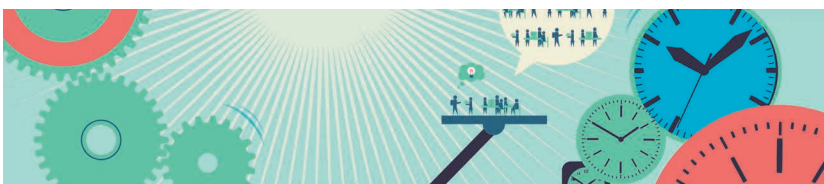


Proteus: Language and Runtime Support for Self-Adaptive Software Development

Saeid Barati, University of Chicago
 Ferenc A. Bartha, Rice University
 Swarnendu Biswas, University of Texas at Austin
 Robert Cartwright and Adam Duracz, Rice University
 Donald S. Fussell, University of Texas at Austin
 Henry Hoffmann and Connor Imes, University of Chicago
 Jason E. Miller, Massachusetts Institute of Technology
 Nikita Mishra, University of Chicago
 Arvind, Massachusetts Institute of Technology
 Dung Nguyen and Krishna V. Palem, Rice University
 Yan Pei and Keshav Pingali, University of Texas at Austin
 Ryuichi Sai, Rice University
 Andrew Wright, Massachusetts Institute of Technology
 Yao-Hsiang Yang, Rice University
 Sizhuo Zhang, Massachusetts Institute of Technology

// Our software framework, Proteus, treats adaptation as a first-class object, enabling rapid development of robust, adaptive applications. Proteus developers specify their programs' intent and adaptable components (or knobs). A control-theoretic runtime continually monitors the running application, adjusting knobs so that the specified intent is met. //



FOR ALMOST 20 YEARS, software engineers have recognized the necessity of building self-adaptive software systems.^{1–3} Self-adaptation increases software lifetime, allowing software to modify its own behavior in response to environmental changes that would otherwise cause failure. Adaptivity has only become more important as computing transitions from batch processes to interactive applications running on energy-constrained devices. We interact with such applications constantly, and we are frustrated when they are too slow. Modern devices, however, are highly sensitive to resource consumption. Battery life is precious, and running our devices flat out will generate too much heat, damaging either the device or the user. Despite exponential improvements in hardware technology, these systems still rely heavily on efficient software. Complicating matters, these systems run in challenging environments, with constantly changing operating conditions and output requirements.

Achieving software adaptivity is challenging, requiring robust mechanisms for making decisions about what and when to adapt. Typically, engineers implementing adaptive software must be experts in both their application domain and in some other field—like control theory. Thus, most software systems have a limited ability to adapt to the types of environmental changes that are inevitable in long-lived software, particularly in ways that minimize resource consumption and maximize efficiency.

An ideal interactive application—whether interacting with mobile users or embedded sensors—should adapt to environmental changes to meet performance goals while minimizing resource usage. Such adaptivity cannot be realized through heuristics or

custom solutions—it requires a disciplined system design methodology with tools enabling software engineers to benefit without becoming experts in new domains. To this end, we are developing Proteus, a software development ecosystem supporting application adaptivity as a first-class construct, which promotes software longevity through resilience to environmental changes and separation of application logic and intent.

Proteus manages applications using a control-theoretic approach, augmented with machine learning (ML), and programming language tools for capturing user intent and implementation variants. Proteus recognizes that most interactive applications have the latent ability to trade computational cost for output fidelity. We refer to these tunable parameters as *knobs*. To cope with unpredictable changes that arise over the course of a mission, Proteus dynamically adjusts knob settings in deployed software to provide flexibility and efficiency that would otherwise be impossible.

