

# Introspective and Adaptive Runtime Systems

Laxmikant (Sanjay) Kale  
University of Illinois at Urbana-Champaign  
kale@illinois.edu

Exascale hardware and applications bring numerous new challenges to the fore. Programmers have to grapple with the increasingly difficult challenges associated with power, temperature, heterogeneity, latencies, transient and persistent load imbalances, and strong scaling. The magnitude of these challenges at exascale strongly argues for lesser manual control by the programmer, and a shift towards automatic resource management during execution. In fact, such automation may be critical to making application development at exascale a feasible and productive effort.

At such scale, applications will increasingly need the assistance of active, adaptive runtime systems that can absorb the burden of orchestrating execution. However, mainstream parallel programming paradigms like MPI are low-level by design, with runtimes that are passive on the resource management front. Research on introspective and adaptive runtime systems is essential for high performance and productivity.

To enable and empower adaptive runtime systems, it will be necessary to change the programming model and execution model so that it generates migratable and separate work units and data units. The number of such units need to be substantially larger than the physical resources (such as number of cores), creating a degree of freedom that an adaptive runtime system can exploit. Furthermore, to maximize the potential for adaptivity, the RTS should implement an asynchronous data driven execution model, instead of abiding by the constraints of the mainstream bulk synchronous SPMD approach. The RTS then manages when and where work is executed, and where data is placed. New research is needed on developing highly scalable resource management strategies that leverage this control to optimize application performance as the application executes and evolves. We next describe the challenges, opportunities and specific research issues such an approach entails.

*a) Introspection:* To be effective, the RTS must be able to observe application execution as it evolves. This involves inherent application characteristics possibly at the level of basic blocks and communication

events. Involving the RTS in scheduling and mapping of work/data units, as well as in mediating communication between them, ensures that it has the hooks to do the measurements needed. These measurements can be used in reactive as well as predictive manner depending on whether the *principle of persistence* applies (see below). In addition to monitoring application behavior, the RTS must also monitor its hardware substrate: it is expected that the hardware will provide multiple signals and data, as well as expose many control knobs. For example, hardware may provide data on core temperatures, frequencies of corrected failures in caches/memories, and may allow software control of frequencies and voltages. Providing a lightweight system to carry out such monitoring and maintain databases, while tailoring the measurements and the associated overhead to only the metrics needed in particular application phases, is a research issue that must be addressed [3].

*b) Principle of persistence:* Many, but not all, exascale applications will have iterative and/or time-stepping structure. In such applications communication and computation behavior tends to persist over time, because of the continuity of the underlying physical and numerical structures. This heuristic, when it applies, can be exploited to provide measurement based resource management strategies by the runtime system.

*c) Scalable load-balancing strategies:* Once RTS has a model of current application behavior, it needs to act from time to time to migrate work and data units so as to restore balance. The relatively coarse grained units enable much efficient balancing, compared with having to repartition the underlying (large) mesh or other user data structures. However, at exascale, even the number of such migratable units is very large. Novel strategies that avoid having to bring the communication graph on one processor, and yet carry out efficient repartitioning of objects, constitute another research issue [15]. Since balancing and associated migrations are expensive, continuous yet lightweight utility analysis to decide when to trigger load-balancing are also needed.

*d) Localized load adjustments:* At exascale, global load balancing steps will be proportionately more expensive, especially in view of the synchronization they bring about. Therefore, they will necessarily be infrequent. During the interim, one will need additional strategies to deal with load variations that arise. These can be transient (“noise”), arising from dynamic variations in the hardware environment [8], or in low-level software, or hardware-handled errors; But they can also be relatively slow-varying, arising from power-related issues (see below), or application evolution. Low overhead strategies for mitigating the impact of these are needed, which will exploit the presence of smaller-grained migratable tasks (work-units) to smooth over the “noise”, and use “within-neighborhood” migrations of tasks as needed.

*e) Other imbalances:* For applications such as combinatorial search, a different category of load-balancing systems are needed; also, for one-time algorithms such as some of the graph problems, persistence is not very useful; instead communication optimizations by the RTS are the main benefit a RTS can provide, along with the use of predictive or reactive load balancing [12].

*f) Communication:* The over-decomposition mentioned above helps reduce stress on communication networks by spreading communication over time to some extent. However, there is a complex trade-off between communication, the degree of over-decomposition, the degree of pipelining, and scheduling/communication overhead. The RTS is in the perfect position to observe, experiment and tune such degree of pipelining; strategies for such tuning are needed. In addition, new load balancers that are topology aware [15] and reduce the overall communication congestion, without requiring expensive global graph partitioning strategies.

*g) Collective operations:* will be challenging at exascale for several reasons. Implementing even simple collectives such as reductions in a situation when the entities contributing to reductions are spread over a subset of processors, and are migrating in response to runtime conditions, is itself challenging. Further, membership of the collection may be changing due to dynamic creation/deletion as the application evolves. We need non-blocking collective operations to avoid losing the use of processor during execution of the collectives. Forming collections (i.e. communicators in MPI parlance) without high memory requirements is another challenge. We need to develop algorithms for these operations; Further, these algorithms should be adaptively selected and tuned by the RTS, based on the evolving communication patterns of the application and the hardware characteristics.

*h) Power, energy, and thermal considerations:* The RTS can observe power-profiles of individual execution blocks, as well as core temperatures and power consumption rates. New strategies for effecting multi-dimensional optimizations between execution time and these metrics are needed. Preliminary work [13] has shown the potential for such strategies, in conjunction with the ability to dynamically balanced load.

