

The Path to Exascale Computation: Adaptive Introspection ¹

Supercomputers are useful only if they enable better solutions to real problems.

Solving Exascale problems on Exascale systems is going to be much more challenging than even solving Petascale problems on Petascale systems. Power at the device level is the root cause of the difficulty. Power concerns have resulted in chips where computation is basically free and data movement is expensive. Many-core, GPGPUs and special purpose devices have put significantly more computation in chips without increasing memory bandwidth. Locality will be crucial to performance. Pushing to do more with fixed power budgets, has resulted in systems using a variety of processors. Distributing work throughout the resulting processor heterogeneity will also be critical. Since the individual threads are not significantly increasing in speed, to reach Exascale orders of magnitude more threads will be required. Identifying parallelism to drive Exascale systems will be difficult. Previously minor issues like reliability and resilience will be perhaps the greatest challenges to producing a usable Exascale system with the shear magnitude of components involved.

It's not just the HW that is becoming more complicated. Applications are becoming more detailed, more adaptive and more heterogeneous. The level of detail required changes across the application space and throughout execution, resulting in load balancing and locality issues. As the applications adapt during execution, flexible scheduling methods will be required. Mapping heterogeneous applications to heterogeneous hardware will be difficult from a correctness view point and even harder from a performance viewpoint. Both are required for Exascale applications.

All levels of the software stack will be involved in solving these problems. One size will not fit all. For some issues, like load balancing, failing at any level will result in poor performance. Other issues, like resilience, need to have the problem sub-divided and the solutions will need to be tailored at each level to manage cost and maximize effectiveness. Additionally an integrated software stack will be necessary to prevent solutions at different levels from interfering with one another. The key for solutions at all levels (and particularly to stop HW interference) is to have information available with the current state of the system.

Information will be the key to successful solution: information about the current state of the application; information about the current state of the HW; information about the current state of the rest of SW stack. Each piece of the SW stack will need information about all of the critical components to make informed decisions to maximize performance and reliability, both locally and globally.

At RENCi, we are advocating an integrated, dynamic, adaptive introspective approach to the entire software stack for Exascale systems. *Integrated* because if one level fails to solve a problem they all fail, the 'best' solution at a level will depend on what other levels are doing and the information to decide the 'best' solution is often most easily collected elsewhere. *Dynamic* because no static solution is going to consistently be best for applications that change over time, on chips whose performance changes over time, on sub-systems whose composition changes over time. *Adaptive* because the solution must vary with the application and HW. Changing internal and external loads will change the optimal balance between threads and resources. Dynamically adjusting will be required for consistently good performance. And *Introspective* because unless the software understands the current performance of the system it has no way of judging the relative performance of any solution. Recognition of any current system bottleneck will allow effective

¹Position Paper on OS and Runtime Software for Exascale Computing by Allan Porterfield (contact author: akp@renci.org – UNC/RENCi - 100 Europa Dr Chapel Hill NC 27517) and Rob Fowler (UNC/RENCi)

counter-measures.

We propose that Runtime/OS systems be built around a data collection mechanism, the **Blackboard**, so that everyone can see the system's current performance. Each tool logs its decisions and makes the data available that it collects to other components and reads information previously collected by other sub-systems. The blackboard can be organized as a group of single writer/multiple reader self-describing database sections. Each section has a single writer to eliminate locking issues, and is self-describing to facilitate changing the information stored over time. The runtime will use the information to decide how many and where to place threads to reduce bottlenecks and optimize performance. Scheduling decisions are multi-objective taking into account many factors including local performance, global performance and power requirements. The runtime's scheduler will interact with the OS and monitor the HW's to reach the best choices. The OS will also be able to use the information for its scheduling decisions. The OS can adjust job layout and to what extent resource are shared between jobs based on the current performance information. Perhaps most critical to effective Exascale execution. The runtime and OS can use information from the HW to proactively try to limit the impact of failing HW. Getting execution off failing HW before it completely fails has a very good cost/effectiveness ratio and will one (of many) techniques to enhance Exascale reliability.

Challenges addressed: The biggest challenge is providing a software stack that allows Exascale computers to solve the otherwise intractable problems. An integrated software stack based on a shared 'Blackboard', provides the information necessary to support adaptive, dynamic performance and reliability solutions. By providing continuously updated information at all levels of the HW/SW stack, dynamic, high performance, reliable applications will be possible.

Maturity: A second-generation prototype of the 'Blackboard' has been recently implemented based on Google Protobuf. It allows the runtime and performance tools to monitor HW for shared resource contention. An experimental Qthread's scheduler has been modified to adjust the number of threads up and down in response to contention. As an academic prototype it is usable for local testing and study, not ready for general release.

Uniqueness: Although some systems have used HW performance monitors to adapt performance, no one else is advocating a generally accessible performance database and building tools based on it information. Information available in the database will improve runtime and OS scheduling by allowing them to adapt to changing circumstances in either the system or the application.

Novelty: As the informatoin in the blackboard increases, it can drive not only novel runtime and OS scheduling policies but more importantly for Exascale, new proactive software resilience and reliability mechanisms. High performance and limited failures is required for completion of applications currently intractable.

Applicability: The blackboard provide information to the PE ina addition to the runtime and OS. This would drive future optimizations or autotuning. The information will also be of use to performance tuning, providing the programmer with information about what slowed performance and where in the application it occurred. The blackboard would be the information backbone for an integrated software environment from the PE, to the runtime, the OS and the HW.

Effort: The effort to develop a runtime/OS tightly integrated with a 'blackboard' can be significant. By allowing finer grain control of scheduling decisions, it is possible that major portions of the underlying systems need to be re-implemented. The benefit of the 'blackboard' structure is that its use can be phased in. The previous version of tools can co-habitat with the blackboard until their improve blackboard-aware versions are completed.