the introduction of uncertainty. Uncertainty in the inputs, even here, results in surprising outcomes (as detailed more in Section 4) and demonstrates the importance of new techniques to support this reasoning more rigorously. Our framework is by no means limited to these sets of equations only but can be applied to evaluations of different architectures including accelerators [2], different optimization objectives like power efficiency [52], or linked to more detailed simulators [9, 38]. As more parameters are added and more complex models are required this should make our proposed approach strictly more valuable and intuition even less reliable.

2.2 Uncertainties in our Example System

There are a total of five types of uncertainties that might be considered under the above model of a system. Uncertainties in target application behavior impact f and c (a future application running on the system might have a different level of parallelism and/or a different unit communication overhead than the benchmarks used to measure the system performance during design). Uncertainties in process/manufacturing can affect both P_{core_i} (different core instances may end up with varying performance properties due to intra-die variation) and N_{core_i} (due to fabrication defects impacting yield). Uncertainties in design may also have an effect on P_{core_i} in that, upon a design bug or failure, cores of that design might not work at all.

Each of these uncertainties is a complex thing to understand. For a technique to be useful it must *not* be highly sensitive to the assumptions about the distributions governing these unknowns. Often times we may have only a few tens of data points from which we can *infer* an underlying distribution. Later, in Section 3 we will describe exactly how this can be done in an automated way using a reversal of the classic power transform, but to evaluate the effectiveness of this approach we need *hidden reference models* to serve as a ground truth.

Pulling from the extensive literature on variation, yield, and program behavior, Table 2 summaries the hidden "ground truth". Our technique will attempt to capture the important aspects of these analytically from a few samples and no knowledge of the

[2] This is the difference between "aleatoric" and "epistemic" uncertainty.

Table 2: Hidden uncertainty models.

$$f \sim \frac{Binomial(M,p)}{M} \tag{11}$$

$$c \sim \frac{Binomial(M,p)}{M} \tag{12}$$

$$N_{core_i} \sim Binomial(M, yield_{core_i}) \tag{13}$$

$$P_{core_i} \sim Bernoulli(p) \times LogNormal(\mu, \sigma) \tag{14}$$

$$yield_{core_i} = (1 + \frac{d \times A_{core_i}}{\alpha})^{-\alpha} \tag{15}$$

equations themselves. For N_{core_i}, i.e. the number of cores that are actually working, from its physical definition, we model it by a binomial distribution ranging $[0, N]$. N is the designed number of $core_i$ but each with only some probability of functioning properly taking after chip yield rate [14] (Equation 13, 15). P_{core_i} is modelled by the product of a LogNormal distribution ranging $(0, \infty]$ and a Bernoulli distribution with probability p of taking value 1 (Equation 14). We model core performance in such a way that it is exposed to two types of uncertainty: design uncertainty and fabrication uncertainty. Although design uncertainties can result in many consequences from degradation of performance to reduction of reliability and so on [13], we only consider severe design bugs that will lead to complete post-silicon failure of the component here. This type of uncertainty naturally follows a Bernoulli distribution by its definition, i.e. the component is either working or not. The probability of failure is set based on reported statistics [18]. Fabrication process uncertainty is modeled by the LogNormal part of the distribution. When the design works, the actual performance of each core also varies as a result of the fabrication process, leaving a Gaussian-like distribution on the positive domain [39, 43, 44]. For the LogNormal part, the location μ and scale σ is computed such that the mean performance follows Pollack's Rule (Equation 9) and the variance meets our desired level in experiments. For f and c, we use a normalized binomial distribution to model their uncertainties (Equation 11, 12). This distribution fits well to the characterizing data for the PARSEC benchmarks [5]. Their range is bound to be $[0, 1]$, while p is set to the mean value. M, which is needed to construct the Binomial distribution, is computed to satisfy the level of variance we desire in simulation.

3 RISK-AWARE ANALYSIS FRAMEWORK

At a high level our technique requires two inputs. First, an executable architecture model under analysis (that relates a set of mutually dependent parameters capturing constraints and dependent variables to optimize). Second, a set of data points drawn from the distribution whose uncertainty you wish to consider (e.g. a set of points relating core resources and performance). If we can pull a few points from the distributions governing these models and then, without any knowledge of the model itself (just the values of the specific samples drawn from it), construct a new model that has close to the same optimization utility as the ground truth, it gives us confidence that this will be useful when applied to a specific set of trade secret data by a manufacturer[3]. If the architecture model can be described as a set of mutually dependent closed-form functions and the uncertainty in distributional representations (e.g. a large set of samples or sampling functions), our tool can symbolically combine and partially solve the closed-form equations. From this form, it can then inject and propagate uncertainty through to the final responsive metrics so that risk can be calculated. First, however, we need a way to extract (or approximate) such distributional representations of architecture uncertainty from a few initial samples.

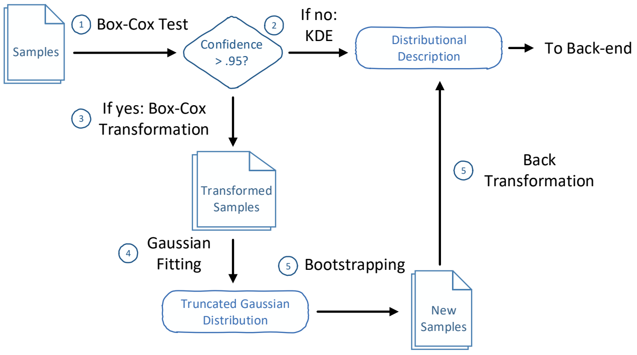


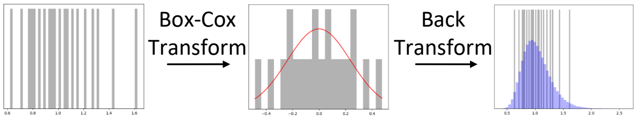
Figure 2: Uncertainty modeling.

3.1 Architecture Uncertainty Model Extraction

Such approximation is done by a two-phase method shown in Figure 2. We first test if the data set can be transformed to normality through the Box-Cox testing [8] in Step ①. If it cannot pass the test (a rare case in practice), we apply Kernel Density Estimation (KDE) methods [46] directly to the data set. These methods find a best-fit non-parametric distribution in Step ② and use its sampling function to facilitate uncertainty propagation. Otherwise, we transform the data set to normality using Box-Cox transformation in Step ③, re-sample (bootstrapping) from the Gaussian distribution in the transformed domain in Step ④, and back-transform the samples to the original domain. Finally we reconstruct the distribution in original domain in Step ⑤ to approximate the *hidden ground truth*. Although not as accurate as the best-fit non-parametric KDE, such bootstrapping method enables us to hand tune the desired uncertainty level in each variable and hence be able to explore the trend as input uncertainties scale. We use the bootstrapping method in our experiment to study the scaling behavior and to examine the accuracy of such approximation, but in practice, a non-parametric distribution is at most times sufficient to facilitate accurate uncertainty analysis.

Figure 3 gives an example of the bootstrapping process. Figure 3a shows the histogram generated from initial samples (the samples are taken from a log normal distribution). Figure 3b shows the histogram after transformation and the fitted Gaussian distribution

[3] Of course with arbitrarily complex "ground truth" such a trick is impossible. In the most general case this problem reduces to one of function inversion, which we know from cryptography can be hard, but luckily the distributions that typically govern the physical and program properties an architect would actually care about are ones that we find are highly amenable to this technique.



a) Initial samples.

b) Transformed samples and fitted gaussian.

c) Bootstrapped distribution.

Figure 3: An example bootstrapping process.

