The Case for Compulsory Approximation

Adrian Sampson
Cornell & Microsoft Research

Abstract

Approximation is a fundamental concept in some application domains. In the next phase of research on approximate computing, the community should absorb lessons and constraints from these fields with *compulsory approximation*. This essay (1) surveys domains with compulsory approximation; and (2) advocates for research that builds new abstractions for old approximations.

1. Approximate Computing's Adolescence

Research on approximate computing has aged out of its infancy. We have an initial slate of approximate hardware designs, new compiler optimizations for approximation on current hardware, and language tools to control accuracy–efficiency trade-offs. But the field has taken only tentative steps toward broader impact: vexing but valid concerns, like the feasibility of quality guarantees, still hinder widespread adoption of approximate systems.

As approximate computing begins to mature, the community should reflect on its strategy. Here are two reasonable visions for the next phase of approximation research:

- *Status quo.* We keep developing general-purpose techniques for approximation and quality control. If we push the quality–efficiency trade-off far enough for enough benchmarks, industry will eventually adopt approximation. This direction involves only shallow engagement with individual application domains.
- *Technology transfer via case study.* Researchers embed with specific application domains where we know approximation can be effective. We learn what it will take to "sell" our favorite techniques, from neural accelerators to stochastic logic circuits, to experts in those domains. These deployments will serve as case studies to inform and encourage broader adoption.

This essay advocates for a third, less obvious strategy to complement these directions. Instead of developing new approximation strategies from whole cloth and *then* working to apply them, the community should go to where approximations are already widespread. In domains where approximation is a fact of life, not an optional luxury, there is no "selling" necessary: approximate computing is already deployed. We call these instances of approximate computing *compulsory approximation*.

This essay makes the case for hunting approximation ideas in the wild. We should bring techniques from domains with compulsory approximation into the approximate-computing fold. The potential benefits are twofold:

1. The domains themselves stand to benefit from new abstractions for approximations they already use. Even established domain-specific approximation strategies can be *ad hoc* and difficult to control; the techniques and tools we have developed in the approximate-computing community can help make them more principled.
2. Approximate computing as a whole can benefit from expertise that is currently locked away within application domains. These domains are older and more mature than *approximate computing* as a buzzword, so they have long contended with problems that our community is only beginning to recognize.

We enumerate examples of domains with compulsory approximation and suggest ways that our community can engage with them.

2. Compulsory Approximation Domains

This section surveys examples of compulsory approximation. We identify domain-specific approximations that (1) have something to teach us in the approximate-computing community; and (2) have problems that we can help address by applying approximation ideas.

2.1. Machine Learning

Machine learning's charter is to approximate problems that we cannot solve exactly—or even precisely define. And while machine learning and AI have undergone a full-scale revolution over the last five years, they can still be difficult to trust.

For example: deep-learning implementers use a technique called *dropout*, which randomly deletes neurons from networks during training [9]. Dropout appears to avoid overfitting, leading to a better training result—but it is hard to see exactly why it works or explain when it might go wrong. Similarly, *Hogwild!* is a technique for parallel stochastic gradient descent that ignores data inconsistency [6]; its sensitivity to the underlying machine's memory model is unclear. Summing up these implementation problems and more, a Google paper recently called machine learning "the high-interest credit card of technical debt": while ML can accomplish amazing feats quickly, the reasons for its success (and, eventually, failure) can be murky [7].

We can learn: Processes and policies for measuring quality and deciding when it is good enough to ship.

We can offer: Tools for expressing and enforcing quality requirements in the language, especially when composing learning components into larger systems.

2.2. Numerical & Scientific Computing

Floating-point numbers are the world's oldest and most widespread deployment of approximate computing. In scientific computing, the practice of bounding floating-point error is just as mature. The floating-point error problem has spawned an entire field of study, numerical analysis, and many textbooks devoted to the topic [1]. Numerical computing represents an extraordinarily thorough case study in high-overhead, manual approaches to deriving strong accuracy properties.

We can learn: Mathematical tools for deriving hard error bounds when the error model resembles floating-point rounding. The same

techniques may not generalize to other approximation strategies—random bit flips or neural accelerators, for example—but they can still be valuable when errors are well behaved.

We can offer: Tools to automate tedious accuracy analyses and transformations that currently require experts. Panchekha et al.'s Herbie tool for fixing numerical stability problems is an important first step [5].

2.3. Real-Time Graphics

In games and other real-time graphics applications, everything is a compromise. Fundamentally, GPU-accelerated rasterization is a faster but worse alternative to full ray tracing. Game engines will go to extremes to maintain a smooth frame rate by simplifying scenes wherever possible.[1] *Level-of-detail* (LOD) techniques simplify objects for quick-and-dirty rendering when they are rendered in the background of a complex scene [2].

