Productive Extreme-Scale Computing
via Common Abstract Machine Models, Programming Models, and Integrated Performance Modeling

Samuel Williams, Brian Van Straalen, Leonid Oliker
Lawrence Berkeley National Laboratory
SWWilliams@lbl.gov
The shared memory multicore era witnessed a graceful evolution of an abstract machine model common to x86, SPARAC, POWER, and various vector processors.

Allows programmers to maintain one (parameterized) model of a machine from one generation to the next.

Unfortunately, the reemergence of accelerators has resulted in many ad-hoc additions to this model.

Each generation of processors redefines the model and how programmers implement and contemplate program execution.
Rather than demanding users exploit additional address spaces or execution models, we need a common model that allows specialization or attributes be applied incrementally…

memory is cached/coherent by default, but programmers/compilers can allocate variables or regions that are private (not coherent) or resilient or fast (near/local) as they see fit.

code runs on the lightweight cores by default but can be offloaded to fat cores that implement the same functionality for sequential performance.
A common machine model does not imply productive portability (write code once and be able to simply compile/run it anywhere)

For that, we need a **common programming model**…

- needs to be **standards-based** (e.g. OpenMP)
- single implementation must run (and run correctly) on all machines, but doesn’t necessarily have to attain optimal performance
- Within this programming model, researchers could optimize for specific architectures or microarchitectures
  - architecture-specific intrinsics
  - parallelism/synchronization constructs, etc…
  - blocking for different cache sizes
  - selecting different algorithmic parameters

Failure to provide a common programming model demands we write different versions for different architectures just to use the machine.

**Some architectures will be deprecated** due to programming challenges rather than performance or cost.
A common machine model and programming model may provide portability and parallel (weak) scalability.

Unfortunately, there are no guarantees on processor efficiency (performance portability).

To quantify efficiency, one must quantify both:
- an upper bound to performance
- and observed performance

As part of SUPER, we are developing an automated Roofline Toolkit that will provide reasonable performance bounds.

Integrated performance monitoring could be used to quantify the time/data movement associated with each key routine or loop nest.

Programmers may focus their efforts on the regions of code that dominate the run time and where the observed performance departs from the Roofline bounds.