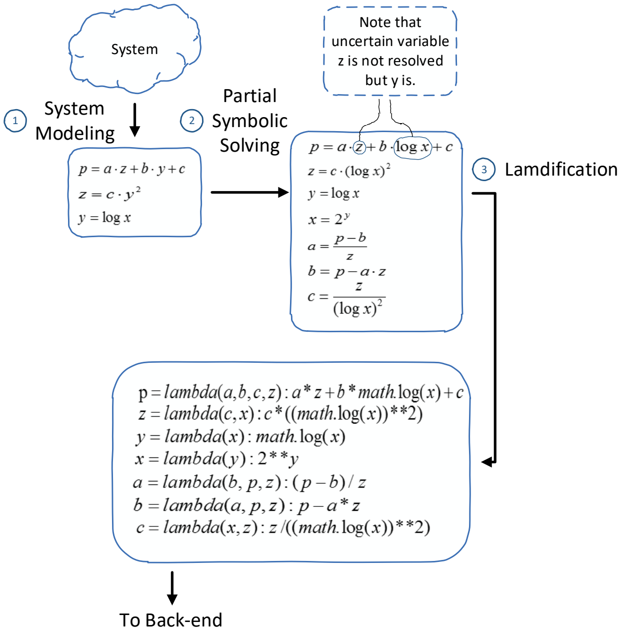

System
1 System Modeling
$p = a \cdot z + b \cdot y + c$
$z = c \cdot y^2$
$y = \log x$
2 Partial Symbolic Solving
Note that uncertain variable z is not resolved but y is.
$p = a \cdot z + b \cdot \log x + c$
$z = c \cdot (\log x)^2$
$y = \log x$
$x = 2^y$
$a = \frac{p-b}{z}$
$b = p - a \cdot z$
$c = \frac{z}{(\log x)^2}$
3 Lamdification
p = lambda(a,b,c,z): a*z+b*math.log(x)+c
z = lambda(c,x): c*((math.log(x))**2)
y = lambda(x): math.log(x)
x = lambda(y): 2**y
a = lambda(b, p, z): (p − b) / z
b = lambda(a, p, z): p − a*z
c = lambda(x, z): z /((math.log(x))**2)
To Back-end

Figure 4: System modeling and symbolic execution.

in the transformed domain. If the initial samples can pass box-cox test, we are guaranteed to find a Gaussian distribution that fits the transformed samples. Figure 3c shows the bootstrapped distribution after back transformation to the original domain laid on top of the original samples.

The output of this uncertainty modeling phase is a set of uncertain variables along with their distributional descriptions to facilitate uncertainty injection and propagation.

3.2 Model Transformation and Execution

In Figure 4, we present an overview with a simple example of how the front-end modeling and symbolic execution works.

The first step ① is system modeling which builds mutually dependent equations described in Section 2.1. The framework then performs the following operations. ② Each variable including uncertain ones is then treated as a symbolic entity and the plain string-formatted equations are passed to symbolic execution [50]. The result is a set of algebraic equivalent equations with each symbol sitting on the left-hand side in one equation. We break the solving into steps and only resolve variables that are not uncertain, and any uncertain variable on the right-hand side in the equations is kept unresolved in its original form. This is to support uncertainty injection and propagation later. ③ The partially solved equations are then converted into callable lambda functions. We also enforce a fixed argument ordering in the lamdification process.

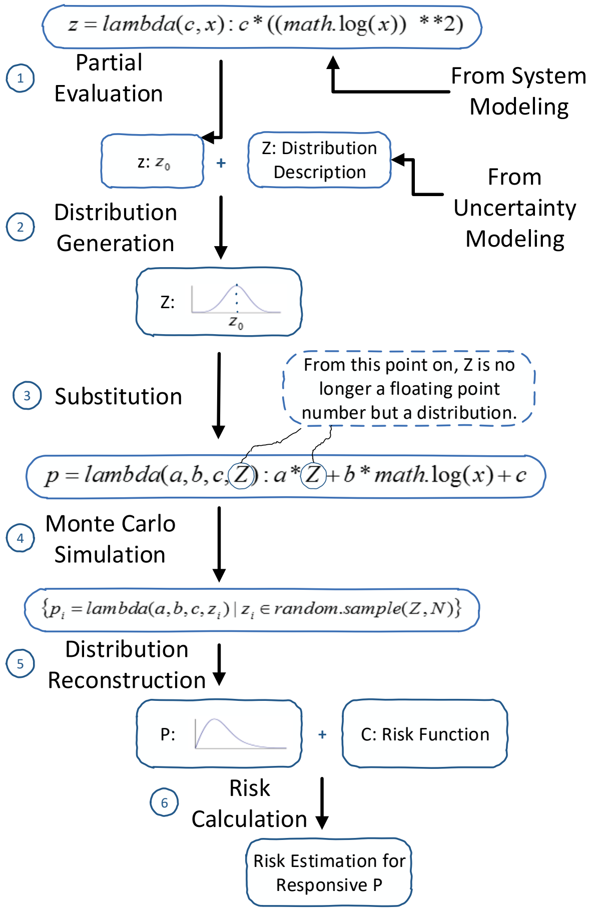

z = lambda(c, x): c*((math.log(x)) **2)
1 Partial Evaluation
From System Modeling
z: z_0
+
Z: Distribution Description
From Uncertainty Modeling
2 Distribution Generation
Z:
z_0
3 Substitution
From this point on, Z is no longer a floating point number but a distribution.
p = lambda(a, b, c, Z): a*Z + b*math.log(x) + c
4 Monte Carlo Simulation
$\{p_i = lambda(a,b,c,z_i) \mid z_i \in random.sample(Z,N)\}$
5 Distribution Reconstruction
P:
+
C: Risk Function
6 Risk Calculation
Risk Estimation for Responsive P

Figure 5: Back-end uncertainty injection and propagation.

At the end of this process we have a set of callable functions that are provided to the back-end for numerical computation.

3.3 Uncertainty Injection and Propagation

Figure 5 shows the back-end uncertainty injection and propagation process.

Given the set of functions and uncertain variables, the framework back-end then proceeds with the following steps. ① Each uncertain variable gets evaluated first as long as there are no uncertain variables in the right-hand side of their solutions. The values of certain inputs, like c and x in this case, should be provided by the system designer. ② The framework then generates distributions for all the uncertain variables. Based on their descriptions, the distribution can be generated alone or together with the evaluated value. In this example, the distribution of Z takes the evaluated value z_0 as its mean. ③ All uncertain variables appear in the argument list of the lambda functions are then replaced by the corresponding distributions. At the completion of this stage we have injected the desired uncertainties into the solved system model. ④ Each function containing distributions or random variables in the argument list is then evaluated by Latin-hypercube Monte Carlo simulation N times [31]. The result of the Monte Carlo simulation is a set of values for each responsive variable in the system. ⑤ We then reconstruct a distribution from the set of values for each responsive variable. At this point, the uncertainties in the inputs are propagated into the distribution properties of the responsive variables. There are systematic errors associated with the Monte Carlo simulation but, in our experiments, we keep N sufficiently large (we use $N = 10,000$) to keep the errors negligible. ⑥ Finally, we calculate risk based on the distribution of the responsive variable and the risk function provided.

4 ANALYSIS OF RISK ON CORE SELECTION

Building from the framework and ground truth models discussed above, we now carry out an analysis on the classic Hill and Marty core selection design problem and demonstrate how performance and risk interrelate and can even be co-managed. The design question is essentially: what cores and how many of them should we put on a CMP in the face of uncertainties? We bound the design space to populate by constraining the total chip size (or resources) to be 256 units and consider the full spectrum of designs (rather than just one big-core coupled with many tiny cores). Specifically we ask the following questions which will be answered by a series of implications we draw from our experiment results:

- How does uncertainty manifest and interact in the system?
- How sensitive is CMP performance in the face of uncertainty?
- Is the conventional risk-oblivious design optimal in terms of architectural risk? Furthermore, is it still optimal even in terms of expected performance?
- When is there a trade-off space between architectural risk and expected performance? What does the trade-off space look like?
- What configuration/design is favored when one considers risk?

After exploring the design space with ground truth distributions, we show that our approximation method still leads to optimal or near-optimal designs from only a handful of samples. Such partial information about the underlying uncertainty distributions is often the case in early modeling and design cycles. While we primarily consider architectural risk in the form of performance, we further demonstrate the use of this analysis in evaluating *monetary* risk function using both ground truth distributions and the approximations.

