# Open Community Runtime:
# A Framework for Cooperative Resource Control in Exascale Systems

Vivek Sarkar, vsarkar@rice.edu [Contact author]
Barbara Chapman, chapman@cs.uh.edu
William Gropp, wgropp@illinois.edu
Rob Knauerhase, rob.knauerhase@intel.com
Tim Mattson, timothy.g.mattson@intel.com
Wilf Pinfold, wilfred.pinfold@intel.com

**Challenges Addressed:** Exascale systems will be qualitatively different from current and past HPC systems. Specifically, they will use massive numbers of multi-core processors with hundreds of homogeneous and heterogeneous cores per chip, their performance will be driven by parallelism and constrained by energy and data movement. They will also be subject to frequent faults and failures. Unlike previous generations of hardware evolution, these trends will have a profound impact on the foundations of future HPC software [5].

The design of Operating Systems and Runtimes for past HPC systems assumed a layered software architecture. Specific layers were isolated through well-defined interfaces allowing innovation within each layer to proceed independent of the other layers. The development of MPI, for example, enabled advances in the communication runtime layer to proceed independently of advances of thread scheduling and other forms of resource control in the operating system layer. Our position is that this form of layered architecture will not be effective for exascale Systems. A new approach based on cooperative resource control across multiple OS/R layers will be essential to to address the programmability, concurrency, energy, locality, and resilience challenges faced by exascale systems. In current approaches, each software layer has its own policies and mechanisms for managing resources such as processors/threads, memories, and network interfaces using abstract models that are oblivious to the presence of other layers sharing the same or related resources. The limitations of such approaches are manifest, for example, in the difficulties faced by attempts to combine scheduling and synchronization of MPI processes with scheduling and synchronization of intra-process threads and tasks. In contrast, effective solutions to cooperative resource control will enable combinations of OS/R components (communication, scheduling, synchronization, memory management, and resilience) to work together in ensuring that key resource constraints are satisfied *in a holistic manner*. Finally, the runtime system must play a central role in addressing programmability challenges by bridging the wide semantic gap between high-level programming models and the complexity of exascale hardware.

**Novelty:** We propose a research agenda comprehending the development of a shared framework for cooperative resource control that we call the Open Community Runtime (OCR). Extreme Scale computing is limited by concurrency, energy efficiency, and resiliency. The runtime system will play a critical role in exascale enablement for multiple reasons because the inherent variability in extreme scale software and hardware components calls for end-to-end asynchrony in system design, and tight integration of inter-node and intra-node parallel runtime systems has proven elusive thus far. The term “runtime” includes lightweight OS services that can be made available in user mode. Further, we expect that the lessons learned on runtime systems from this proposed research will also influence kernel design in future operating systems [14].

The framework will enable cooperative resource control by including first-class support for negotiation of resources across runtime services, and will allow for multiple implementations of common runtime APIs, perhaps with different trade-offs in different dimensions of resource usage. An exemplar of the OCR framework will be made available in open source form. Software projects that the authors of this position paper and their collaborators are involved with will serve as the starting point for development of OCR.

OCR will support innovations in programmability by providing a common substrate for researchers exploring a wide range of programming models. The authors of this position paper are interested in exploring a range of programming models on OCR that include Concurrent Collections [13,15], Habanero-C [13], MPI, and OpenMP. We will emphasize modern programming models (including future versions of MPI and OpenMP) that generate small units of work for computation and communication mapped by OCR onto coarser-grained system resources such as processors and network interfaces, while meditating the control of resources to ensure global constraints for resource usage and resilience are satisfied. Furthermore, since OCR will map onto a range of extremely scalable systems, it will provide a common software infrastructure underneath which hardware researchers can freely innovate.

There are many differences between our proposed approach and existing solutions. Past runtime systems have either been optimized for dynamic parallelism that is oblivious of locality (e.g., Cilk, OpenMP, Intel Thread Building Blocks) or for locality in the absence of dynamic parallelism (e.g., MPI, shmem, UPC). In contrast, OCR will support unbounded amounts of dynamic parallelism [2] with locality control [7] so as to be ``forward scalable'' to multiple generations of parallel hardware. MPI and PGAS programming models only express two levels of locality -- local and remote. HPCS languages such as Chapel and X10 mitigate this by allowing the programmer to express dynamic parallelism with ``locales'' and ``places'', but their locality model is also restricted to two levels. In contrast, OCR will support hierarchical place abstractions with arbitrary depths [4] that include support for heterogeneous accelerators [10,13]. Synchronization goes hand-in-hand with communication. Classical approaches to intra-node synchronization involved blocking of tasks at key events such as collective operations (e.g. reductions). In contrast, OCR will make these events as asynchronous as possible through the support of primitives such as codelets [9], data-driven tasks [11], phasers [1,3,6,8], and delegated isolation [12]. In summary, we are unaware of any existing solution that focuses on an open source runtime framework like the OCR while supporting a wide range of choices in programming models and system hardware.

