Architectural Risk

Weilong Cui and Timothy Sherwood
University of California, Santa Barbara

Designing a system involves taking on risk that a design will fail to meet its performance goals. While risk assessment and management are typically treated as independent to questions of performance, they are more tightly linked than one might expect. A "risk-minimizing" and "performance-optimizing" design might not be the same, and new techniques to help make smarter choices between the two are needed. Surprisingly, even simple performance/risk tradeoffs are nearly impossible to reason about with intuition alone.

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 are 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. A shifting application landscape, the challenges of continued transistor scaling, and the emergence of new chip technologies, new memory technologies, and new computing devices and paradigms[1-3] add up to a computing landscape wrought with uncertainty.

The true risks a company takes on when attempting to bring a new architecture to market are complex and varied. 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 architectural design decisions are made at the level of spreadsheets and other high-level analytical models or data points drawn from experience. Unlike in software,[4] operating systems,[5] and device modeling,[6] 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.

While it would be wonderful if we could gather highly detailed data about each and every uncertainty, the reality is that such data is often highly sensitive and very hard to come by. We describe a technique that can be used (even assuming no *a priori* knowledge) in practice to effectively estimate a host of established architecturally relevant distributions from limited data points for the purpose of quantifying and exploring the impact of a varying amount of uncertainty in the system.

More importantly, we show how even seemingly straightforward questions such as core selection can lead to some surprising new interactions when uncertainty is taken into consideration. Being "risk-unaware" can lead you to decisions that not only have less desirable distributions (such as with a tail of potentially really bad designs) but also can be strictly sub-optimal even when only considering expected (average-case) performance.

Published by the IEEE Computer Society
0272-1732/18/$33.00 ©2018 IEEE

DEFINING ARCHITECTURE RISK

When attempting to gain an understanding of risk, it is easiest to start with discrete events and build from our intuition for finances. Imagine comparing two portfolios, P1 and P2. Under P1, we have two possible outcomes: 75 percent of the time, we get a 3X return, while 25 percent of the time, we get a 1X loss. The expected return (when doing the weighted average) is 2X. Compare that to P2 where 50 percent of the time, we get a 2X return, and 50 percent of the time, we get a 1.5X return. The expected value of P2 is 1.75X and clearly less than that of P1. If one just wants to maximize expected return, P1 is the way to go. But if it is your entire life savings, of course everyone will pick P2 because, in reality, it is a far worse outcome to lose all your money than a simple "-1" multiplier would indicate. While we still care about expected value, it must be balanced against risk—the sum of "badness" that comes from bad outcomes. Therefore, to quantify risk, we must assign a "badness" function over the outcomes and take the "expected badness" into account. For a portfolio, we might assign one based on the client losing money (for example, the square of the total loss). In the example above, the risk, under such a metric, would be 0.25 for P1 and 0.0 for P2. Thus, intuitively, risk is the impact of some uncertain event weighed by its probability of occurring.[7]

In the performance space, we too have a risk to manage: the risk of failing to meet our performance objective. Just like in the financial model where it is bad for someone to lose money but really bad to lose a lot of money, we apply a function over the extent to which performance failed to meet the objective as our metric of risk. Here we can say that risk is the impact (such as the negative effect of failing to hit a performance target) of some uncertainty (such as technical unknowns relating to performance) weighted by its probability. We then need to consider this risk in concert with expected performance. Of course, real systems have far more than two or three possible "outcomes" and are instead governed by a distribution of possibilities, but the approaches are identical in the limit. While we have concentrated on a more informal description of risk here, in our recent MICRO conference article, we present a more careful and complete formalism of architectural risk.[8] Figure 1 captures this intuition graphically.

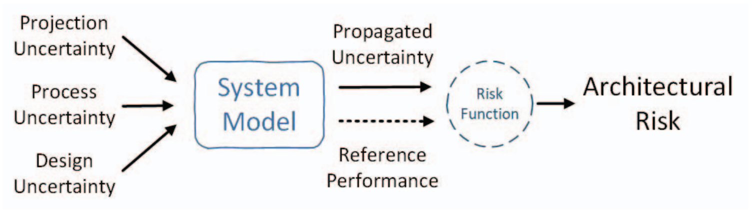

Figure 1. High-level overview of the relationship between uncertainty and risk.