Developing an Adaptive Video Encoder With Proteus

Video encoders find redundancy in video streams and then represent that redundancy in a compact manner. Video-encoding standards specify how to encode redundancy, but developers are free to find the redundancy in many possible ways.⁴

Searching for redundancy requires tradeoffs: the more resources, e.g., time or energy, spent looking for similar pixels, the higher the encoding quality. To make an encoder as general as possible, developers expose these tradeoffs to users as parameters that can be set for individual deployments.⁵ While these parameters provide great flexibility, they also introduce two major

challenges that transcend video encoding and are common to large, interactive applications:

- *Complexity*: These parameters affect different modules in the larger system and can interact in nonintuitive ways.^{6,7} For example, two parameters may both individually increase quality but reduce it when set simultaneously. Thus, the complexity challenge is modeling combinations of parameter settings.
- *Dynamics*: Optimal settings can vary with environmental changes, however.⁸ For example, the effect of parameters often depends both on the input and the availability of resources such as central processing unit cores. Thus, the dynamics challenge is finding optimal parameter settings as the input and the underlying system change.

Proteus makes it easy to develop robust adaptive software by directly addressing the twin challenges of complexity and dynamics (see Figure 1). Proteus users do not specify specific parameter (or knob) settings because it is too hard and the optimal settings may change. Instead, users specify their intent as to the high-level goals the program should achieve. For example, we intend for our video encoder to operate at 30 frames/s while maximizing quality. Proteus developers specify changeable components or knobs. In our video encoder, we have identified many knobs that affect encoding time and quality.⁹ Proteus learns the complex interactions of specified knobs¹⁰ and then controls knob settings to maintain intent in dynamic environments.^{11,12} Two key Proteus contributions are the system design and interfaces allowing

learning and control to be applied simultaneously.

Learning Video Performance/Accuracy Tradeoffs

Given a set of user-specified knobs, Proteus must model their effects on the intended behavior. This is a complex modeling problem because knobs combine in complicated ways and because behavior can vary between inputs. To get a sense of this complexity, we show results when using ML to model our video encoder.¹⁰ Figure 2(a) presents the results of encoding a variety of input videos with different knob settings. Each point in the figure represents the cost (frame encoding latency in milliseconds) and error (inverse of peak-signal-to-noise ratio) for one input video and knob settings. Points that correspond to the same input are colored identically.

Even for a single input, there are many knob combinations that produce the same output error but that have widely different costs. Given an input video and output error, we minimize cost by considering only the left-most point for each knob combination. Figure 2(b) shows these Pareto-optimal points for each input. Since these Pareto-optimal curves vary by input, this is a difficult learning problem. A naïve strategy would exhaustively explore all knob settings, but this space is huge and it is quite likely that the video scene would change before the exploration even finished.

For complex applications, the performance and quality are nonlinear, nonconvex functions of the inputs and knobs. So, it is difficult—if not impossible—to derive closed-form expressions for them. Our prior work uses ML to characterize application performance and quality, and Proteus builds off this article.¹⁰ Thus, Proteus