**Uniqueness:** Components of OCR will be useful for sub-exascale systems, hence at the level of individual components, there is considerable overlap between OCR and current systems. We are unaware, however, of other research programs that could realize the integration across the runtime solution stack for exascale systems as planned for OCR. While the urgency for OCR is perhaps lower on sub-exascale systems since a single programming model (e.g., MPI+X) may suffice and energy constraints are less severe, it is critical to make OCR available to application developers soon so they can prepare their codes for the exascale era.

The resilience problem, i.e. the requirement that a system will continue to make progress in the face of failures of individual components, is less critical for sub-exascale systems. OCR will differ from established practice in current systems by directly addressing the resilience problems with fundamental research into power and performance efficient APIs that are resilient and enable a dynamic negotiation of resource sharing, as well as introspective self-observation to detect/anticipate and adapt to faults.

**Applicability:** While the emphasis is on exascale systems, components of OCR will be applicable to extreme scale systems at sub-exascale levels (e.g., departmental petascale, embedded terascale). Hence, hardware researchers (such as the co-authors on this proposal from Intel) responding to challenges raised by OCR could develop hardware innovations that would impact mainstream microprocessors. Likewise, the programming models enabled by OCR could also be helpful in addressing the very-high-core-count software challenges facing the computer industry.

**Maturity and Effort:** The authors of this proposal have served as key participants in exascale workshops, studies, and reports through which they have gained a shared understanding of the OS/R challenges facing exascale systems. This shared understanding has deepened through the lessons learned in the UHPC Runnemede project led by Intel. Further, the authors plan to leverage existing software developed in their groups or by their collaborators as a starting point for development of OCR (Habanero-C, Intel codelets, MPICH2 Nemesis, OpenUH runtime, qthreads, X10 distributed runtime). At the same time, the absence of any backward compatibility requirements for OCR ensures that it can focus on providing clear, exascale-oriented APIs enabling innovation without any legacy commitments. Thereby, it can provide flexibility in design, implementation, and schedule while its key results can remain applicable/portable to legacy programming models. We envision a core effort that is under 10PY/year for 3 years to realize the proposed OCR framework, with annual intermediate releases. The ultimate goal is for the effort to grow on a community-wide basis as researchers beyond the core team use the OCR for hardware and software exploration and also contribute additional components to OCR.

## References

[1] Phasers: a Unified Deadlock-Free Construct for Collective and Point-to-point Synchronization. Jun Shirako, David Peixotto, Vivek Sarkar, William Scherer. Proceedings of the 2008 ACM International Conference on Supercomputing (ICS), June 2008.
[2] Work-First and Help-First Scheduling Policies for Terminally Strict Parallel Programs. Yi Guo, Rajkishore Barik, Raghavan Raman, Vivek Sarkar. 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS), May 2009.
[3] Phaser Accumulators: a New Reduction Construct for Dynamic Parallelism. Jun Shirako, David Peixotto, Vivek Sarkar, William Scherer. 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS), May 2009
[4] Hierarchical Place Trees: A Portable Abstraction for Task Parallelism and Date Movement. Yonghong Yan, Jisheng Zhao, Yi Guo, Vivek Sarkar. Proceedings of the 22nd Workshop on Languages and Compilers for Parallel Computing (LCPC), October 2009.
[5] Software Challenges in Extreme Scale Systems. V. Sarkar, W. Harrod, A.E. Snavely. SciDAC Review Special Issue on Advanced Computing: The Roadmap to Exascale, pp. 60-65, January 2010.
[6] Hierarchical Phasers for Scalable Synchronization and Reduction. Jun Shirako, Vivek Sarkar. 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS), April 2010.
[7] SLAW: a Scalable Locality-aware Adaptive Work-stealing Scheduler. Yi Guo, Jisheng Zhao, Vincent Cavé, Vivek Sarkar. 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS), April 2010.
[8] Unifying Barrier and Point-to-Point Synchronization in OpenMP with Phasers. Jun Shirako, Kamal Sharma, Vivek Sarkar. 7th International Workshop on OpenMP (IWOMP), June 2011.
[9] Using a Codelet Program Execution Model for Exascale Machines. Stephane Zuckerman, Joshua Suetterlein, Rob Knauerhase, Guang Gao. 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era (EXADAPT '11), June 2011.
[10] Dynamic Task Parallelism with a GPU Work-Stealing Runtime System. Sanjay Chatterjee, Max Grossman, Alina Sbirlea, Vivek Sarkar. 2011 Workshop on Languages and Compilers for Parallel Computing (LCPC), September 2011.
[11] Data-Driven Tasks and their Implementation. Sagnak Tasirlar, Vivek Sarkar. Proceedings of the International Conference on Parallel Processing (ICPP) 2011, September 2011.
[12] Delegated Isolation. Roberto Lublinerman, Jisheng Zhao, Zoran Budimlic, Swarat Chaudhuri, Vivek Sarkar. Proceedings of OOPSLA 2011, October 2011.
[13] Mapping a Data-Flow Programming Model onto Heterogeneous Platforms. Alina Sbirlea, Yi Zou, Zoran Budimlic, Jason Cong, Vivek Sarkar. Conference on Languages, Compilers, Tools and Theory for Embedded Systems (LCTES), June 2012.
[14] For Extreme Parallelism, Your OS is Soooo Last Milennium". Rob Knauerhase, Romain Cledat, Justin Teller. USENIX Hot Topics in Parallelism (HotPar), June 2012.
[15] Concurrent Collections on Distributed Memory Theory Put Into Practice. F.Schlimbach, J.Brodman, K. Knobe. International European Conference on Parallel and Distributed Computing (Euro-Par), August 2012.