A Taxonomy of Uncertainty Sources

Computer architectures are exposed to at least three major sources of uncertainties throughout the system stacks: projection uncertainty, process uncertainty, and design uncertainty. Note that we do not consider model inaccuracy as uncertainty in this work (see the "Uncertainty versus Inaccuracy" sidebar).

Projection uncertainty

Projection uncertainty comes from assumptions about the future. At the application level, a system design might target a specific set of applications, which might 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 might target underlying technologies still working their way out of the research labs. The performance of these future technologies is uncertain by nature.

Process uncertainty

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 underperform, while still others will fail to work at all.

Design uncertainty

Design uncertainty comes from the hardware design process itself. Components (such as cores and accelerators) with unresolved critical errors or that introduce significant security vulnerabilities might be prevented from being accessible in an initial rollout of a product. This class of uncertainty is 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.

QUANTIFYING ARCHITECTURAL RISK

The interactions of different uncertainties, as we show later, quickly become impossible to intuit or estimate using "back-of-the-envelope" calculation. We combine bootstrapping methods, symbolic algebraic manipulation, and Monte Carlo simulations to automate the quantification. At a high level, if the architecture model can be described as a set of mutually dependent closed-form relations and the uncertainty as a large set of samples or sampling functions, our framework can then construct uncertainty models, propagate such uncertainties through the system model, and calculate architectural risk automatically.

Uncertainty Modeling

Given a few data points (drawn from the hidden ground truth distribution governing the uncertainty), we first test whether the dataset can be transformed to normality through the Box-Cox test (Step 1 in Figure 2). If it cannot pass the test (a rare case in practice), we apply kernel density estimation (KDE) methods directly to the dataset. These methods find a best-fit non-parametric distribution (Step 2) and use its sampling function to facilitate uncertainty propagation. Otherwise, we can transform the dataset to normality using Box-Cox transformation (Step 3), fit a Gaussian distribution in the transformed domain (Step 4), re-sample/bootstrap (Step 5), and back-transform the samples to the original domain and reconstruct the distribution (Step 6) to approximate the hidden ground truth. Although not as accurate as the best-fit non-parametric KDE, such a method enables us to hand-tune the desired uncertainty level from each source and hence be able to explore the trend as input uncertainties scale. Figure 2 highlights this process with a simple running example of samples from a lognormal distribution.

Uncertainty Propagation

In Figure 3, we present an overview with a simple example of how the front-end modeling and symbolic execution works. System modeling builds a set of mutually dependent equations describing the high-level performance relations of the system (Step 1), which are then passed to a symbolic execution engine like SymPy. The result is a set of symbolic solutions for each (Step 2), which are then converted into lambda functions (Step 3). At the end of this process, we have a set of callable functions that are then provided to the back-end for numerical computation.

Figure 4 shows the back-end uncertainty injection and propagation process. Given the set of functions and uncertainty models, each uncertain variable gets evaluated first if there are no uncertain variables on the right side of their solutions (Step 1). The values of certain inputs (usually architecture parameters), c and x in this case, should be provided by the system designer. The framework then generates distributions for all the uncertain variables (Step 2) and inserts them into the corresponding lambda (Step 3). Each function is then evaluated using Monte Carlo simulation (Step 4). The result of the Monte Carlo simulation is a set of samples for each responsive variable in the system. We then re-construct a distribution from the propagated samples (Step 5). Finally, we calculate risk based on the distribution of the responsive variable and the risk function provided (Step 6).