4.1 Uncertainty Manifestation

Experiment Setup. In order to answer the question of how uncertainty manifests in the output, we inject a total of five types of uncertainties into the four input variables of our model. The uncertainties we inject are application characteristics uncertainties in f and c, process variation in P_{core_i} and N_{core_i} as well as design uncertainty in P_{core_i}.

Table 3 describes how much uncertainty we inject. With $\sigma = 0$, the inputs are certain values ($\hat{f}$, $\hat{c}$, $\hat{P}_{core_i}$ and $\hat{N}_{core_i}$), the resulting performance is the conventional "certain" result without propagated uncertainty. With $\sigma > 0$, f is centered on $\hat{f}$ with a standard deviation of $\sigma \cdot (1 - \hat{f})$, such that the standard deviation of f is kept small enough that f only varies at the least significant digit of $\hat{f}$ and does not completely change the application to another category (again, we are being conservative here and a larger uncertainty will make the risk even more important). Similarly, c is centered on $\hat{c}$ but with a standard deviation of $\sigma \cdot \hat{c}$ as c in itself is very close to 0. As for core performance P_{core_i}, the performance uncertainty is centered around $\hat{P}_{core_i}$ and the standard deviation is set to be $\sigma \cdot \hat{P}_{core_i}$. There are two special types of uncertainty that are not centered around the corresponding certain values. The fabrication uncertainty is added when we consider that each core has a probability of failure (not functioning properly). We keep yield rate for each type of core evaluated constant throughout the computation. Table 4 lists the yield rates computed using Poisson chip yield model [29]. Note that yield rate is not dependent on σ but only on core size. Design uncertainty is modeled by a Bernoulli with probability $\sigma \cdot \gamma$, and we set the intrinsic probability γ based on an estimation of existing data [18].

Table 3: Injected uncertainties.

Input	Certain Value	Uncertain Value	
		Mean	Std
f	$\hat{f}$	$\hat{f}$	$\sigma \cdot (1 - \hat{f})$
c	$\hat{c}$	$\hat{c}$	$\sigma \cdot \hat{c}$
P_{core_i}	$\hat{P}_i$	$\hat{P}_i$	$\sigma \cdot \hat{P}_i$
Fabric	$N_{core_i} \sim Binomial(\hat{N}_{core_i}, \quad yield_{core_i})$		
Design	$P_{core_i} \sim Bernoulli(\sigma \cdot \gamma) \times P_{core_i}$		

Table 4: Yield rates.

core size	8	16	32	64	128
yield	98%	96%	92%	85%	75%

We hand-tune the injected uncertainty level σ from 0 to 1 for four different categories of applications and three different architecture designs. The four applications are characterized by different values of parallelizable portion $\hat{f}$ and unit communication overhead $\hat{c}$. We call $\hat{f} = 0.999$ high parallelism (HP) and $\hat{f} = 0.9$ low parallelism (LP). We refer to $\hat{c} = 0.001$ as low communication cost (LC) and $\hat{c} = 0.01$ as high communication cost (HC). To see the impact we consider three example designs which are symmetric (32x8) and asymmetric (1x128 + 16x8) designs from Hill and Marty's setting as well as an extended full heterogeneous architecture in which 5 types of cores are present.

Results and Discussion. In Figure 6, we show examples on the resulting performance distribution. The mean of the performance distribution should be the expected performance under uncertainties and its standard deviation (or variance) measures how much

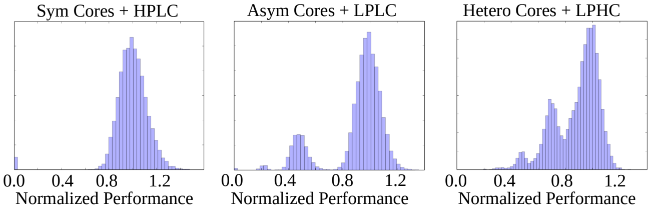


Figure 6: Performance distribution under uncertainties. "Sym Cores" stands for a configuration of 32x8 (32 small cores of size 8), "Asym Cores" stands for a configuration of 1x128 + 16x8 (one large core of size 128 and 16 small cores of size 8), and "Hetero Cores" stands for a full heterogeneous configuration of 2x8 + 1x16 + 1x32 + 1x64 + 1x128.

each chip differs from one another in terms of performance. The shape of the distribution also matters in that it relates to how much architectural risk a design is exposed to. Based on our architectural risk definition in Section 2, given a reference performance (or performance goal), risk is essentially a weighted area under the curve to the left of the performance goal. Looking at the figure, an important observation is that the resulting performance distribution is very irregular even with our simple and regular input distributions.

In Figure 7, we present how the expected performance under the impact of input uncertainties behaves as σ increases. In most cases, input uncertainties lead to worse expected performance compared to its "certain" version, while in some cases, a strong uncertainty, especially on f and c, can lead to better expected performance because of the asymmetric impact f and c have on performance. For example, in the asymmetric architecture with an application of $\hat{f} = 0.9$ and $\hat{c} = 0.001$, an $f = 0.9 + 0.1$ will raise the performance from 36.38 to 37.93 (an 1.55 increase) while an $f = 0.9 - 0.1$ lowers performance to 34.96 (an 1.42 decrease). In another word, given the same deviation from $\hat{f}$, the impact of a higher f is greater than the impact of a lower f, resulting in a better expected performance. From an architectural point of view, this asymmetric impact results from the fact that asymmetric design often fits applications with more inherent parallelism better.

If we compare the performance across different applications for the symmetric design, we can see that when f gets smaller (uncertainty on f gets larger) and c unchanged, the impact of uncertainty on f becomes dominant (compare the first and third figure on the first row), while when c gets larger (uncertainty on c grows) and f unchanged, the impact of uncertainty on c becomes dominant (compare the third and fourth figure on the first row). This observation holds for all three designs and meets our expectation that the uncertainty on the dominating characteristic of the application has a greater impact on the output.

If we instead pick an application and compare across all three designs, we can tell that the performance "boost" brought by the asymmetry on f and c diminishes as the chip becomes more heterogeneous. The same example math of $f = 0.9 + 0.1$ and $f = 0.9 - 0.1$ suffices to show that the asymmetry is barely observable in the very heterogeneous design. In another word, the more heterogeneous the chip is, the less sensitive to application uncertainty it is.

Another observation begins to surface in this comparison study. The overall impact of uncertainty on core performance P_{core_i} behaves differently for chips of different heterogeneity and heterogeneous chips are generally more sensitive to architectural uncertainties. In the symmetric case, all cores are of the same size and the uncertainty in the performance of each core cancels out one another, leaving the result performance unchanged. While in the asymmetric case, the collection of the small cores still behave in such a way, but when the big core has a degraded performance, it has a much larger impact on performance that cannot be offset by the other smaller cores. In the very heterogeneous design, however, the collective behavior of the cores contributes to a better expected performance. This explains why the impact of architecture uncertainty grows when the architecture design becomes more heterogeneous.

Figure 8 gives three examples of the uncertainty in the output of the model as the input uncertainties vary. In general, uncertainty in performance grows as the input σ increases. This follows our intuition that the more uncertain the input is, the more uncertain the output should be. Most of the input uncertainties propagate through the model sub-linearly, indicating some tolerance for uncertainty the model exhibits. We also compare across the designs for the same application, and find out that the more heterogeneous the chip is, the more uncertainty-tolerant it is.

A counter-intuitive fact is that the composite uncertainty in the output is not simply an accumulation of all the input uncertainties. In fact, the uncertainties are not even additive. To better demonstrate this behavior, we conduct a series of experiments by removing one type of uncertainty at a time. In Figure 9, we use the asymmetric design as an example to show that the output uncertainty sometimes rises when there is less uncertainty in some of the inputs. This happens because uncertainty has two possible effects on the output performance: it may contribute to a better performance or it may lead to a worse performance. And different components of the system (different inputs) respond to uncertainty with different magnitudes, as well as directions (better or worse). When combined, different uncertain inputs may enhance each other, leaving performance shifted more from the expected value, while in other situations they may attenuate each other reducing shifts in performance.