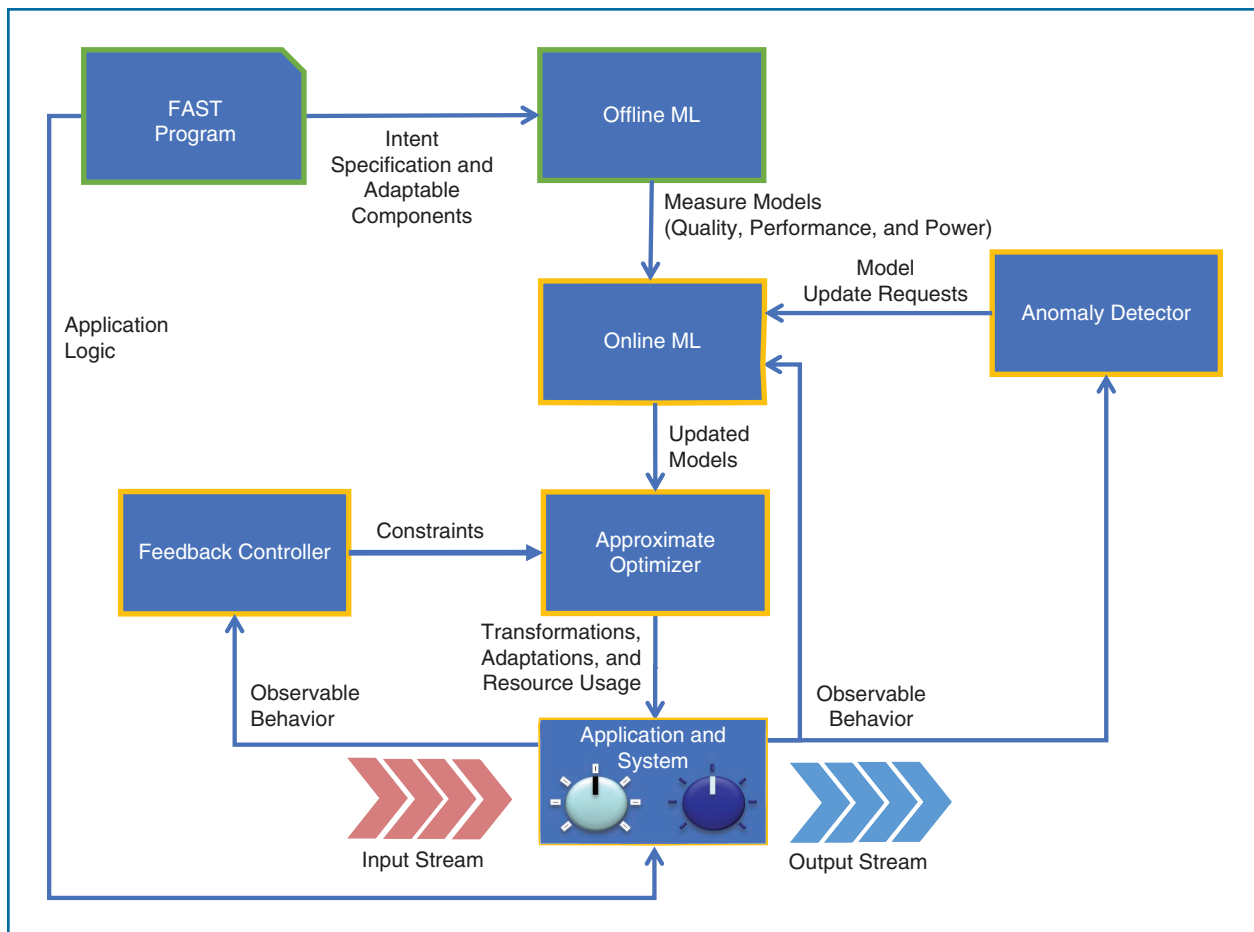


FIGURE 1. An overview of the Proteus approach to building dynamically adaptive software. Using the FAST programming language, developers specify not just the functionality of programs but also their nonfunctional intent (e.g., performance, quality, and energy goals) and configurable knobs that affect the intent. At compile time, offline ML models the relationship between intent and knobs. At runtime, online ML and control theoretic components monitor the application's behavior and continually adjusts knob settings to ensure the intent is met.

needs a set of training inputs and metrics to evaluate the quality and performance. The offline learning runs the program on these inputs using a variety of knob settings and learns performance and quality models. At this stage, Proteus assumes the underlying environment is stable.

The effectiveness of Proteus depends on the prediction accuracy of its machine-learned models. To mitigate the effect of bad models, the Proteus controller detects conditions that could

arise from such models and requests new models based on execution-time data. Our implementation of online ML uses random forests. To update the model, the decision trees are rebuilt using a weighted average of historical data and observations since the last update.

Controlling Video Encoder Intent

Control theory has recently received a great deal of attention because it

provides formal guarantees for reasoning about a software system's dynamic behaviors.¹³ In this section, we assume a ground truth model relating specific video encoder settings to their performance and quality. We then use existing techniques to synthesize a controller to meet a user-specified frame rate.^{9,14} The controller is a virtualized adaptive proportional-integral-derivative controller. Unlike traditional control systems, the Proteus controller first computes a virtualized control signal,

