

Fine-Grained, Event-Driven Runtime Research For Exascale Performance, Power, and Reliability

Rob Knauerhase, Intel Labs (rob.knauerhase@intel.com)

Kath Knobe, Intel Corporation (kath.knobe@intel.com)

The exascale research community has produced a well-known set of challenges that either complicate or break existing notions of application design, operating system/runtime facilities, and hardware architectures. Our recent research has investigated extreme-scale system software, and has led us to relatively unconventional approaches¹ to the role of operating systems and runtime environments for exascale HPC. With support from the DARPA UHPC program, we have developed an exemplar Intel Research Runtime (IRR) which implements a fine-grained event-driven execution model derived in part from prior work in dataflow programming and existing notions (both theoretical and practical) of very small “codelets” as an atomic unit of application work. We propose that the Council consider runtime research approaches comprehending:

- very fine granularity of tasks (codelet-style application decomposition)
- event-driven execution (derived from satisfaction of control-flow and data-flow dependencies)
- dynamic runtime observation (monitoring system behavior so the runtime can adapt)
- execution frontiers (specific runtime state maintenance derived from the above)

in its agenda and plans for future exascale programs.

Challenges Addressed: IRR is a foundational component of the Open Community Runtime², with which we intend to explore *cross-layer* approaches to resource control, communication, and scheduling. OCR will provide both an open-source “pluggable playground” for novel experimentation as well as a framework to allow broader adoption of the concepts and interfaces/codes that prove successful.

Beyond this, however, the research directions we propose also specifically address exascale performance and programmability, by allowing multiple points at which optimizations may be applied. For example, the fine-grained decomposition of HPC applications into small chunks whose execution is predicated by their input dependencies allows us to quickly change the mapping of work to compute resources with minimal costs for context-switching overhead and *not* having to postpone reallocations for a giant barrier in the application. In a like manner, we can optimistically position (or as appropriate, relocate) code and data to exploit power savings from simple proximity within the system. With runtime observation in software (and/or hardware), we can accommodate system state (load, environment, faults) and system policy (guidance about performance/power tradeoffs) with facile changes to the app-to-hardware mapping. Indeed, our approach allows the dynamic substitution of different but equivalent computational “chunks” in response to factors which cannot be known at design or compile time. Lastly, we envision a “tuning language” that would provide developer guidance into runtime behavior, specifying insight into both time- and space-locality which the runtime can use in determining how and where computations progress.

We also are exploring the notion of “execution frontiers” for adaptability, fault-tolerance, and reliability. As the runtime executes the codelets of an application, it implicitly keeps track of inputs, outputs, and dependencies. The current status of the program is reflected in this constantly-changing state (the execution frontier). For fault tolerance (checkpoint-restart), instead of a stop/snapshot/restart paradigm, we can trivially maintain this asynchronously evolving frontier; the application can be restarted from any preceding execution frontier. Rather than traditional rollback to a previously-defined “big

barrier” checkpoint, our continuous maintenance of our evolving state is barrier free and needs no support from the programmer. We can provide the appearance of checkpoint-continue processing (where the whole application never comes to a full stop) by applying checkpoint-restart independently to small parts of the application. Moreover, the user obtains these benefits without having to design and specify traditional pan-application checkpoints.

With an application that is hierarchically subdivided into regions, the execution frontiers can be used to support a variety of flexible execution policies, for example, remapping of partially executed regions of the application. In addition, regions designated by “sub-execution frontiers” can be run multiple times, either with the same algorithm (e.g. for hardware fault detection and accommodation), or different mathematical algorithms (variable precision, comparison for resiliency, or choosing the quickest result).

Maturity: Much of our enthusiasm for this agenda results from work we accomplished in our UHPC-supported Runnemed program. Our early experiments with codelet implementations of microbenchmarks, graph-traversal, and synthetic aperture radar indicate that the approach is at a minimum sufficiently equivalent to, with indication that it should be superior to, existing techniques.

Uniqueness: Our approach is unique to exascale systems in that it was literally born from “blank-sheet” co-design given only early exascale boundaries. We leveraged our experience in broader system software and HPC environments to identify specifically which ideas would stand up to the power concerns of exascale systems, while also explicitly accommodating performance and programmability. Because some of these challenges are shared by peta-scale supercomputing and “big data” cloud environments, there may be overlap with other research programs. However, in those areas, extreme scales bring about complications that do not arise and thus wouldn’t necessarily even be comprehended by other efforts.

Novelty: Our ideas of how to eliminate legacy OS overhead by upsetting traditional ideas – designing out unneeded or overly-expensive functionality and squeezing-down remaining services to their simple essences – differs greatly from contemporary HPC execution models as well as datacenter/grid/cluster approaches. While there has been previous work (in software and hardware) on dataflow systems, we believe that modern hardware and compilers are sufficiently advanced to allow re-evaluation of 1980’s dataflow in a contemporary environment. We also hope to discover the novel aspects of determining and *omitting* exascale-inappropriate dataflow concepts (e.g. not requiring or postponing theoretical progress guarantees and suchlike) to minimize the overhead of our runtime system.

Applicability: If successful, we hope to apply facets of our approach to high-core-count programming in other types of systems. For example, as processors move into hundreds of cores (viz. Intel’s Xeon™ Phi throughput-computing roadmap) and connected groups (clusters) of these manycore machines, we expect to apply the insights we gain for dynamic adaptation, programmability, and resiliency.

Effort: The interdependent, cross-layer nature of this approach will likely require input from a number of specialties – application areas, algorithm design, compiler technology, and system software, potentially along with hardware architects. In our other work, we have begun assembling diverse teams of industrial researchers and academics as needed to refine our ideas and validate through experimentation

We look forward to participating in various efforts of exascale research, and thank the Council for its solicitation of viewpoints and opinions in this space.

¹ “*For Extreme Parallelism, Your OS is Soooo Last Millennium*”, Knauerhase, Cledat, and Teller; in proceedings of Hot Topics in Parallelism, June 2012

² See also co-authored position paper “*Open Community Runtime: A Framework for Coöperative Resource Control in Exascale Systems*”, concurrently submitted to DoE OS/R Technical Council