In summary, regarding how uncertainties propagate and how sensitive CMPs to these input uncertainties, we have the following implications.

Implication 1. Uncertainties propagate through the model with non-intuitive interaction, the resulting performance distribution is beyond what a simple "back-of-the-envelope" estimation can reveal.

Implication 2. The more heterogeneous the chip is, the less sensitive its expected performance to application uncertainty, but the more sensitive its expected performance to architecture uncertainty.

Implication 3. The more heterogeneous the chip is, the more tolerant/robust its performance is to input uncertainties.

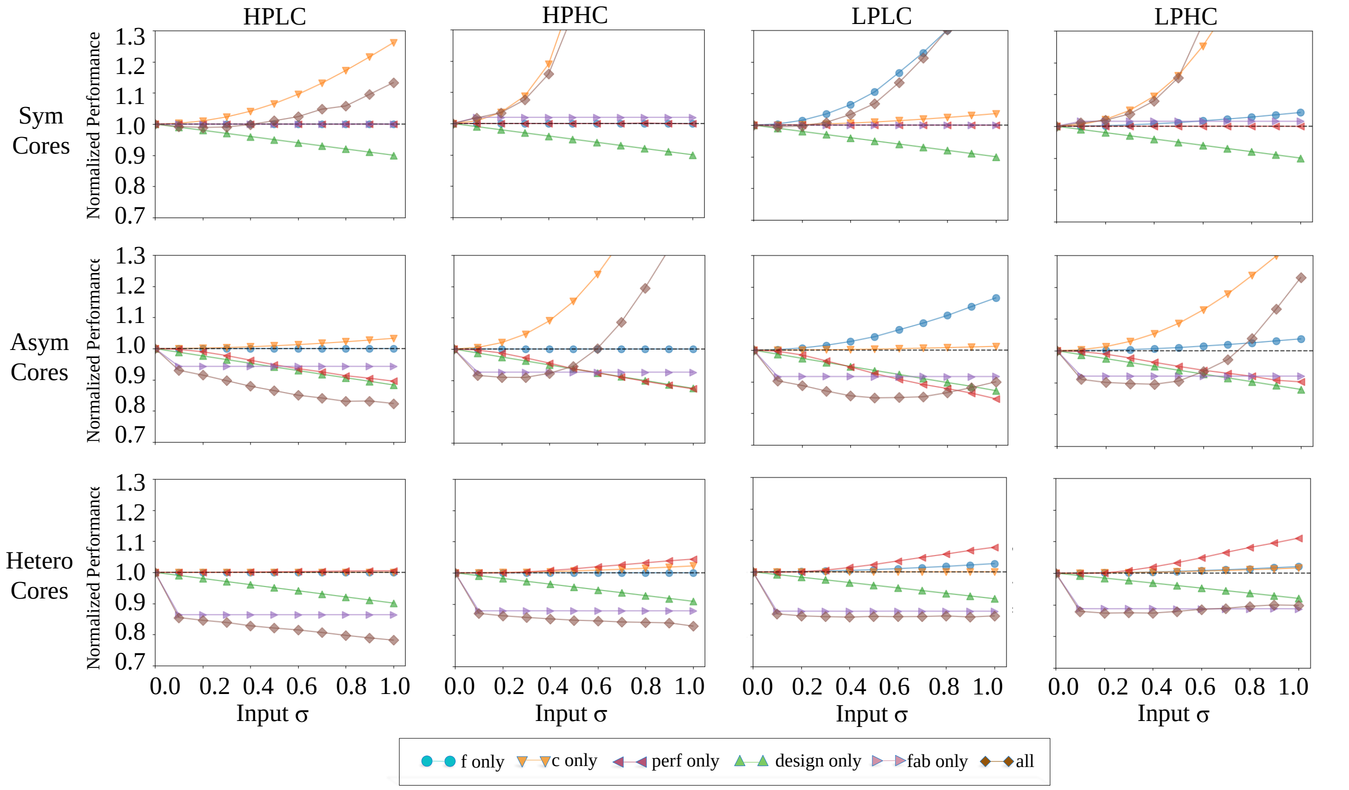


Figure 7: Uncertainty manifestation on output performance. Legend indicates which type of uncertainty is under consideration, and expected performance is normalized to risk-unaware performance.

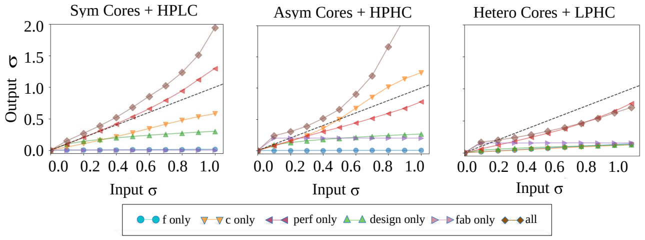


Figure 8: Example uncertainty manifestation on output uncertainty. Legend indicates which type of uncertainty is under consideration and standard deviation is normalized to risk-unaware performance.

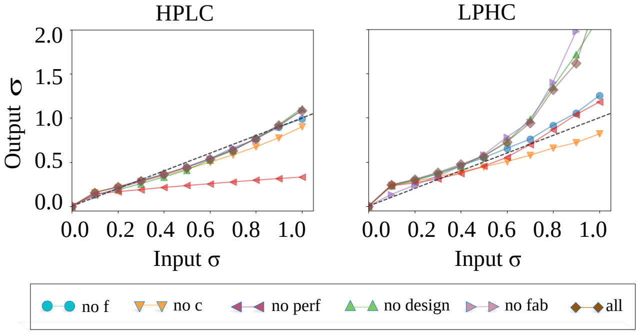


Figure 9: Non-accumulative output uncertainty for asymmetric architecture. Legend indicates which type of uncertainty is excluded and all means all types of uncertainty are considered.

4.2 Impact on Design

Given the complexity of propagated uncertainties in the three designs above, we now expand our search space and explore how uncertainty and risk may impact design decisions in the uncertainty wrought design space.

Experiment Setup. The injected uncertainties are identical to the setups in Section 4.1. We exhaustively enumerate all valid design options. Each valid design option is a configuration taking up 256 on-chip resource units (a total size of 256) with a combination of different cores, each of which having a size of power of two ranging from 8 to 256 (e.g. a valid design can be 32 cores of size 8, 1 core of size 256 or 16 cores of size 8 plus 1 core of size 128). Some combination does not consume all the on-chip resources, and in those cases, we group all resource left into one additional core (e.g. 8 cores of size 8 plus one core of size 192 is also valid). We also use a quadratic risk function in this exploration. In other words risk is the sum square of the performance loss below some reference due

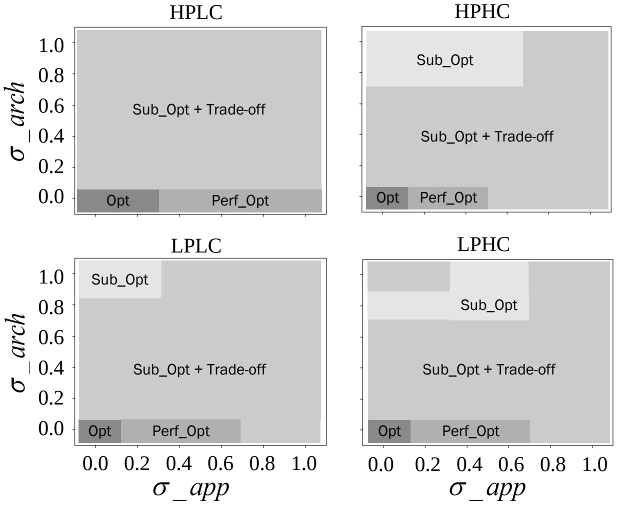


Figure 10: Impact on design of application uncertainties and architecture uncertainties. σ_app **is the uncertainties in** f **and** c **and** σ_arch **is the uncertainties in** P_{core_i} **and** N_{core_i}**. "Opt" means conventional design is optimal in both expected performance and risk. "Perf Opt Only" means conventional design is only optimal in terms of expected performance. "Sub-opt" means conventional design is strictly sub-optimal in both expected performance and risk. "Sub-opt + Tradeoff" means not only conventional design is sub-optimal, there is also a trade-off space between performance-optimal design and risk-optimal design.**

to uncertainty for all eventualities. The idea is that performance well below expectation is much worse than performance just below expectation (similar to minimizing sum square error).