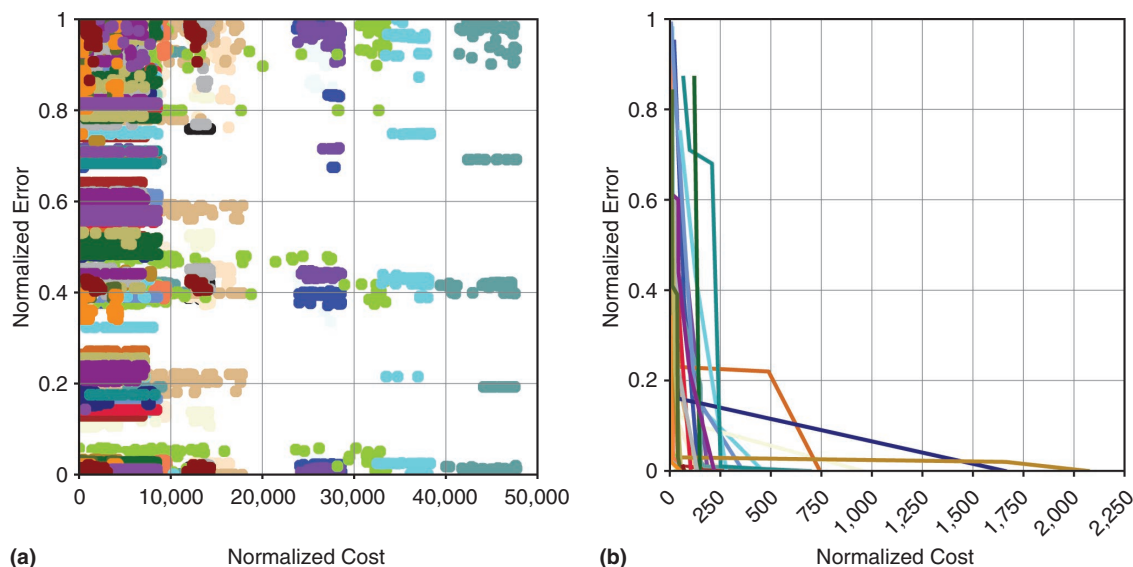


FIGURE 2. Learning tradeoffs for video encoding. This figure highlights that (a) many knob settings are not useful—sacrificing quality for no benefit and (b) the optimal tradeoffs vary greatly per input.

and then, a separate optimization step realizes that signal with specified knobs.

Figure 3(a) shows the benefits of control. We encode a 1,180-frame video with the intent of 30 frames/s and the highest possible quality. We compare control (turquoise line)—which dynamically adjusts knob settings—to selecting a single configuration based on the learned model (green line). After 670 frames, the input changes from a relatively easy scene, with ample redundancy, to a much harder one, which requires spending more time finding redundancy or sacrificing quality. The system is not explicitly aware that this change has occurred, but Proteus detects the change when performance dips below the goal. Proteus automatically adjusts knobs to bring performance back to the intent, while minimizing quality loss. The learned model puts the encoder into a good initial state, but it lacks a mechanism

to adapt to scene change. Control enables this adaptation but requires the model to guide its changes.

Combining Learning and Control

Our prior work¹⁰ shows learning builds accurate models of complex software, while control handles unpredictable environmental changes. Each overcomes a unique challenge. Intuitively, their combination should provide the benefits of both. The challenge is defining abstractions allowing the learner's output to be used by a control system, while ensuring that the resulting combination still provides useful formal guarantees, so that Proteus users can be sure their intent will be met. There are two specific obstacles. First, the learner produces nonlinear models of a discrete knob space, while most control designs assume continuous linear systems. Second, there is uncertainty in the learned

models, i.e., many knob setting combinations are estimated and never measured, which can negatively affect the controller.

To overcome these obstacles, Proteus builds on our prior work proposing abstract controllers.^{11,12} Whereas most software controllers manage literal values—i.e., set a specific parameter to a known value—Proteus controls virtual values. For example, to meet a performance goal, Proteus controls speedup rather than some specific knobs.

This abstract control system creates a layer of indirection between the behavior the controller is enforcing and the specific knob settings that achieve it. The Proteus runtime maps the virtual control signal to specific knob settings using the learned models. This abstract controller is thus quite general, operating with a number of different artificial intelligence (AI)/ML engines and allowing

the combination to be easily ported to many different platforms.¹¹ Furthermore, the abstract controller detects when intent is not met and requests a new model from Proteus's online learner. Compared to existing work on model predictive control (MPC), the Proteus controller is much more flexible, supporting both dynamic changes in available knobs and a much wider range of objective functions. Existing MPC approaches would have to be resynthesized if a knob becomes unavailable.