We can learn: Low-overhead techniques for tuning accuracy at run time in response to resource demands. The soft real-time constraints in game engines require them to adapt to changing conditions; the same capability should apply in other domains.

We can offer: Tools for navigating the huge space of individual approximation decisions that game developers typically implement.

2.4. Communications

Wireless communication channels are fundamentally noisy, so protocols and circuits for communication are forced to contend with random errors.

We can learn: Information-theoretic tools for quantifying a program's vulnerability to probabilistic error models. This is a counterpoint to numerical analysis, where the target is *deterministic* errors.

We can offer: New circuit-level approximation strategies that match the error models that communications hardware can already tolerate. Some work already proposes information-theoretic approaches to approximation that fits this mold [8].

3. Why Bother?

Systems fields in computer science—architecture, compilers, programming languages, operating systems, and so on—-have a strong bias in favor of generality. Research that benefits one narrow domain limits its impact and, therefore, must meet a higher bar for publication. So why should our community bother studying domain-specific, compulsory approximation techniques?

In the long term, we stand to gain insights from studying compulsory approximation that will not come from "voluntary" approximation research. This section describes three potential benefits from studying compulsory approximation.

3.1. A Benevolent Trojan Horse

Exploiting existing approximation strategies, as they exist in the wild, means working within an application domain's constraints. For example, a project could apply language abstractions inspired by Chisel [4] to express error resilience in wireless protocols and their implementations. This strategy should be easier than importing an unfamiliar, off-the-shelf approximation technique such as low-refresh DRAM [3]. Meanwhile, it offers a foothold into a domain that might otherwise resist an approximate-computing invasion.

The initial phase will help us, the approximation community, understand the domain's requirements for adopting new approximation techniques. Are random faults OK, or are deterministic errors more digestible? What kinds of guarantees are important, and what can be left to testing? Armed with these insights, we will have a shot at a second phase, in which we transfer our own general approximation techniques by adapting them to meet the domain's expectations.

3.2. Our Achilles' Heel

A persistent problem in approximate computing is the need for an objective *quality metric*. Any evaluation section of an approximation paper hinges on quality metrics, and quality-monitoring tools can offer enforcement that is only as good as the metrics they target. But for many domains, we have little evidence that our quality metrics are meaningful: for example, does PSNR really capture the essence of quality for audio signal? For some applications, such as games, objective metrics may not exist at all: domain experts use an "I know it when I see it" approach to evaluating quality.

By understanding how compulsory approximations operate in domains today, we can develop a better understanding for how developers think about quality. The community should use these concepts to design new workflows for approximate computing—ideally, workflows that sidestep the need for *a priori*, objective, per-application quality metrics.

3.3. I Came, I Saw, I Generalized

The domains surveyed in this essay are mature fields: they have developed their approaches to compulsory approximation over decades. There is a reasonable chance, therefore, that they contain well-developed ideas that go beyond what we have invented in our application-agnostic community.

We should harvest approximation and quality-control ideas from these domains. There is no shame in stealing established techniques from other fields if we can show that they generalize in surprising ways to other applications.

4. Call to Action

This essay is not meant to discourage continued research on general-purpose, benchmark-driven approximation techniques; that strategy is important too. But our field needs to grow up. We need to counter the popular hobbyhorse that domain-specific approximation is enough, so *approximate computing* as an independent concept is doomed. Let's take the next big step toward broader impact by embracing compulsory approximation as part of the approximate-computing family.

References

[1] Richard L. Burden, J. Douglas Faires, and Annette M. Burden. *Numerical Analysis: 10th Edition*. 2015. ISBN 1305253663.

[2] Yong He, Tim Foley, Natalya Tatarchuk, and Kayvon Fatahalian. A system for rapid, automatic shader level-of-detail. In *SIGGRAPH Asia*, 2015.

[3] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. Flikker: Saving refresh-power in mobile devices through critical data partitioning. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[4] Sasa Misailovic, Michael Carbin, Sara Achour, Zichao Qi, and Martin C. Rinard. Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2014.

[5] Pavel Panchekha, Alex Sanchez-Stern, James R. Wilcox, and Zachary Tatlock. Automatically improving accuracy for floating point expressions. In *ACM Conference on Programming Language Design and Implementation (PLDI)*. ACM, 2015.

[6] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[1] This video demonstrates "the cost of 60 frames per second" in the game *Halo 5: Guardians*. `https://youtu.be/-gQMulb6T2o`

[7] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. Machine learning: The high interest credit card of technical debt. In *SE4ML: Software Engineering for Machine Learning*, 2014.

[8] Naresh Shanbhag. Statistical information processing: Computing for the nanoscale era. In *International Symposium on Low Power Electronics and Design (ISLPED)*, 2015.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15 (1): 1929–1958, January 2014.