Results and Discussion. We present results considering both architecture uncertainties (process uncertainty and design uncertainty) and application uncertainties (uncertainty in f and c) in Figure 10. The conventional performance-optimizing uncertainty-oblivious design is at most times not the optimal choice not only in terms of risk but also, very counter-intuitively, even in terms of expected performance. If we take a look at the architecture uncertainties (σ_arch on the y-axis) and application uncertainties (σ_app on the x-axis) respectively, we can tell that architecture uncertainties usually impose a larger impact on design decisions. In all four types of applications, the optimal design shifting along the y-axis occurs even when there is only 20% uncertainty or less. Meanwhile, the application uncertainties shift the best design at a slower pace. If we consider application uncertainty alone (with a fixed architecture uncertainty), we can see that the application uncertainties shift the risk-optimal design easily but the performance optimality shifts only when application uncertainty is abundant. In one case where parallelism is high and communication overhead is low, application uncertainty does not shift the performance optimal design at all even at a level of 100% of the mean.

An example trade-off space with LPHC application between performance-optimal design and risk-optimal design is shown in Figure 11. To help readability and understanding, we do not include all curves at every input uncertainty level in Figure 11a, but the trends of other curves are very similar to the examples we show in the figure. We can tell that the amount of input uncertainties shifts the possible outcomes of all designs in the performance-risk space and determines how the trade-off space look like. In most cases, there exists a trade-off space between the performance-optimal design and the risk-optimal design. Taking the curve marked in Figure 11a as an example, one can mitigate almost 60% of risk at the cost of less than 3% performance. Figure 11b zooms in on the example curve and shows both the Pareto-optimal designs as well as the non-optimal designs with relatively strong expected performance (within 89% of the best expected performance) in the space. Figure 11c further zooms in on the two representative designs on the Pareto curve. We can see that having a more "concentrated" distribution around the performance goal helps bring down the architectural risk of the lower design, while the upper design has a wider distribution leaving a larger risk but better expected performance.

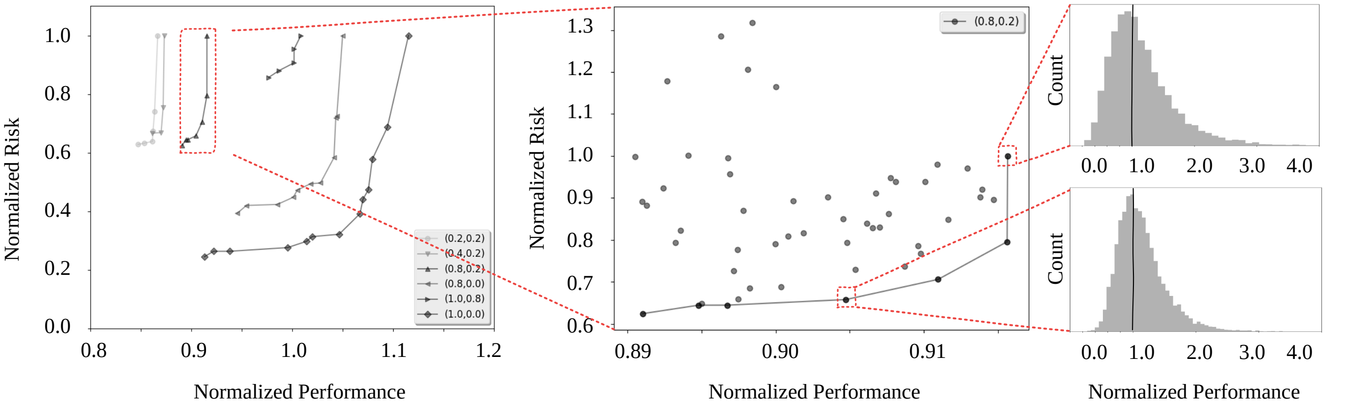
In summary, regarding how conventional design performs in the uncertainty wrought design space, we have the following implications.

Implication 4. Conventional architectural risk-oblivious design is at most times not the architectural risk optimal design when there's even moderate amount of input uncertainties. Conventional design is also oftentimes not even optimal in terms of expected performance, i.e. there is another core-configuration that yields better expected performance in the face of uncertainty.

Implication 5. Architecture uncertainties usually have a larger impact on design optimality while application uncertainties have a relatively smaller impact.

Implication 6. At most times there exists a trade-off space between performance-optimal design and risk-optimal design, and one can mitigate a good amount of risk at the cost of a relatively small performance degradation.

Next we explore what exactly the optimal core configurations are and what configurations are generally preferred in terms of both expected performance and architectural risk using results with LPHC as an example. Figure 12a gives the performance-optimal core configuration distributions in our search space. If we consider application uncertainty σ_app alone, when it gets larger, the histograms tend to grow towards the left edge and concentrate on a few selections. This "concentrating" trend means that *more asymmetric* configurations are generally favored when there is more application uncertainty. This trend results from the asymmetric impact we discussed in Section 4.1. A large core is needed to compensate the performance loss due to a lower f or higher c while the herd of small cores are better performing than a distribution of heterogeneous cores for parallel execution. However, when σ_arch gets larger, i.e. there are more architecture uncertainties involved, we can see that the histograms tend to spread out across different core sizes. This "spreading" trend indicates that *less asymmetric* configurations are preferred when there is more architecture uncertainty. This is due to the fact that mid-sized cores are chosen because the performances of cores have variations and a mid-sized core can step in during serial execution to compensate the performance loss due to a less performant large core. Another reason is that multiple cores of each type are chosen to fight the intra-die



a) Pareto curves at eifferent uncertainty level.

b) Zoom-in Pareto curve and non-Pareto points.

c) Zoom-in performance distribution.

Figure 11: Example trade-off space between performance-optimal design and risk-optimal design. Each point in the space is a performance distribution of a certain design (core-selection) at some input uncertainty level (denoted as a tuple of (*σ_app*, *σ_arch*)) with the LPHC application. Performance is normalized to that of the conventional case (risk-oblivious performance-optimal design). Risk is normalized to that of the performance-optimal design at each input uncertainty level.

process variation, leading to fewer core types on chip because of total area/resource constraint. These two counter-directional trends are the main reasons that the design space is very irregular and complicated.

Similarly in Figure 12b, we show the optimal core configuration for architectural risk optimal designs. Comparing with Figure 12a, we can tell that in most cases a “spread-out” or *more symmetric* configuration is needed to minimize architectural risk. However, the general trend when uncertainty grows is a blend of such “spreading” and those in Figure 12a and is very irregular and extremely hard, if not impossible, to intuit.

To sum up, regarding what configuration/design is in general more favorable, we have the following implications.

Implication 7. More symmetric configurations tend to minimize architectural risk while more asymmetric configurations tend to maximize expected performance in the face of uncertainties.

Implication 8. As application uncertainty gets larger, more asymmetric configurations are preferred, while as architecture uncertainty gets larger, more symmetric configurations are favored.

4.3 Approximating Uncertainty and Risk

In this section, we explore, when there is no a prior knowledge of the *hidden ground truth* distributions of the input uncertainties but merely a few samples drawn from them, how our approximation method performs.

Experiment Setup. We take only k samples from each of the distributional input uncertainties listed in Table 2. We then apply our bootstrapping method discussed in Section 3.1 on these samples to acquire approximations to the true uncertainty distributions. We then again conduct an exhaustive search through the design space using settings in Section 4.2 with the approximate uncertainties.