While the Proteus controller works on higher abstractions than typical controllers, it still provides formal guarantees that it will converge to the desired intent.¹¹ These guarantees are probabilistic and based on confidence intervals (which correspond to uncertainties due to measurement error and inherent system variability) provided by the learners. Thus, while any AI/ML approach could be paired with the controller, the best results will come from those which provide accurate confidence intervals.

Figure 3(b) shows the benefits of the Proteus approach (orange line) compared to a naïve combination of learning (with random forests) and control (pink line). The naïve combination simply uses existing control synthesis methodologies¹³ with a learned model instead of an empirically measured model. In this case, the behavior may fail to stabilize at the goal. In contrast, Proteus's combination of an abstract control system and self-monitoring converges to the desired behavior even when it starts with a bad model.

The second experiment [Figure 3(b)] compares Proteus to a straightforward approach of combining learning and control (denoted by *naïve*). Naïve simply replaces the standard modeling procedure for control

design with a learned model. In this case, the model is obtained using a training set (of videos) that poorly represents the current input. The naïve combination of learning and control oscillates wildly from low to high performance, alternately violating and exceeding the intent

(sacrificing encoding quality). In contrast, Proteus also begins oscillating, detects that its initial model is inappropriate for this input and dynamically constructs a new one. Proteus then brings the performance back to the intent and still adapts to the scene change.