================= (cut off this page and following :) ===============

**Discarded brainstorms for titles (work some of these concepts into text)**
**Open Community Runtime --- a Framework for Cooperative, Adaptive, Cross-layer Exascale development**
**Open Community Runtime --- a Framework for Cooperative, Adaptive, Cross-layer extreme-scale systems**
**Correct Exposure: Managing cross-layer interactions required to achieve Exascale**
**Open Community Framework: runtime and "up/down interactions"(reword?) to enable extreme scale**
**Open Runtime Framework - a "sandbox" for exploring cross-layer exposure of app/system features**
**Extreme community Framework - pluggable runtime interfaces to explore vertical integraion (??)**
**Community Runtime Framework: cooperative APIs for dynamically-adaptive exascale systems**

**((TGM Writes)) my concern is all of this lacks a coherent perspective that makes it stand out. Change the names around and this could probably apply to any group thinking about exascale. What makes us different? I think the following captures the gist of what can make us stand out. ((END TGM Note))**

**Over much of the history of parallel computing, we have been able to use a layered architecture. <<insert picture with layers in a stack application, programming environment, runtime, OS, HW >>> . The interfaces between layers were essential. A group picked the layer they cared about, worried about the interfaces above and below, and largely ignored the rest.**

**In the exascale era, this model breaks down. Instead of a layered architecture we have a richly connected graph of components <<insert picture>>. This is the deeper meaning behind the focus on codesign for exascale systems.**

**The solution is to manage the complexity by exposing not just the interfaces to the different components, but to define how the components fit toether and interact. For example, the applications programmer must understand the programmiong model, how it interacts with teh runtime and how that runtime maps the execution onto hardware. These details must all be exposed in a way our applicaions programmers can manage.**

**This is in many ways the challenge of exascale.**

**<<< and there is more but I don't ahve time now. .. I need to join the call and after we talk abouit this and reach agreement, I can work on verbiuage later>>>**

**Two Page Position Paper Outline**

1. **(What is the challenge) [Mattson]**
   a. **No one programming model. Extreme scale likely to require composition of programming models. Example is scheduling of cores (MPI + OpenMP); extends to NICS (MPI + UPC); extends to power adaptation (no busy waits anywhere)**
   b. **Energy requirements and likelyhood of faults likely to require dynamic management of resources**
2. **(Why is this Exascale and not Petascale --- Uniqueness, novelty) [Knauerhase, Gropp]**
   a. **Petascale still manageable with a single programming model, though ad hoc MPI + X being used**
   b. **Energy can't be used to solve problems (there isn't any extra).**
   c. **Faults are likely (while not 1000x a petascale system, likely 10-100x a petascale system)**
   d. **Requires fundamental research into power and performance efficient APIs that are resilient and**