Results and Discussion. Figure 13 presents the approximation quality. Aside from some numerical fluctuations, the general trend is clearly showing that we can bound the error in terms of both expected performance and risk within 5% with 50 samples or even fewer. When the sample size exceeds 100, the quality of approximation is very stable and the approximation is very close to the hidden ground truth. We will further show the approximation quality with a concrete example in Section 4.4.

4.4 From Architectural Risk to Financial Risk

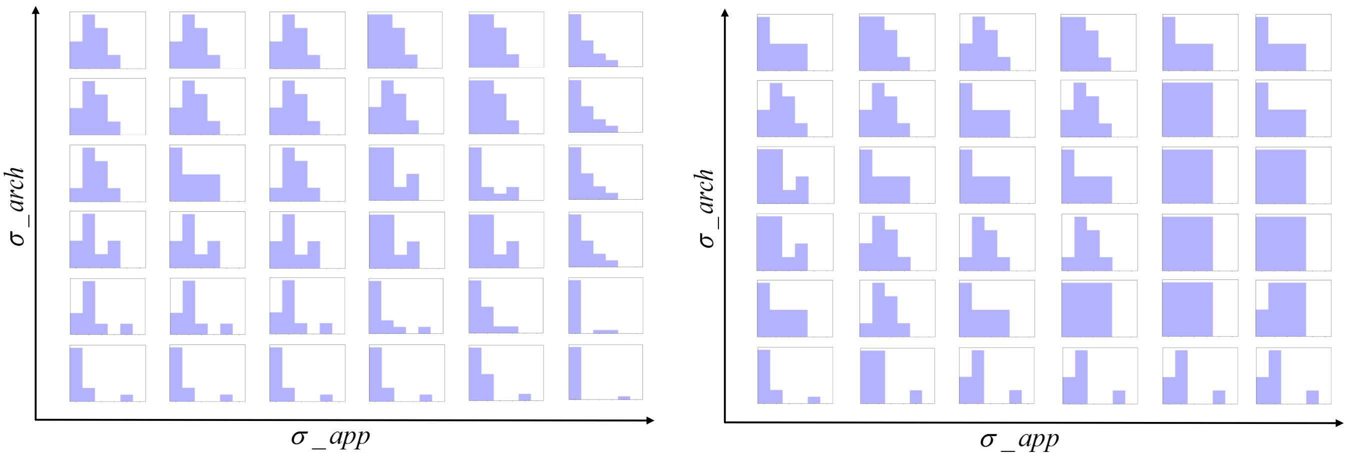
Now that we understand the impact of uncertainty and risk on core selection, we take a step back and re-examine the meaning of our risk function. The quadratic risk function used in above experiments may seem a little abstract when it comes to interpreting the meaning of its absolute value. One may ask the question of: *what is the cost in dollars if a sub-optimal design is chosen in the face of uncertainty?*

To answer this type of question, a different risk function which ties performance to a concrete dollar value in the market is needed. Here we show an example using a simple monetary mapping for normalized chip performance. Table 5 lists the relationship of normalized performance and dollars estimated from a publicly available CPU price list [25].

Table 5: Correlation between Normalized Chip Performance and Market Value.

Perf	< 0.6	[0.6, 0.8)	[0.8, 0.9)	[0.9, 1.0)	≥ 1.0
$	100	200	300	600	1000

The risk function is defined in Equation 16 to reveal risk in terms of dollar cost due to performance uncertainty. We consider all input



a) Expected performance optimal designs

b) Architectural risk optimal designs

Figure 12: Core configurations of optimal designs for LPHC. Each design is represented in a histogram of core distribution. Each bar in the histogram corresponds to the count for each type of core of which the size ranging 8, 16, 32, 64, 128, and 256 from left to right . *σ_app* and *σ_arch* both range from 0 to 1. Note that all designs are bounded by the same area constraint and the y-axis of each histogram is not normalized to better show the ratio between different types of cores.

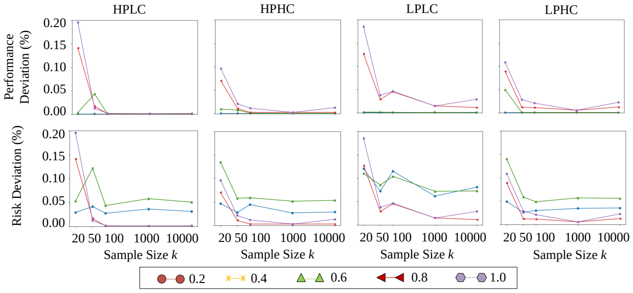


Figure 13: Quality of approximation. Different curves are averages taken for input uncertainties (both *σ_app* and *σ_arch*) less than a given threshold marked by corresponding legend.

uncertainties at $\sigma = 0.2$ with LPHC as an example and derive the performance distribution of a risk-unaware optimal design, a risk-aware optimal design with *hidden ground truth*, and an approximate risk-aware optimal design with sample size $k = 50$ in Figure 14. Based on our definition in Section 2, the architectural risk of the conventional design is \$348.53 which means \$348.53 per chip on average lost (compared to the price of a chip at performance 1.0) due to uncertainty with an average case performance of 0.95. For risk-aware optimal design with hidden ground truth, the architecture risk is \$293.64 with an average performance of 1.00. For the approximated risk-aware optimal design, the architectural risk is \$301.38 with an expected performance of 0.99. At this point, for this specific setting, we can answer that *\$47.15 per chip can be saved with even better expected chip performance using an approximate uncertainty analysis.*

$$C(P_e, \hat{P}) = \$(\hat{P}) - \$(P_e) \tag{16}$$

5 RELATED WORKS

5.1 Modeling and Design Space Exploration

As discussed earlier Hill and Marty extends Amdahl's Law to multi-core scenarios [23]. Altaf and Wood apply a similar analytical modeling to estimate performance of system with accelerators [2]. Esmaeilzadeh et. al. explore the power limit on mutli-core scaling using an extension of Hill and Marty's model [16]. This line of analytical research has led to many works [12, 21, 49, 52] focusing on different aspects of the system with different assumptions.

In addition to analytical models built purely from a high-level inspection of the system, there is also a class of research using empirical modeling which exploits statistical and machine learning techniques to examine uncertainty for the purpose of inferring a better system model [4, 9, 15, 22, 26, 32–38, 42]. Many of these techniques begin with parametrized models and then statistically fit the parameters. Alameldeen and Wood in particular examine the variability in multi-threaded workloads [1] and, by injecting random errors into the detailed simulator, develop a method to account for this type of uncertainty in simulation. Most of the these works address the problem of discovering good models and/or focus on minimizing statistical errors from those models. Rather than proposing a new model, or reducing errors introduced by existing ones, we instead argue for a design and modeling approach that embraces *extrinsic uncertainties* and provides tools for making good decisions in the face of the *associated risks*. Although we examine primarily an extension to the Hill and Marty model, our analysis and framework can also be used with more detailed or complicated