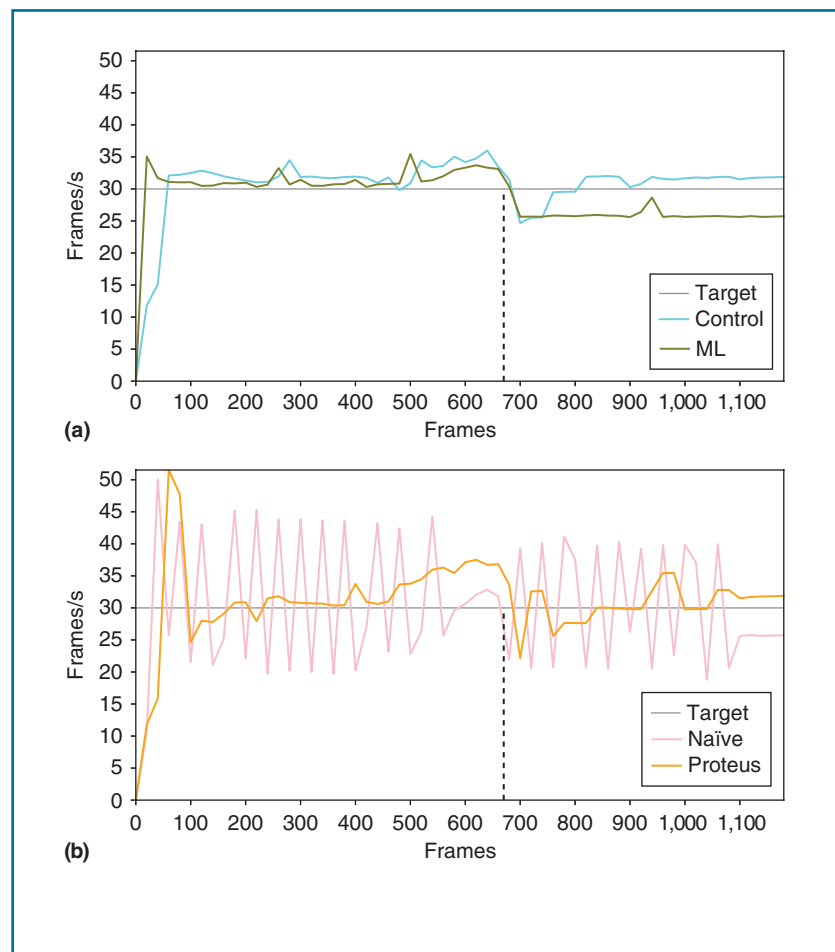


FIGURE 3. A summary of results from several experiments that compare different approaches to control a video encoder application. The horizontal axis corresponds to the frames of a video with two scenes—changing at frame 670—with the first being much easier to encode than the second. The gray line shows the desired target of achieving 30 frames/s. (a) The first experiment evaluates how ML and control respond to the dynamism in the environment. ML finds an initial configuration that delivers the performance, but then it falls well below 30 frames/s when the scene changes. Control starts with the same model, but it adjusts to the scene change to get performance back to the intent. (b) The second experiment is also shown.

The effectiveness of Proteus depends on the prediction accuracy of its machine-learned models.

Programming Language Support

Proteus exposes three abstractions to make programming with intent simple, yet expressive:

- 1) A runtime that interprets the intent and orchestrates application execution, measurement, control, configuration, and learning.
- 2) A library used to specify the following: measures, what the runtime should monitor; knobs, the variables that can (re-)configure the running application; and the controllable code, e.g., the video encoder's main loop.
- 3) An intent specification language.

Modifying applications for adaptation using Proteus requires only a few lines of library code and a declarative intent specification, whose syntax encodes the goals and constraints:

```
intent x264 max(quality) such
that performance == 30
```

and specifies the configuration space that the learner explores and the controller manages:

```
knobs subme = [1, 2, 3, 4, 5, 6,
               7, 8, 9, 10, 11]
              reference 7
utilizedCores = [1, 2, 3, 4]
                reference 4
...
```

Decoupling intent specifications from the application code has several benefits. It lends itself to flexible interpretation, e.g., using different optimization algorithms to solve the optimization problem they encode. Decoupling enables modification and extension of the intent specification without changing the application code, useful both during execution—where a change in the application's mission might be served by changing the intent—and during design—where changes to the intent do not require changing application code. Considering an application's entire life-cycle, decoupling also makes it possible to extend the intent specification language itself, with little or no change to the core application.

The need for an intent expression language becomes apparent when we consider possible extensions. For example, seen as an encoding of a mathematical programming problem, the intent may be naturally extended to support multiple constraints, which requires generalizing the constraint syntax to a logic. Applications with complex knobs also require more sophisticated syntax for expressing their configuration space. For example, assigning a value to some knob may only be meaningful when another knob has a particular value.

Proteus provides language and runtime support for developing adaptive applications

that meet goals in complex, dynamic environments. Proteus puts ML and control theory in developers' hands without requiring expertise in either field. Instead, developers focus on their application domains and specify their program's intents, making the development of adaptive, resilient software easier for the broader community of software developers. The Proteus language makes intent specification and adaptations first-class objects. The Proteus runtime uses those adaptations to ensure goals are met, which makes it convenient to set execution goals and update them on the fly. To showcase the practicality and ease-of-use of developing with Proteus, we have used it to control three complex, real-world applications. While this article motivates our approach through the case study of a video encoder, we have used Proteus components on many applications and systems.^{10–12} 

References

1. R. Laddaga, "Guest editor's introduction: Creating robust software through self-adaptation," *IEEE Intell. Syst.*, vol. 14, no. 3, pp. 26–29, May 1999.
2. J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.
3. B. H. Cheng, et al., "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*, B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin: Springer-Verlag, 2009, pp. 1–26.
4. J. L. Ozer, *Video Encoding by the Numbers: Eliminate the Guesswork From Your Streaming Video*. Galax, VA: Doceo Publishing, 2016.



SAEID BARATI is a Ph.D. student at the University of Chicago. His research interests include self-aware systems, approximate computing, resource scheduling and optimization, and embedded systems. Contact him at saeid@cs.uchicago.edu.



ROBERT CARTWRIGHT is a professor of computer science at Rice University. His research interests include program verification, program semantics, exact real arithmetic, and typing systems. Cartwright received a Ph.D. in computer science from Stanford University. He is a former member of the ACM Education Board and the CRA Board of Directors. He is a fellow of the ACM. Contact him at cork@rice.edu.



