Rethinking Hardware (and Software) for Disciplined Parallelism

Sarita V. Adve, Professor, Department of Computer Science, University of Illinois at Urbana-Champaign.
Area: "Home and business computing," but the position below applies to other platforms as well.

The problem: A fundamental obstacle to popular parallel programming is the lack of robust, general-purpose parallel programming models. Current models are not only conceptually more difficult to understand than the sequential model, but require abandoning decades of advances at the core of robust sequential software engineering practices (e.g., safety, modularity, composability). Threads-based shared-memory, arguably the most widely used model, is known to be difficult to program, debug, and maintain due to data races, ubiquitous non-determinism, etc. Even the most fundamental property of "what value a shared-memory read may return" (a.k.a. the memory model) is notoriously difficult to specify - recent efforts on language memory models have produced very complex semantics (Java) or abandoned key safety and security features (C++) [1]. We claim that building our systems on a foundation where it is so difficult to specify even this fundamental property is at best questionable. We take the position that we must rethink the way we both express and implement parallelism from the ground up, and together for both software and hardware.

The opportunity: Pessimists might abandon shared-memory; we believe the problem is not with shared-memory per se (which has well-known advantages), but in our models allowing undisciplined, "wild" shared-memory interactions. Some may argue that the programming model should be left to our software colleagues; however, the memory models mess today could partly be attributed to such a software-oblivious hardware approach. We propose that a key part of a future architecture research agenda should be centered on work with software colleagues to co-design (1) general-purpose disciplined parallel programming models; (2) disciplined and expressive software and hardware interfaces for these models; and (3) efficient implementations that fully exploit the information in these disciplined models and interfaces. Without such a co-designed effort, architects will arguably continue to design for wild shared-memory that will be hard to program, difficult to scale, and wasteful in complexity, performance, power, and resiliency resources.

Possible approaches: There is growing momentum to provide disciplined shared-memory models; e.g., guaranteed data-race-freedom; determinism; disciplined non-determinism (with transactional variants); many of which restore modularity and safety attributes from the sequential world. While there is increasing consensus on these high-level properties, the disciplined parallel interfaces at different system levels (application program, middleware/OS, hardware ISA) to enable these properties are still an open question. Recent work includes hardware or runtime mechanisms, whole-program compile-time analysis, and/or programmer annotations, with different implications on efficiency, functionality, and flexibility at different system layers. We believe architects should play a major role in the evolution of these interfaces at all system levels, to ensure both hardware efficiency and to play a central role as enablers of these disciplined models. There are numerous possibilities for rich lines of potentially paradigm-changing research; we highlight a few to illustrate this is not just a pie-in-the-sky vision.

Taking a bottom-up view, we believe that the parallel hardware interface must enable expressing structured parallel control (e.g., nested fork-join vs. arbitrary threads) and explicitly express side effects impacting parallel execution (Java/C++ already require explicit synchronization, but do not enforce correctness or capture data communication information). More explicit expression of (guaranteed) side-effects can allow hardware to implement more efficient communication and scheduling sub-systems; e.g., replace unscalable hardware coherence mechanisms for wild shared memory with "just-as-needed" software-driven communication strategies that exploit the disciplined model guarantees. Taking a top-down view, this also benefits high-level programmers – structured parallel control and explicit effects annotations in high-level software can address the issues of safety, modularity, composability, and maintainability. Modular compiler analysis coupled with such annotations can enable the above high-level semantic properties in a way that can be incrementally adopted in evolving software engineering practices (e.g., without requiring mass-scale migration to new models such as functional languages). Not surprisingly, the top-down and bottom-up views are entirely compatible, providing synergistic benefits and motivation for adopting such interfaces across all system levels.

While much research remains to be done, initial software ideas abound; e.g., region types and effects, ownership types and effects, serialization sets, transactional constructs, commutativity constructs; etc. Which will prevail and how to translate them into efficient hardware interfaces is unknown. A co-designed approach will ensure that architects design interfaces that are compatible with high-level software models; that these are exploited to ensure a future efficient scalability roadmap; and that hardware support is exploited to influence the evolution of high level disciplined models.

[1] S. V. Adve and H.-J..Boehm. Memory Models: A Case for Rethinking Parallel Languages and Hardware. To appear in *Communications of the ACM (CACM)*. http://rsim.cs.illinois.edu/~sadve/Pubs/10-cacm.pdf.