Research Challenges for Approximate Computing

Brian Demsky, University of California, Irvine

Achieving a 10-100x performance improvement by trading off accuracy will require revisiting many of the fundamental assumptions of the current computing stack and in particular the hardware-software interface.

For example, if we hope to achieve improved performance by tolerating non-deterministic errors, *what is the abstraction through which errors are exposed to software?*

This question could have a number of different answers. For example, hardware could expose errors as certain "efficient" operations that return approximately correct results. While this certainly represents a new challenge in software development, it seems tractable and researchers have already made significant progress on this problem.

Unfortunately, it remains unclear that much power can be saved by allowing errors in just the data path of programmer-visible operations. Published results suggest that at least for earlier generations of processors, power consumed by control structures represents a dominant portion of the overall power budget. In any case, Amdahl's law suggests that power savings will quickly be limited by control structures.

It is of course possible to achieve greater power savings by building processors in which errors can occur in program control, decoding instructions, and in instruction and data caches. It appears extremely difficult for software to tolerate such errors in their full generality. However, if these errors are predictable or new mechanisms are provided to manage such errors, it may be possible.

Given the power consumed by control structures in current processors, we may need to explore *what computational models are well suited for error-tolerant computing?* In particular, fully realizing the potential power savings will likely require new computational models in which control can be hard-wire or the power it consumes drastically reduced. These changes will likely require new or redesigned programming languages that are better matched to the new computation models.

Finally, we need to better understand *how approximation affects computations and develop tools that enable us to explore, understand, and control these effects.* In general, we can expect these tools to fall into two general categories — (1) tools that allow us to evaluate whether approximate answers are close enough and (2) tools that provide hard guarantees about the propagation or isolation of errors. Monte-Carlo simulation and/or probabilistic reasoning systems can help us understand whether approximate answers are good enough.

In other cases, we may need hard guarantees that limit the affects of errors. One potential issue is that some expressions of a computation may tend to accumulate errors in such a way that after some time period the results become unacceptable. In some cases this may be intrinsic to the computation being performed, but in other cases there may be alternative expressions of the computation that avoid these issues.

A potential tool for constraining the effects of errors is self-stabilization — a guarantee that a computation will eventually reach the "correct" state even in the presence of errors. In the context of many computations, this can be as simple as the guarantee that erroneous values must eventually leave the computation. In the context of the Java programming language, we have developed static analysis that can verify self-stabilization. Our results show that certain control systems and media decoders do in fact self-stabilize.