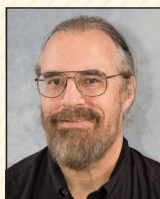
FERENC A. BARTHA was a research scientist at Rice University and is currently a research scientist at the University of Szeged, Hungary. His research interests include dynamical systems, resource-aware systems, and validated numerical methods. Bartha received a Ph.D. in mathematics from the University of Bergen, Norway. Contact him at barfer@math.u-szeged.hu.



ADAM DURACZ is a research scientist at the Rice University Center for Computing at the Margins. His research interests include modeling, simulation, verification, and learning of dynamical systems. Duracz received a Ph.D. in computer science from Halmstad University, Sweden, in 2017. Contact him at adam.duracz@rice.edu.



SWARNENDU BISWAS is an assistant professor in the Department of Computer Science and Engineering at the Indian Institute of Technology, Kanpur. His research interests include program analysis, concurrency and memory models, compilers and runtime systems, parallel computing, and approximate computing. Biswas received a Ph.D. from The Ohio State University. He is a member of the ACM. Contact him at swarnendu@cse.iitk.ac.in.



DONALD S. FUSSELL is the Trammell Crow Regents professor and chair of the Computer Science Department, director of the Laboratory for Graphics and Parallel Systems, a member of the Computer Engineering Research Center of the Electrical and Computer Engineering Department, and a fellow of the IC² Institute at the University of Texas at Austin. His research interests include computer graphics, computer games, computer architecture, and computer systems design. Fussell received a Ph.D. in mathematical sciences from the University of Texas at Dallas. Contact him at fussell@cs.utexas.edu.

5. VideoLAN Organization, “x264,” 2013. [Online]. Available: <http://www.videolan.org/developers/x264.html>
6. N. Siegmund, A. Grebhorn, S. Apel, and C. Kästner, “Performance-influence

- models for highly configurable systems,” in *Proc. 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 284–294.
7. J. Meinicke, C.-P. Wong, C. Kästner, T. Thüm, and G. Saake, “On

- essential configuration complexity: Measuring interactions in highly-configurable systems,” in *Proc. 31st IEEE/ACM Int. Conf. Automated Software Engineering*, 2016, pp. 483–494.



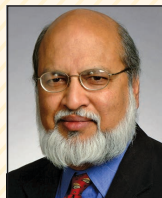
HENRY HOFFMANN is an associate professor of computer science at the University of Chicago. His research interests include self-aware computing systems. Hoffmann received a Ph.D. in electrical engineering and computer science from the Massachusetts Institute of Technology. He is a Member of the IEEE and the ACM. Contact him at hankhoffmann@cs.uchicago.edu.



NIKITA MISHRA is a researcher at the University of Chicago. Her research interests include developing machine-learning models to alleviate computing system performance, such as power management and computing resources prediction. Mishra received a Ph.D. in computer science from the University of Chicago. Contact her at nikitamishra07@gmail.com.



CONNOR IMES is a computer scientist at the University of Southern California Information Sciences Institute. His research interests include self-aware/goal-oriented computing, fault tolerance, and long-lived systems. Imes received a Ph.D. in computer science from the University of Chicago. Contact him at cimes@isi.edu.



ARVIND is the Johnson Professor of Computer Science and Engineering at the Massachusetts Institute of Technology, where he is a member of the Computer Science and Artificial Intelligence Laboratory. His research interests include parallel systems and software, high-level hardware synthesis languages, and enabling rapid development of embedded systems. Arvind received a Ph.D. in computer science from the University of Minnesota, Minneapolis. He is a Fellow of the IEEE and the ACM and a member of the National Academy of Engineering. Contact him at arvind@csail.mit.edu.



JASON E. MILLER is a research scientist at the Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology (MIT). His research interests include parallel processor and system architectures, architectural simulators, and adaptive self-optimizing systems. Miller received a Ph.D. in computer science from MIT. Contact him at jasonm@csail.mit.edu.



DUNG NGUYEN is a research scientist at Rice University. His research interests include software engineering with special interests in object-oriented modeling, design patterns, and programming. Nguyen received a Ph.D. in mathematics from the University of California, Berkeley. Contact him at dxnguyen@rice.edu.