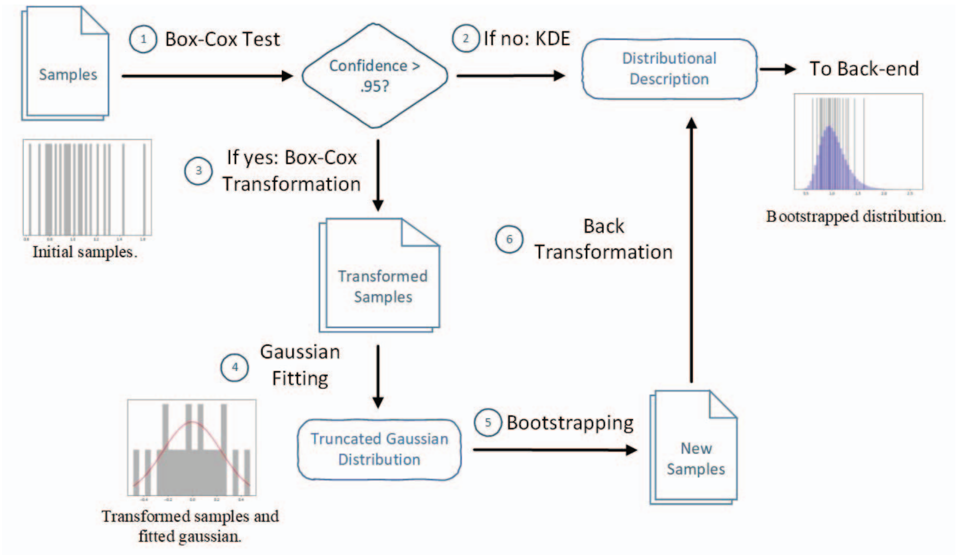

Figure 2. Uncertainty modeling. Depending on the result of the Box-Cox test, we can either fit a non-parametric distribution using KDE or fit a parametric model through Box-Cox transformation. Both forms can facilitate uncertainty propagation by the back-end.

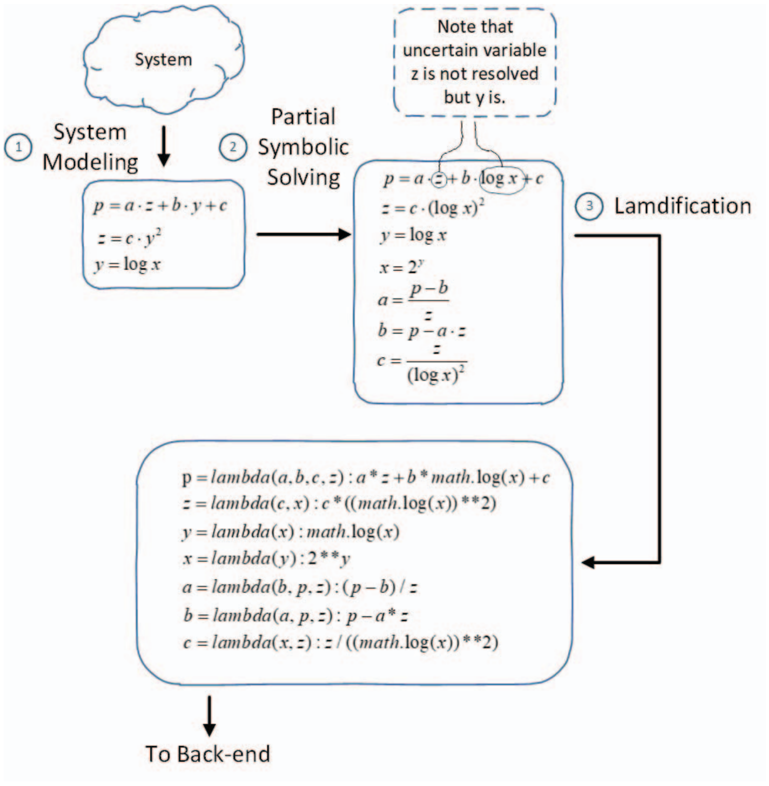
System
1 System Modeling
$p = a\cdot z + b\cdot y + c$
$z = c\cdot y^2$
$y = \log x$
2 Partial Symbolic Solving
Note that uncertain variable z is not resolved but y is.
$p = a\cdot z + b\cdot \log x + c$
$z = c\cdot(\log x)^2$
$y = \log x$
$x = 2^y$
$a = \frac{p-b}{z}$
$b = p - a\cdot z$
$c = \frac{z}{(\log x)^2}$
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 3: Front-end workflow for automatic risk quantification.