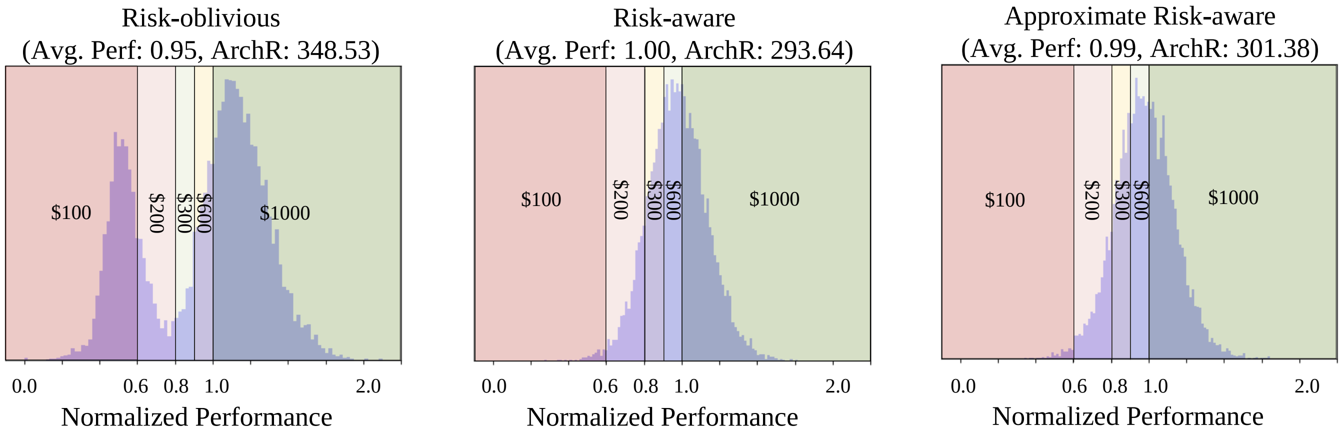


Figure 14: Binning of design results under uncertainties. A price binning based on Table 5 is laid on top of the performance distribution derived from each design with LPHC application at an input uncertainty of $\sigma\_app = \sigma\_arch = 0.2$. These designs correspond to the point in Figure 10 LPHC at coordinate (0.2, 0.2).

models as long as they can be expressed in, or approximated by, an interacting set of closed form equations.

5.2 Economical Thinking and Computer Architecture

This paper is certainly not the first to be inspired by economic thinking as applied to Computer Architecture. Bornholt et. al. deal with application-visible uncertain data at the programming language and runtime level [7]. Guevara et. al. combine market mechanism and resource allocation techniques to explore datacenter architectures [19]. Zahedi and Lee use game theory to study how to better allocate hardware resource in a cloud environment [54]. Fan et. al. also use game theory to handle power management in datacenters [17]. Guevara et. al. tackle the problem of runtime variation in datacenters and propose strategies to mitigate the risk of not meeting performance target [20]. These and other papers concentrate on the application of economic reasoning to better allocate resources in the face of competing interests, rather than examining the cost of extrinsic uncertainty on the high level design. Combining these lines of work would be an interesting area for future exploration.

6 CONCLUSION

We are living in interesting times for computer architecture — both from above by the applications, and below by the technology, we find ourselves pressed between many new uncertainties. While developing new computer system has always involved a risk of failing to meet performance goals, the new magnitude of these uncertainties may now lead to either overly conservative design practices on one hand, or “fragile” designs on the other. The *degree* to which uncertainty actually changes the expected performance of a design (and thus the nature of what an “optimal” design really is) is not something that has been discussed much in prior work. In this work we show that it is possible to define, model, and quantify architectural risk. Luckily the ideas of risk and risk-management are well understood in economics and by drawing upon this expertise, we are able to describe a new analytic framework for high-level risk-aware architectural analysis. We embody this framework in a symbolic/statistical analysis system that eases the exploration of these surprisingly complex design spaces. We show that ignoring the degree to which parameters are unknown, even under fairly simple and conservative performance modeling assumptions, can lead to designs with radically different risk profiles. While there is always a design that maximizes expected performance, we show (even absent the confounding factors of cost, energy, thermal constraints, etc.) the “optimal” design may only be a point in the risk-performance trade-off space.

7 ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grants No. 1239567, 1162187, and 1563935.

We’d like to thank Gabriel Loh for his constructive feedback and critiques; Changwei Xu for being our statistics consultant; Yan Tang for discussion on data transformation and comments on the manuscript; and the anonymous reviewers, for their invaluable feedback.

REFERENCES

[1] A. R. Alameldeen and D. A. Wood. 2003. Variability in architectural simulations of multi-threaded workloads. In *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.* 7–18. https://doi.org/10.1109/HPCA.2003.1183520

[2] Muhammad Shoaib Bin Altaf and David A. Wood. 2017. LogCA: A High-Level Performance Model for Hardware Accelerators. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17).* ACM, New York, NY, USA, 375–388. https://doi.org/10.1145/3079856.3080216

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2010. A View of Cloud Computing. *Commun. ACM* 53, 4 (April 2010), 50–58. https://doi.org/10.1145/1721654.1721672

[4] Omid Azizi, Aqeel Mahesri, Benjamin C. Lee, Sanjay J. Patel, and Mark Horowitz. 2010. Energy-performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10).* ACM, New York, NY, USA, 26–36. https://doi.org/10.1145/1815961.1815967

[5] M. Bhadauria, V. M. Weaver, and S. A. McKee. 2009. Understanding PARSEC performance on contemporary CMPs. In *2009 IEEE International Symposium on Workload Characterization (IISWC).* 98–107. https://doi.org/10.1109/IISWC.2009.5306793

[6] Shekhar Borkar. 2007. Thousand Core Chips: A Technology Perspective. In *Proceedings of the 44th Annual Design Automation Conference (DAC '07).* ACM, New York, NY, USA, 746–749. https://doi.org/10.1145/1278480.1278667

[7] James Bornholt, Todd Mytkowicz, and Kathryn S. McKinley. 2014. Uncertain<T>: A First-order Type for Uncertain Data. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14).* ACM, New York, NY, USA, 51–66. https://doi.org/10.1145/2541940.2541958

[8] G. E. P. Box and D. R. Cox. 1964. An Analysis of Transformations. *Journal of the Royal Statistical Society. Series B (Methodological)* 26, 2 (1964), 211–252. http://www.jstor.org/stable/2984418

[9] David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: A Framework for Architectural-level Power Analysis and Optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA '00).* ACM, New York, NY, USA, 83–94. https://doi.org/10.1145/339647.339657

[10] C.L. Philip Chen and Chun-Yang Zhang. 2014. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences* 275 (2014), 314 – 347. https://doi.org/10.1016/j.ins.2014.01.015

[11] Kwang-Ting Tim Cheng and Dmitri B. Strukov. 2012. 3D CMOS-memristor Hybrid Circuits: Devices, Integration, Architecture, and Applications. In *Proceedings of the 2012 ACM International Symposium on International Symposium on Physical Design (ISPD '12).* ACM, New York, NY, USA, 33–40. https://doi.org/10.1145/2160916.2160925

[12] Eric S. Chung, Peter A. Milder, James C. Hoe, and Ken Mai. 2010. Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '43).* IEEE Computer Society, Washington, DC, USA, 225–236. https://doi.org/10.1109/MICRO.2010.36

[13] K. Constantinides, O. Mutlu, and T. Austin. 2008. Online design bug detection: RTL analysis, flexible mechanisms, and evaluation. In *2008 41st IEEE/ACM International Symposium on Microarchitecture.* 282–293. https://doi.org/10.1109/MICRO.2008.4771798

[14] J. A. Cunningham. 1990. The use and evaluation of yield models in integrated circuit manufacturing. *IEEE Transactions on Semiconductor Manufacturing* 3, 2 (May 1990), 60–71. https://doi.org/10.1109/66.53188

[15] Christophe Dubach, Timothy Jones, and Michael O'Boyle. 2007. Microarchitectural Design Space Exploration Using an Architecture-Centric Approach. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 40).* IEEE Computer Society, Washington, DC, USA, 262–271. https://doi.org/10.1109/MICRO.2007.26

[16] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. Dark Silicon and the End of Multicore Scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11).* ACM, New York, NY, USA, 365–376. https://doi.org/10.1145/2000064.2000108

[17] Songchun Fan, Seyed Majid Zahedi, and Benjamin C. Lee. 2016. The Computational Sprinting Game. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16).* ACM, New York, NY, USA, 561–575. https://doi.org/10.1145/2872362.2872383

[18] H. D. Foster. 2015. Trends in functional verification: A 2014 industry study. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC).* 1–6. https://doi.org/10.1145/2744769.2744921

[19] M. Guevara, B. Lubin, and B. C. Lee. 2013. Navigating heterogeneous processors with market mechanisms. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA).* 95–106. https://doi.org/10.1109/HPCA.2013.6522310