8. P. Jamshidi, N. Siegmund, M. Velez, C. Kästner, A. Patel, and Y. Agarwal, “Transfer learning for performance modeling of configurable systems: An exploratory analysis,” in *Proc.*

32nd IEEE/ACM Int. Conf. Automated Software Engineering, 2017, pp. 497–508.

9. H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M.

Rinard, “Dynamic knobs for responsive power-aware computing,” in *Proc. 16th Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 2011, pp. 199–212.



KRISHNA V. PALEM is the Ken and Audrey Kennedy professor of computer science at Rice University. His research interests include embedded and nanoscale computing, with emphasis on ultralow energy systems through inexactness. Palem received a Ph.D. in electrical and computer engineering from the University of Texas at Austin. He is a Fellow of the IEEE, ACM, and AAAS. Contact him at palem@rice.edu.



ANDREW WRIGHT is a Ph.D. student at the Massachusetts Institute of Technology (MIT). His research interests include flexible processor design, accelerators, and processor specification and verification. Wright received an S.M. in electrical engineering and computer science from MIT. He is a Student Member of the IEEE. Contact him at acwright@mit.edu.



YAN PEI is a Ph.D. student at the University of Texas at Austin. His research interests include high-performance computing, control theory, and approximate computing. Pei received a B.S. in electrical engineering from Shanghai Jiao Tong University, China. Contact him at ypei@cs.utexas.edu.



YAO-HSIANG YANG is a Ph.D. student at Rice University. His research interests are formal semantics and statistical machine learning. Yang received an M.Sc. in computer science from National Taiwan University, Taipei City. Contact him at yy45@rice.edu.



KESHAV PINGALI is the W.A. "Tex" Moncrief chair of grid and distributed computing in the Institute for Computational Engineering and Science and a professor in the Department of Computer Science at the University of Texas at Austin. His research interests include programming languages, compilers, and runtime systems for multicore and manycore processors. Pingali received an Sc.D., E.E., and S.M. from the Massachusetts Institute of Technology. He is a Fellow of the IEEE, ACM, and AAAS. Contact him at pingali@cs.utexas.edu.



SIZHUO ZHANG is a Ph.D. student at the Massachusetts Institute of Technology (MIT). His research interests include processor design, memory system, and accelerators. Zhang received an M.S. from MIT. He is a Student Member of the IEEE and ACM. Contact him at szzhang@csail.mit.edu.



RYUICHI SAI is a Ph.D. student at Rice University. His research interests include programming languages, compiler construction, and software engineering. He received an M.Sc. degree from University of Houston. Contact him at ryuichi@rice.edu.

10. X. Sui, A. Lenharth, D. S. Fussell, and K. Pingali, "Proactive control of approximate programs," in *Proc. 21st Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 2016, pp. 607–621.
11. N. Mishra, C. Imes, J. D. Lafferty, and H. Hoffmann, "CALOREE: Learning control for predictable latency and low energy," in *Proc. 23rd Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 184–198.
12. C. Imes, D. H. K. Kim, M. Maggio, and H. Hoffmann, "POET: A portable approach to minimizing energy under soft real-time constraints," in *Proc. 21st IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015, pp. 75–86.
13. A. Filieri, et al., "Control strategies for self-adaptive software systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 11, no. 4, pp. 1–31, Feb. 2017.
14. H. Hoffmann, "JouleGuard: Energy guarantees for approximate applications," in *Proc. 25th Symp. Operating Systems Principles*, 2015, pp. 198–214.



IEEE COMPUTER SOCIETY
DIGITAL LIBRARY

Access all your IEEE Computer Society subscriptions at
computer.org
/mysubscriptions

IEEE Annals of the History of Computing

From the analytical engine to the supercomputer, from Pascal to von Neumann, from punched cards to CD-ROMs—*IEEE Annals of the History of Computing* covers the breadth of computer history. The quarterly publication is an active center for the collection and dissemination of information on historical projects and organizations, oral history activities, and international conferences.

www.computer.org/annals