Modeling and Simulation of Exascale Systems and Applications

Thomas Sterling, Matthew Anderson, Maciej Brodowicz
CREST, Indiana University

Asynchrony is a property of high performance computing that is emerging as one of the dominant factors in achieving efficiency and scalability as the field strives to achieve sustained exascale performance on real-world applications by the end of this decade. Asynchrony is the unpredictable variability of response time, either unbounded or bounded, for requests of resource access or functional services within and by computing systems. It imposes an uncertainty in the time domain of operation. This position paper summarizes a performance modeling and simulation approach to address several key challenges associated with exascale computing systems and, specifically, the aggravating effects of asynchrony when scaling to hundreds of millions of cores.

Asynchrony effects are and will be increasingly manifest in key ways that require active methods. Runtime-guided execution is one possible solution that has already been shown to alleviate the effects of asynchrony. Further, the fault response and energy control methods that will be needed for an exascale machine will introduce additional sources of asynchrony and complexity that are more easily addressed using a runtime system.

Coarse-grained simulation of a runtime system can serve to predict scaling trends in the presence of asynchrony as well as fault tolerance behavior, power usage, and performance even on hardware and at system scales that do not exist yet. While coarse-grained simulation of a runtime system models an application dataflow and some application characteristics of interest, it does not produce correct output from the application and typically runs orders of magnitude faster than actually running the full application itself. The approach involves creating a non-trivial manual implementation of an application that contains approximations appropriate for use in a coarse-grained simulator. This manual approximation of an application results in a so-called skeleton code. The accuracy of the prediction process is limited both by the accuracy of the manual implementation of the skeleton code and the ability of the coarse-grained simulator to properly replicate the runtime-guided execution behavior.

A robust way to model how active methods manage asynchrony to hundreds of millions of cores is enabling the runtime system to perform its own coarse-grained simulation. If the runtime system itself were capable not just of the runtime-guided execution of an application resulting in correct application output but also the coarse-grained simulation resulting in performance predictions, the need for a manually implemented skeletonized code would be eliminated. Moreover, any implementation mismatches between an independent coarse-grained simulator and the runtime system itself could be transparently addressed. For example, application developers might run their application at scale to verify correct output from the application. Then, they might wish to simulate the application performance at even larger scales on hardware that doesn't exist yet to produce performance predictions. The runtime system would support both modes of computation. This has the benefit of adding modeling tools directly to the runtime system to accelerate hardware and software co-design efforts.

There are several related efforts underway in the exascale simulation community. Each involves either trace generation or manual skeleton code generation. The CoDEx project sought to create a hardware/software codesign environment consisting of the ROSE compiler, the Green Flash Project, and SST/macro. In this environment, traces obtained and extrapolated via the ROSE compiler framework could be combined with the SST/macro event simulator and Green Flash simulator of node architectures. An intermediate step of potentially very large trace generation is required. Many-tasking runtime systems were not considered. SST/macro, however, has developed a macroscale simulator suitable for modeling many-tasking runtime systems. In SST/macro, all but the easiest applications require a manual skeleton code implementation. While there is an effort underway to produce compiler-based skeletonization, no automatic solution for skeleton code generation exists at present. The LogGOPSim simulator is another simulator capable of predicting performance at large scales using converted MPI traces as simulator input. This avoids the problem of developing of a skeleton code but adds a new problem: the MPI traces become so large when run at scale that efficient LogGOPSim simulation becomes impossible. Additionally, the LogGOPSim model has never been used for modeling many-tasking runtime systems.

Challenges addressed:

This approach eliminates the gap between runtime-guided application performance and runtime-guided application performance modeling. It brings modeling tools directly to the runtime system to accelerate hardware and software codesign. It eliminates the bottleneck of manually creating skeleton codes or generating large output traces for external event simulators.

Maturity:

There are several open source, mature, modern many-tasking runtime systems which are already fully instrumented with performance counters. The underlying components necessary for this approach are mature.

Uniqueness:

This approach is unique from all other simulator approaches in three ways: first, it does not require a skeleton code; second, it does not require trace generation; third, it adds crucial performance modeling capability to tools the application developers will already be using and be familiar with.

Novelty:

Achieving scalability in the presence of asynchrony for exascale computing will require both active methods and performance modeling capability for codesign. This approach is novel in that in provides both solutions within the same mechanism: the runtime system.

Applicability:

Runtime systems show broad applicability across all hardware and application domain spaces; the performance modeling aspects of a runtime system would enjoy the same broad applicability space.

Effort:

Fully instrumented runtime systems require few additional modeling tools in order to implement this approach. Application developers would require no additional effort other than application implementation for performance modeling.