**enable a dynamic negotiation of resource sharing**

3. **Applicability**: To what extent will the proposed approach, if successful, be applicable to other areas?
    - Insights will help in many-core extreme scale computing
    - Some runtime components may be applicable
4. **(Why can this work -- Maturity, effort)**
    a. **Leverages existing work on runtime components, particularly for communication**
        i. **including successes from UHPC/Runnemede**
    b. **Provides clear, Exascale-oriented APIs allowing innovation (as well as short term solutions)**
    c. **Focus on energy efficient, scalable runtimes that enable negotiation between components separates design from implementation (want to say that schedule is more manageable)**
    d. **Effort**: How much effort is needed to effectively explore this approach?
    e. **Mention open source runtime components that will be used to start the effort: Habanero-C runtime, Intel Codelets (?), qthreads, Nemesis communication runtime in MPICH-2, OpenUH runtime(?)**

**0. Introduction (stolen wholesale from Wilf's email)**

Scientific computing has been characterized by long periods of incremental improvement punctuated by short periods of rapid change. The last period of rapid change saw the introduction on communicating sequential processing (CSP) and bulk synchronous scheduling. This period was tumultuous because a competition was set up between the vector supercomputers of the day and the upstart 'killer micros'. This competition was ultimately unproductive since the leading systems of today use the best of both vector and CSP.

We see a similar period of rapid change ahead [[RK - one line about multicore explosion as motivation?]] that will be precipitated by the demands of Exascale computing. These demands include an explosion in parallelism, an expectation that some processes will fail in every job, and rapidly increased algorithmic sophistication. The complexity of the next generation of supercomputers will be beyond the capabilities of even the most heroic programmer.

To both encourage the rapid change needed to achieve Exascale and minimize disruption to mission we envision a programming environment that supports current, developmental and future tools, languages, constructs and capabilities. We envision the heart of this environment to be an Open Runtime Framework that allows a researcher or programmer to develop a new tool, language, construct or capability that interoperates with other tools, languages, constructs and capabilities in the framework.

[[RK - Mention what people (we each) can contribute to OCR (existing work, leverage UHPC/Runnemede) and how that makes our effort different]]

1. **Background**

- Exacale challenges include parallelism, energy efficiency, and resiliency
- Important to use right runtime primitives as foundation for future programming models and compilers to address these challenges
- Inherent variability in exascale software and hardware calls for end-to-end asynchrony in system design
- Tight integration of inter-node and intra-node parallel runtime systems has proven elusive thus far
- Many members of the HPC community believe that evolutionary efforts to extend runtime systems will break down for Exascale systems and a new approach is needed

**2) Approach**

We propose the development of an Open Collaborative Runtime (OCR) system that:

- is representative of execution models expected for future extreme scale systems
- can be used to obtain early results to validate new ideas
- is made manifest as an open-source project

- allows substitution of proprietary components when so desired
- delivers end-to-end performance that is "good enough"

Scope of OCR includes:

- Task scheduling with adaptive control (scheduling policies for locality, energy, data/task migration)
    - [RK] Observation mechanisms (hw and sw) in support of adaptive control, especially for energy-efficiency and resiliency
- Intra- and inter-node memory management
- Intra- and inter-node synchronization
- Intra- and inter-node communication (including global address space)
- Concurrent data structures (includes atomic variables)
- Checkpointing interfaces
- Parallel I/O interfaces
- Tools interfaces
    - [RK]differentiating from compiler-backend optimizations where needed (for portability)
- Compiler interfaces
    - [RK]Supporting C/Fortran but also non-traditional languages/expressions (e.g. Concurrent Collections)

Motivation:

- Address revolutionary challenges collaboratively
- Reduce duplication of infrastructure effort so as to improve productivity of new research and path-finding efforts in the community
- Support new programming models as well as interoperability with existing programming model

Issues include:

(a) effective sharing of runtime code (simplifying deployment of new models)
(b) coordination for shared resources (including cores, NIC, memory, threads)
(c) need for extensibility - different programming models may have unique needs
(d) leveraging existing knowledge (various current runtimes; known issues)
(e) provisioning for future needs, including power management, fault/resilience, more dynamic scheduling/use of resources, overlap of communication and computation
(f) consistent and precise semantics (esp for one-sided operations and for memory models)
(g) expose resource and schedulers to support composition of parallel modules potentially written using different programming models.

Forward Steps

- Extensible framework
- Forum for developing standards for semantics, resource negotiation
- Exemplar framework allowing multiple groups to contribute and use

---

Call https://collab.mcs.anl.gov/display/exaosr/Call+for+Position+Papers