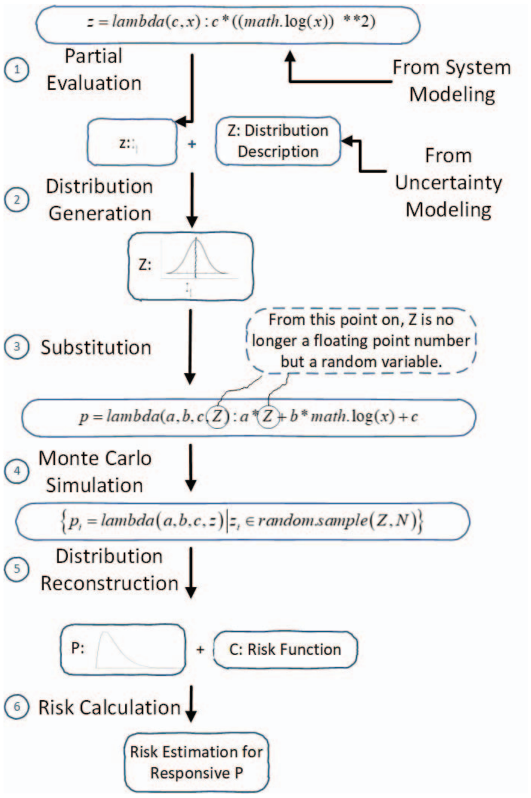
$z = lambda(c,x): c*((math.log(x)) **2)$
1 Partial Evaluation
From System Modeling
z::
+
Z: Distribution Description
From Uncertainty Modeling
2 Distribution Generation
Z:
3 Substitution
From this point on, Z is no longer a floating point number but a random variable.
$p = lambda(a,b,c,Z): a*Z + b*math.log(x)+c$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

Figure 4: Back-end workflow for automatic risk quantification.

DESIGN SPACE EXPLORATION OF CHIP MULTIPROCESSOR (CMP) CORE SELECTION

Using an extension of the classic Hill and Marty model,[9] we 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 256 units and consider the full spectrum of designs (rather than just one big core coupled with many tiny cores). To highlight the insights from this study, we use an application with relatively low parallelism and high communication overhead, which we refer to as LPHC.

Closed-Form System Model

The full closed-form system model uses the following equations:

1. $Speedup = \dfrac{1}{T_{seq} + T_{par}}$
2. $T_{seq} = \dfrac{1 - f + c \times N_{core}}{P_{seq}}$
3. $T_{par} = \dfrac{f}{P_{par}}$
4. $P_{seq} = \max\left\{P_{core_i} \middle| N_{core_i} > 0\right\}$
5. $P_{par} = \sum_{i \in core\_types} N_{core_i} \times P_{core_i}$

6. $N_{core} = \sum_{i \in core\_types} N_{core_i}$
7. $P_{core_i} = \sqrt{A_{core_i}}$
8. $A_{total} = \sum_{i \in core\_types} N_{core_i} \times A_{core_i}$

The design problem of core selection translates into deciding values for P_{core_i} and N_{core_i} under constraints. In this model, one chooses the best-performing core design to execute the serial code (Equation 4) and uses the aggregated performance of all cores to execute the parallel code (Equation 5). Pollack's rule is used to model core performance as a function of resources consumed (Equation 7). Designs are bounded by the total area/resource available on chip (Equation 8). In addition to these classic assumptions, we also take communication overhead among different cores into account, denoted as *c* in Equation 2, which is some fraction of the sequential workload. The amount of communication overhead is proportional to the total number of cores on chip (Equation 6) to capture the overhead introduced along with parallelization.

Hidden Ground Truth Uncertainty Models

There are five types of uncertainties that might be considered in the above system. Uncertainties in target application behavior impact *f* and *c*. Uncertainties in process/manufacturing can affect both P_{core_i} (different core instances might end up with varying performance properties due to intra-die variation) and N_{core_i} (due to fabrication defects impacting yield). Uncertainties in design might 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.

By pulling from literature[10-12] and by their physical definitions (we simplify design uncertainty to a 0/1 case, or a Bernoulli distribution), we are able to construct the hidden ground truth distributions that govern these uncertainties:

- $f, c \overset{i.i.d}{\sim} \frac{Binomial(M, p)}{M}$
- $N_{core_i} \sim Binomial(N_i, yield_{core_i})$
- $P_{core_i} \sim Bernoulli(p) \times LogNormal(\mu, \sigma)$
- $yield_{core_i} = (1 + \frac{d \times A_{core_i}}{\alpha})^{-\alpha}$

Note that *M* is set to meet the level of uncertainty we desire in simulation, while N_i is the designed number of $core_i$. Input uncertainty (standard deviation of the governing distributions) is set to σ x μ, where μ is the certain value (if you are risk-unware), and we hand-tune the uncertainty level by varying σ. In our evaluation, we put ourselves in the designers' shoes by using only a handful of samples from such distributions but no knowledge of the equations themselves.