*i) Heterogeneity:* It is expected that the exascale hardware will contain a heterogeneous mix of computing elements. Deciding how to map work units to physical resources Deciding how to apportion work between accelerator and host cores, and across different nodes, is a two-dimensional optimization problem that will be taxing for the programmer to handle explicitly at exascale. The RTS should automate this task, building up on the preliminary work such as. [7], [9], [10].

*j) Resilience:* This is a large topic, which should be addressed separately. We only note that an adaptive RTS will necessarily play an important role in resilience protocols at exascale [11], [16].

**Related work:** Concepts related to the proposed approach have been explored in Charm++ [5], [6], X10 [2], Chapel [1], ParalleX/HPX [4], QuickThreads [14] etc. The specific related work is cited above when feasible.

**Challenges addressed:** power/energy/temperature, load balancing, strong scaling requirements, heterogeneity, resilience, communication costs (as discussed above).

**Maturity:** Some capabilities of RTSs are demonstrated in Charm++, along with feasibility of building useful programming models with them. Exascale requires extending/scaling these ideas.

**Uniqueness (to Exascale):** The capabilities needed, as described above, are uniquely needed for exascale. Baseline, medium scale capabilities exist, but are not adequate to exascale because of the scaling limits of load balancers, new needs for stronger control of power/energy, etc.

**Novelty:** Although the overall approach inherits some DNA from ongoing work, the challenges outlined above arise only at exascale, and largely, have not been addressed in the runtime systems work so far.

**Effort:** To develop/demonstrate techniques in a research setting, for one programming paradigm, will require a medium size project for 3-5 years, preferably spread over multiple institutions. To make a truly universal introspective runtime system, applicable to multiple programming models (including, say, Charm++, ParalleX, extended Chapel/X10, ..) will require a multi-institution large project.

## REFERENCES

- [1] B. Chamberlain, D. Callahan, and H. Zima. Parallel programmability and the chapel language. *Int. J. High Perform. Comput. Appl.*, 21:291–312, August 2007.
- [2] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar. X10: an object-oriented approach to non-uniform cluster computing. In *OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 519–538, New York, NY, USA, 2005. ACM.
- [3] I. Dooley. *Intelligent Runtime Tuning of Parallel Applications With Control Points*. PhD thesis, Dept. of Computer Science, University of Illinois, 2010. <http://charm.cs.uiuc.edu/papers/DooleyPhDThesis10.shtml>.
- [4] G. Gao, T. Sterling, R. Stevens, M. Hereld, and W. Zhu. ParalleX: A study of a new parallel computation model. In *IEEE International Parallel and Distributed Processing Symposium, 2007*, pages 1–6, march 2007.
- [5] L. Kale, A. Arya, A. Bhatele, A. Gupta, N. Jain, P. Jetley, J. Lifflander, P. Miller, Y. Sun, R. Venkataraman, L. Wesolowski, and G. Zheng. Charm++ for productivity and performance: A submission to the 2011 HPC class II challenge. Technical Report 11-49, Parallel Programming Laboratory, November 2011.
- [6] L. Kalé and S. Krishnan. CHARM++: A Portable Concurrent Object Oriented System Based on C++. In A. Paepcke, editor, *Proceedings of OOPSLA'93*, pages 91–108. ACM Press, September 1993.
- [7] L. V. Kale, D. M. Kunzman, and L. Wesolowski. Accelerator Support in the Charm++ Parallel Programming Model. In J. Kurzak, D. A. Bader, and J. Dongarra, editors, *Scientific Computing with Multicore and Accelerators*, pages 393–412. CRC Press, Taylor & Francis Group, 2011.
- [8] V. Kale and W. Gropp. Load balancing for regular meshes on smps with mpi. In *Proceedings of the 17th European MPI users' group meeting conference on Recent advances in the message passing interface, EuroMPI'10*, pages 229–238, Berlin, Heidelberg, 2010. Springer-Verlag.
- [9] D. Kunzman, G. Zheng, E. Bohm, and L. V. Kalé. Charm++, Offload API, and the Cell Processor. In *Proceedings of the Workshop on Programming Models for Ubiquitous Parallelism*, Seattle, WA, USA, September 2006.
- [10] D. M. Kunzman and L. V. Kalé. Towards a framework for abstracting accelerators in parallel applications: experience with cell. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, USA, 2009. ACM.
- [11] E. Meneses, G. Bronevetsky, and L. V. Kale. Dynamic load balance for optimized message logging in fault tolerant hpc applications. In *IEEE International Conference on Cluster Computing (Cluster) 2011*, September 2011.
- [12] O. Sarood, A. Gupta, and L. V. Kale. Cloud Friendly Load Balancing for HPC Applications: Preliminary Work. In *International Workshop on Cloud Technologies for High Performance Computing at ICPP*, Pittsburgh, PA, USA, September 2012.
- [13] O. Sarood and L. V. Kalé. A 'cool' load balancer for parallel applications. In *Proceedings of the 2011 ACM/IEEE conference on Supercomputing*, Seattle, WA, November 2011.
- [14] K. B. Wheeler, R. C. Murphy, and D. Thain. Qthreads: An API for programming with millions of lightweight threads. In *International Parallel and Distributed Processing Symposium/International Parallel Processing Symposium*, pages 1–8, 2008.
- [15] G. Zheng, A. Bhatele, E. Meneses, and L. V. Kale. Periodic Hierarchical Load Balancing for Large Supercomputers. *International Journal of High Performance Computing Applications (IJHPCA)*, March 2011.
- [16] G. Zheng, X. Ni, E. Meneses, and L. Kale. A scalable double in-memory checkpoint and restart scheme towards exascale. Technical Report 12-04, Parallel Programming Laboratory, February 2012.