Position Statement for Panel on GRAND CHALLENGES IN EMBEDDED SOFTWARE

Edward A. Lee
Department of EECS
University of California, Berkeley
Berkeley, CA 94720, USA
eal@eecs.berkeley.edu

Abstractions currently used in computing hide timing properties of software. As a consequence, computer scientists have developed techniques that deliver improved average-case performance and/or design convenience at the expense of timing predictability. For embedded software, which interacts closely with physical processes, timing is usually an essential property. Lack of timing in the core abstractions results is brittle and non-portable designs. Moreover, as embedded software becomes more networked, the prevailing empirical test-based approach to achieving real-time computing becomes inadequate.

I believe it is necessary to reintroduce timing predictability as a first-class property of embedded processor architectures. Architectures currently strive for superior average-case performance that regrettably ignores predictability and repeatability of timing properties. "Correct" execution of a C program has nothing to do with how long it takes to perform any particular action. C says nothing about timing, so timing is not considered part of correctness. Architectures have developed deep pipelines with speculative execution and dynamic dispatch. Memory architectures have developed multi-level caches and TLBs. The performance criterion is simple: faster (on average) is better.

The biggest consequences have been in embedded computing. Avionics offers an extreme example: in "fly by wire" aircraft, where software interprets pilot commands and transports them to actuators through networks, certification of the software is extremely expensive. Regrettably, it is not the software that is certified but the entire system. If a manufacturer expects to produce a plane for 50 years, it needs a 50-year stockpile of fly-by-wire components that are all made from the same mask set on the same production line. Even a slight change or "improvement" might affect timing and require the software to be re-certified. All users of embedded software face less extreme versions of this problem. Upgrading an engine controller in a car to a newer microprocessor, for example, often requires substantial redesign of the software and thorough retesting. Even "bug fixes" in the software can be extremely risky, since they can change timing behavior.

Designers have traditionally covered these failures by finding worst case execution time (WCET) bounds and using real-time operating systems (RTOS's). But these require substantial margins for reliability, and ultimately reliability is (weakly) determined by bench testing of the complete implementation. Moreover, WCET has become an increasingly problematic fiction as processor architectures develop ever more elaborate techniques for dealing stochastically with deep pipelines, memory hierarchy, and parallelism.

The reader may object that there are no true "guarantees" in life, so the correct solution should be to accept timing variability and to build in robustness. However, synchronous digital hardware—the technology on which most computers are built—can deliver astonishingly precise timing behavior with reliability that is unprecedented in any other human-engineered mechanism. Software abstractions, however, discard several orders of magnitude of precision. Compare the nanosecond-scale precision with which hardware can raise an interrupt request to the millisecond-level precision with which software threads can respond.

To fully exploit such timing predictability would require a significant redesign of much of computing technology, including operating systems, programming languages, compilers, and networks. I believe we must start by creating a new generation of processors whose temporal behavior is as easily controlled as their logical function. We call them precision timed (PRET) machines [1]. Our basic argument is that real-time systems, in which temporal behavior is as important as logical function, are an important and growing application; processor architecture needs to follow suit.

Of course, timing precision is easy to achieve if you are willing to forgo performance; the engineering challenge in PRET machines is to deliver both precision and performance. In [1], we argue that the problem should be first tackled from the hardware design perspective, developing precision timed (PRET) machines as soft cores on FPGAs. The near term goal would be that software on PRET machines be integrated with what would traditionally have been purely hardware designs. This provides a starting point for a decades-long revolution that will make timing predictability an essential feature of computing.

1. REFERENCES

[1] S. A. Edwards and E. A. Lee. The case for the precision timed (PRET) machine. In *Design Automation Conference (DAC)*, San Diego, CA, 2007.