Results and Discussion

Uncertainty propagation

Figure 5 presents the propagated uncertainty and its impact on expected performance. We can tell that the propagated uncertainty is not additive, nor is its impact on the expected system performance. In fact, the expected performance is not even monotonic, in this case. This is mainly because, unlike risk (which we discuss later), uncertainty has a two-sided impact on performance: The system might underperform in some cases but can outperform in others. The important message is that uncertainties propagate through the model with non-intuitive interaction, and the resulting performance distribution is usually beyond what a simple "back-of-the-envelope" estimation can reveal.

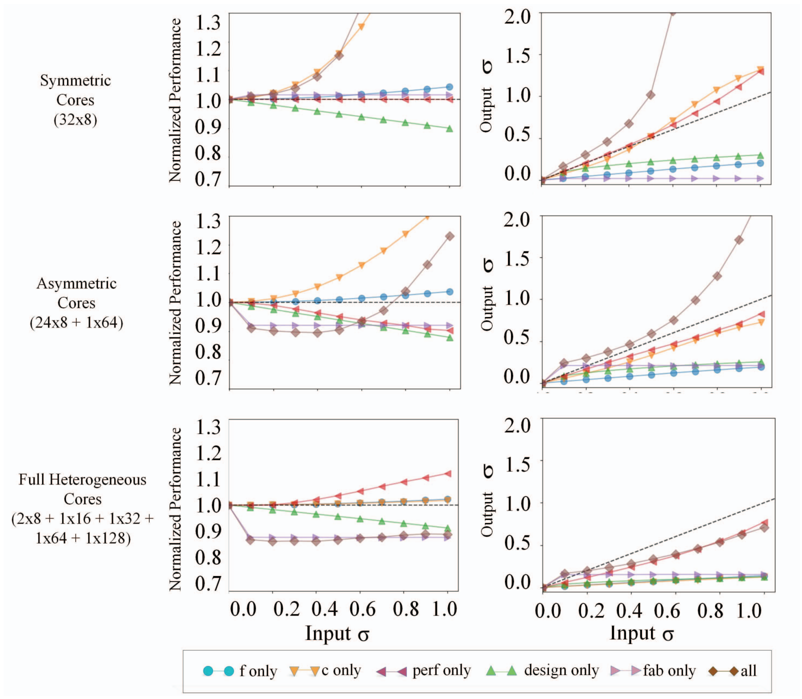

Figure 5. Expected performance and propagated uncertainties at different input uncertainty level *σ* for LPHC application. Three topologies, denoted as (core_number x core_size), are shown here.

Tradeoff space and impact on design

Figure 6 shows the performance-risk space for all configurations. Each configuration is merely a point in this new space, and the conventional performance-only optimal design is often not even on the Pareto curve. Another important message is that there is this new tradeoff space between expected performance and architectural risk and that even when the architectural decisions being evaluated are as simple as "what is the distribution of core sizes that should be used," a significant amount of risk can be mitigated with very modest moves away from the design that is absolutely optimal for nominal performance.

Fundamental trends in core configurations

In terms of what core configurations are favored, as shown in Figure 7, if we consider application uncertainty alone, when it gets larger, more asymmetric configurations are generally favored. This trend results from the asymmetric impact uncertainty has on performance (it can either lead to worse performance or, sometimes, better). 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 architecture uncertainty gets larger, we can see that more symmetric configurations are preferred. 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 lower-performing large core. Another reason is that multiple cores of each type are chosen to fight the intra-die process variation, leading to fewer core types on chip because of total area/resource constraint. These two counter-directional trends are the main reasons why the design space is very irregular and complicated.

Architectural risk to financial risk

Using the publicly available Intel 2017 CPU price list as the risk function (architectural risk, in this case, measures the dollar value lost because of underperformance), we show in Figure 8 that being risk-aware and choosing a different design than the conventional performance-only optimal option recovers more than $40 per chip even with a relatively small amount of input uncertainties.

