The return of logic

Vijay Saraswat
IBM TJ Watson Research Center

Exascale programming offers staggering challenges for programmer.

We propose a return to old ideas as the way forward. We assert that productive exascale programming models will be

- high-level (very close to the application domain),
- declarative (programs can be understood as assertions in the application domain),
- determinate (the result of the computation will not depend on the idiosyncracies of a thread scheduler)
- statically type safe (so that a large class of programming errors are caught at compilation time),
- implicitly concurrent (compilers aware of the application domain will generate efficient code for the scale of the machine at hand), and
- capable of exploiting semantics-based techniques (e.g. sketching, declarative debugging) for program development.

We propose that concurrent constraint programming (CCP) can offer the foundation for such a productive exascale programming model. CCP was proposed in 1987 as a framework for concurrency in which multiple activities communicate by posting and checking constraints on shared variables and has seen a significant amount of theoretical work in the intervening years.

A key characteristic of CCP is that it is parametric in the constraint system -- it can be adapted to different domains by simply choosing different constraint systems; the notion of programs, computations, correctness remain unchanged. Subsequent work extended CCP to support discrete time, continuous time, and also a notion of continuous space. Thus programs can be written and understood in CCP directly as (partial) differential equations. Most importantly, they can be debugged through completely novel techniques (algorithmic debugging, declarative debugging) that exploit the declarative reading of the programs.

Early implementations of CCP languages have been used in scientific and engineering applications (at Xerox PARC, NASA), in systems biology, and in combinatorial optimization problems.

With the maturation of the X10 programming model -- places, async, at, finish, clocks -- we believe we have the foundations for a practical, scalable implementation of CCP at exa-scale, building on top of the X10 tool chain.