Challenges and Future Directions of Software Technology

The need for Explicit Programming Environments

Dr. John L. Gustafson, Director
Intel Labs
Before making sweeping statements...

Distinguish carefully between

• **Application-facing** software, and

• **Hardware-facing** software.
Application-Facing Software Challenges

Increasingly need to negotiate with the user about:
- Reliability (and repeatability)
- Precision
- Time to solution

Programmers need concessions, not demands for:
- 100% reliability, same answer every time
- Overkill precision everywhere, just in case
- Run instantly
An unpleasant math surprise lurks...

64-bit precision is looking long in the tooth. (gulp!)

At 1 exaflop/s ($10^{18}$), 15 decimals don’t last long.
It is unlikely that a code uses just the right precision.

- Too few bits gives unacceptable errors
- Too many bits wastes memory, bandwidth, joules
Ways out of the dilemma...

- Better hardware support for 128-bit, if only for use as a check.
- Interval arithmetic has promise, if programmers can learn to use it properly (not just apply it to point arithmetic methods).
- Increasing precision automatically with the memory hierarchy might even allow a return to 32-bit.
- Maybe it’s time to restore Numerical Analysis to the standard curriculum of computer scientists?
Hardware-Facing Software Challenges

- Finding concurrency in codes (something even single-chip programmers now need to do) isn’t the hard part. In fact, it’s already automated.
- Figuring out \textit{where} to put data is the unsolved problem. And the automated solutions have hidden assumptions that often don’t work well.

If you still teach the PRAM model, please stop.
Another unpleasant surprise lurking…

*No more automatic caches?*

- Hardware cache policies are designed to minimize *miss rates*, at the expense of low cache *utilization* (typically around 20%).
- Memory transfers are almost half the power consumed by a computer, and computing is now power-constrained.
- Software will need to manage memory hierarchies explicitly. And source codes need to expose memory moves, not hide them.
Speculative execution tricks to make programming easier may be on the way out; they reduce *performance per watt*.

As a result, application-facing programmers will have to learn to economize precision.

Hardware-facing programmers will have to learn to manage data locality explicitly.