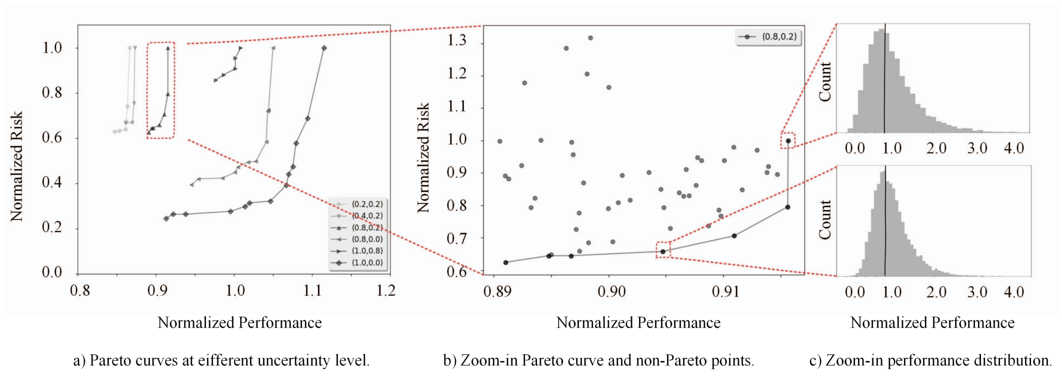

Figure 6. Example tradeoff space between performance-optimal and risk-optimal design. Each point in the space is a performance distribution of a certain design (core-selection) at some input uncertainty level (denoted as a tuple of (σ_app, σ_arch)) with the LPHC application. Performance is normalized to that of the conventional case (risk-oblivious, performance-optimal design). Risk is calculated using a simple quadratic cost function (to penalize the far-left tail) and is normalized to that of the performance-optimal design at each input uncertainty level.

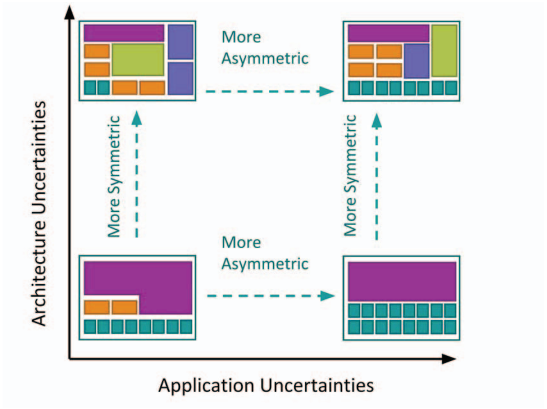

Figure 7. Core configurations of performance-optimal designs for LPHC. As application uncertainty grows, more asymmetric configurations are preferred, while as architecture uncertainty grows, more symmetric configurations are generally more favored.

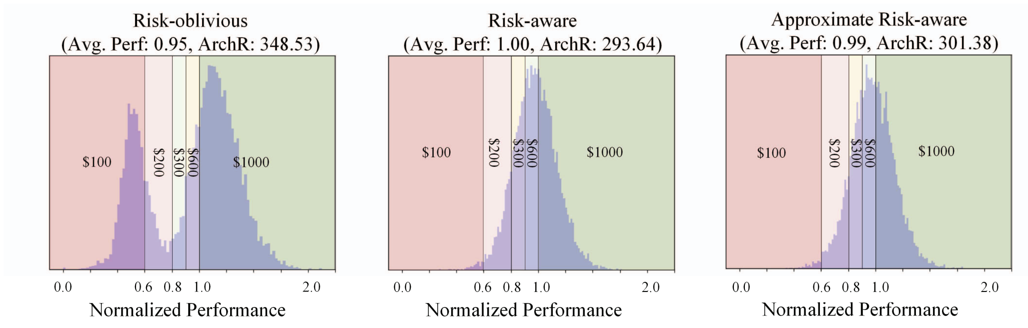

Figure 8. Binning of design results under uncertainties. A price binning is laid on top of the performance distribution derived from each design with LPHC application at an input uncertainty level of σ_app = σ_arch = 0.2.

CONCLUSION

Dealing quantitatively with uncertainty at early design cycles and the risk it creates requires new concepts and tools. The goal of this new line of work is to promote a new first-order design concern (architectural risk) and to provide a systematic framework to quantify such risks, with the aim of helping find designs that are more robust to the impacts of uncertainty than performance-only optimal designs while still maintaining very strong performance in the common case. We already consider risk in practice today, as it has become a more and more common practice to "fuse out" features of the chip in case the new components impact performance negatively. As we integrate more and more heterogeneous systems, the options available to us grow.

As this growth happens, both in scale and complexity, we will be forced away from highly detailed cycle-accurate simulations to higher-level models to cover a large design space and explore the fundamental tradeoffs. The most well-studied heterogeneous core selection problem of Hill and Marty serves as an example of this and shows how considering architectural risk and quantifying its interaction with performance can lead to new insights about how to design a chip in the face of uncertainties. Such a simple high-level model is significantly confounded by the introduction of uncertainty, let alone a much more complicated real processor.