[20] M. Guevara, B. Lubin, and B. C. Lee. 2014. Strategies for anticipating risk in heterogeneous system design. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA).* 154–164. https://doi.org/10.1109/HPCA.2014.6835926

[21] John L. Gustafson. 1988. Reevaluating Amdahl's Law. *Commun. ACM* 31, 5 (May 1988), 532–533. https://doi.org/10.1145/42411.42415

[22] A. Hartstein and Thomas R. Puzak. 2002. The Optimum Pipeline Depth for a Microprocessor. In *Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA '02).* IEEE Computer Society, Washington, DC, USA, 7–13. http://dl.acm.org/citation.cfm?id=545215.545217

[23] M. D. Hill and M. R. Marty. 2008. Amdahl's Law in the Multicore Era. *Computer* 41, 7 (July 2008), 33–38. https://doi.org/10.1109/MC.2008.209

[24] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano. 2005. A novel nonvolatile memory with spin torque transfer magnetization switching: spin-ram. In *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.* 459–462. https://doi.org/10.1109/IEDM.2005.1609379

[25] Intel. 2017. Intel Processor Pricing. https://www.intc.com/investor-relations/investor-education-and-news/cpu-price-list. (2017). Accessed: 2017-04-03.

[26] Engin Ïpek, Sally A. McKee, Rich Caruana, Bronis R. de Supinski, and Martin Schulz. 2006. Efficiently Exploring Architectural Design Spaces via Predictive Modeling. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII).* ACM, New York, NY, USA, 195–206. https://doi.org/10.1145/1168857.1168882

[27] N. E. Jerger, A. Kannan, Z. Li, and G. H. Loh. 2014. NoC Architectures for Silicon Interposer Systems: Why Pay for more Wires when you Can Get them (from your interposer) for Free?. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture.* 458–470. https://doi.org/10.1109/MICRO.2014.61

[28] Stanley Kaplan and B. John Garrick. 1981. On The Quantitative Definition of Risk. *Risk Analysis* 1, 1 (1981), 11–27. https://doi.org/10.1111/j.1539-6924.1981.tb01350.x

[29] I. Koren and Z. Koren. 1998. Defect tolerance in VLSI circuits: techniques and yield analysis. *Proc. IEEE* 86, 9 (Sep 1998), 1819–1838. https://doi.org/10.1109/5.705525

[30] C. Lam. 2008. Cell Design Considerations for Phase Change Memory as a Universal Memory. In *2008 International Symposium on VLSI Technology, Systems and Applications (VLSI-TSA).* 132–133. https://doi.org/10.1109/VTSA.2008.4530832

[31] Abraham Lee. 2014. *Real-time latin-hypercube-sampling-based Monte Carlo Error Propagation.* https://github.com/tisimst/mcerp

[32] B Lee and David Brooks. 2006. Statistically rigorous regression modeling for the microprocessor design space. In *ISCA-33: Workshop on Modeling, Benchmarking, and Simulation.*

[33] Benjamin C. Lee and David Brooks. 2008. Efficiency Trends and Limits from Comprehensive Microarchitectural Adaptivity. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIII).* ACM, New York, NY, USA, 36–47. https://doi.org/10.1145/1346281.1346288

[34] Benjamin C. Lee and David M. Brooks. 2006. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII).* ACM, New York, NY, USA, 185–194. https://doi.org/10.1145/1168857.1168881

[35] B. C. Lee and D. M. Brooks. 2007. Illustrative Design Space Studies with Microarchitectural Regression Models. In *2007 IEEE 13th International Symposium on High Performance Computer Architecture.* 340–351. https://doi.org/10.1109/HPCA.2007.346211

[36] Benjamin C. Lee, David M. Brooks, Bronis R. de Supinski, Martin Schulz, Karan Singh, and Sally A. McKee. 2007. Methods of Inference and Learning for Performance Modeling of Parallel Applications. In *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '07).* ACM, New York, NY, USA, 249–258. https://doi.org/10.1145/1229428.1229479

[37] Benjamin C. Lee, Jamison Collins, Hong Wang, and David Brooks. 2008. CPR: Composable Performance Regression for Scalable Multiprocessor Models. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 41).* IEEE Computer Society, Washington, DC, USA, 270–281. https://doi.org/10.1109/MICRO.2008.4771797

[38] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).* 469–480.

[39] X. Liang and D. Brooks. 2006. Mitigating the Impact of Process Variations on Processor Register Files and Execution Units. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06).* 504–514. https://doi.org/10.1109/MICRO.2006.37

[40] G. H. Loh, Y. Xie, and B. Black. 2007. Processor Design in 3D Die-Stacking Technologies. *IEEE Micro* 27, 3 (May 2007), 31–48. https://doi.org/10.1109/MM.

2007.59

[41] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. 2014. Race Logic: A Hardware Acceleration for Dynamic Programming Algorithms. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA '14)*. IEEE Press, Piscataway, NJ, USA, 517–528. http://dl.acm.org/citation.cfm?id=2665671.2665747

[42] Mark Oskin, Frederic T. Chong, and Matthew Farrens. 2000. HLS: Combining Statistical and Symbolic Simulation to Guide Microprocessor Designs. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA '00)*. ACM, New York, NY, USA, 71–82. https://doi.org/10.1145/339647.339656

[43] A. Rahimi, L. Benini, and R. K. Gupta. 2016. Variability Mitigation in Nanometer CMOS Integrated Systems: A Survey of Techniques From Circuits to Software. *Proc. IEEE* 104, 7 (July 2016), 1410–1448. https://doi.org/10.1109/JPROC.2016.2518864

[44] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. 2008. VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects. *IEEE Transactions on Semiconductor Manufacturing* 21, 1 (Feb 2008), 3–13. https://doi.org/10.1109/TSM.2007.913186

[45] JÃijrgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61 (2015), 85 – 117. https://doi.org/10.1016/j.neunet.2014.09.003

[46] David W. Scott. 2008. *Kernel Density Estimators*. John Wiley Sons, Inc., 125–193. https://doi.org/10.1002/9780470316849.ch6

[47] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder. 2003. Discovering and exploiting program phases. *IEEE Micro* 23, 6 (Nov 2003), 84–93. https://doi.org/10.1109/MM.2003.1261391

[48] Dmitri B. Strukov, Gregory S. Snider, Duncan R. Stewart, and R. S. Williams. 2008. The missing memristor found. *Nature* 453, 7191 (May 01 2008), 80–3. https://search.proquest.com/docview/204473846?accountid=14522 Copyright - Copyright Nature Publishing Group May 1, 2008; Last updated - 2014-03-19; CODEN - NATUAS.

[49] Xian-He Sun and Yong Chen. 2010. Reevaluating Amdahl's law in the multicore era. *J. Parallel and Distrib. Comput.* 70, 2 (2010), 183 – 188. https://doi.org/10.1016/j.jpdc.2009.05.002

[50] SymPy Development Team. 2016. *SymPy: Python library for symbolic mathematics*. http://www.sympy.org

[51] M Mitchell Waldrop. 2016. The chips are down for Moore's law. *Nature* 530, 7589 (2016), 144–147.

[52] D. H. Woo, D. H. Woo, D. H. Woo, D. H. Woo, H. H. S. Lee, H. H. S. Lee, H. H. S. Lee, and H. H. S. Lee. 2008. Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era. *Computer* 41, 12 (Dec 2008), 24–31. https://doi.org/10.1109/MC.2008.494

[53] L. Yavits, A. Morad, and R. Ginosar. 2014. The effect of communication and synchronization on Amdahl's law in multicore systems. *Parallel Comput.* 40, 1 (2014), 1 – 16. https://doi.org/10.1016/j.parco.2013.11.001

[54] Seyed Majid Zahedi and Benjamin C. Lee. 2014. REF: Resource Elasticity Fairness with Sharing Incentives for Multiprocessors. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*. ACM, New York, NY, USA, 145–160. https://doi.org/10.1145/2541940.2541962