This type of analysis essentially opens the door to a new architecture research space centered around three basic questions:

1. How can architectural choices strike an effective balance between performance, resource utilization (energy or area), and risk?
2. How can one introduce new microarchitectural support, new computer organizations, and hardware design tools to change the risk-performance tradeoff space in fundamentally new ways?
3. How can we work together as a community to properly leverage the best thinking on each of these individual risk factors to create a cohesive understanding?

While Questions 1 and 2 are already a close parallel to other computer architecture works, Question 3 will require both new methods and coordination. In our current approach, we carry out the analysis with a combination of symbolic execution, statistical methods, and Monte Carlo simulation. The general framework of identifying, modeling, and propagating uncertainty to architectural risk can be extended to different techniques for other flavors of system analysis. For example, when applied to complex circuit-level design, an analytical generalized polynomial chaos (gPC) method can be used to perform the propagation instead of Monte Carlo simulation, which will allow it to scale to hundreds or even thousands of input variables. These methods present a new opportunity. If widely used, they might form a foundation that enables a new collective effort by the academic community to collect, share, and interactively refine models that can guide real-life system designs.

SIDEBAR: UNCERTAINTY VERSUS INACCURACY

There is a philosophical distinction between the uncertainties we consider (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, and so on—but in the end, measurement inaccuracy could be eliminated given enough resources. However, even if one had infinite resources, the uncertainties we proposed would still exist. It is much harder (or even impossible) to remove such uncertainties without a fundamentally new understanding of the future. (This is the difference between "aleatoric" and "epistemic" uncertainty.)

ACKNOWLEDGMENTS

This material is based on work supported by the NSF under grants 1740352, 1730309, 1717779, 1563935, 1444481, and 1341058, as well as a gift from Cisco Systems.

REFERENCES

1. D. Strukov et al., “The missing memristor found,” *Nature*, 2008.
2. G. Loh, Y. Xie, and B. Black, “Processor Design in 3D Die-Stacking Technologies,” *IEEE Micro*, 2007.
3. H. Wassel et al., “Opportunities and Challenges of using Plasmonic Components in Nanophotonic Architectures,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2012.
4. J. Bornholt, T. Mytkowicz, and K. McKinley, “Uncertain<t>: Abstractions for uncertain hardware and software,” *IEEE Micro*, 2015.
5. A. Alameldeen and D. Wood, “Variability in architectural simulations of multi-threaded workloads,” *The Ninth International Symposium on High-Performance Computer Architecture* (HPCA), 2003.
6. S. Mittal, “A Survey of Architectural Techniques for Managing Process Variation,” *ACM Computing Surveys (CSUR)*, 2016.
7. S. Kaplan and B. Garrick, “On the Quantitative Definition of Risk,” *Risk Analysis*, 1981.
8. W. Cui and T. Sherwood, “Estimating and understanding architectural risk,” *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017.
9. M. Hill and T. Sherwood, “Amdahl's Law in the Multicore Era,” *Computer*, 2008.
10. A. Rahimi, L. Benini, and R. Gupta, “Variability Mitigation in Nanometer CMOS Integrated Systems: A Survey of Techniques from Circuits to Software,” *Proceedings of the IEEE*, 2016.
11. J. Cunningham, “The use and evaluation of yield models in integrated circuit manufacturing,” *IEEE Transactions on Semiconductor Manufacturing*, 1990.
12. H. Foster, “Trends in functional verification: A 2014 industry study,” *52nd ACM/EDAC/IEEE Design Automation Conference* (DAC), 2015.

ABOUT THE AUTHORS

Weilong Cui is a PhD candidate in the Department of Computer Science at the University of California, Santa Barbara. His research interests include statistical/economic-inspired methods and programming language for architecture performance modeling, as well as novel micro-/system-architecture and its interaction with software. Cui has a master’s degree in computer science from Peking University, China. Contact him at cuiwl@cs.ucsb.edu.

Timothy Sherwood is a professor in the Department of Computer Science at the University of California, Santa Barbara. His research focuses on the development of processors exploiting novel technologies, designed for provable properties, and/or implementing hardware-aware algorithms. Sherwood has a PhD in computer science from the University of California, San Diego. He is a senior member of the IEEE and an ACM Distinguished Scientist. Contact him at sherwood@cs.ucsb.edu.