LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-474731

From Petascale to Exascale: Eight Focus Areas of R&D Challenges for HPC Simulation Environments

R. Springmeyer, C. Still, M. Schulz, J. Ahrens, S. Hemmert, R. Minnich, P. McCormick, L. Ward, D. Knoll

March 18, 2011

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

From Petascale to Exascale: Eight Focus Areas of R&D Challenges for HPC Simulation Environments

Programming Models

Patrick McCormick (LANL), lead, Richard Barrett (SNL), Bronis de Supinski (LLNL),
Evi Dube (LLNL), Carter Edwards (SNL), Paul Henning (LANL),
Steve Langer (LLNL), Allen McPherson (LANL)

Scope

Programming models bridge the gap between the underlying hardware architecture and the supporting layers of software available to applications. Programming models are different from both programming languages and application programming interfaces (APIs). Specifically, a programming model is an abstraction of the underlying computer system that allows for the expression of both algorithms and data structures. In comparison, languages and APIs provide implementations of these abstractions and allow the algorithms and data structures to be put into practice – a programming model exists independently of the choice of both the programming language and the supporting APIs.

Programming models are typically focused on achieving increased developer productivity, performance, and portability to other system designs. The rapidly changing nature of processor architectures and the complexity of designing an exascale platform provide significant challenges for these goals. Several other factors are likely to impact the design of future programming models. In particular, the representation and management of increasing levels of parallelism, concurrency and memory hierarchies, combined with the ability to maintain a progressive level of interoperability with today's applications are of significant concern.

Overall the design of a programming model is inherently tied not only to the underlying hardware architecture, but also to the requirements of applications and libraries including data analysis, visualization, and uncertainty quantification. Furthermore, the successful implementation of a programming model is dependent on exposed features of the runtime software layers and features of the operating system. Successful use of a programming model also requires effective presentation to the software developer within the context of traditional and new software development tools. Consideration must also be given to the impact of programming models on both languages and the associated compiler infrastructure.

Exascale programming models must reflect several, often competing, design goals. These design goals include desirable features such as abstraction and separation of concerns. However, some aspects are unique to large-scale computing. For example, interoperability and composability with existing implementations will prove critical. In particular, performance is the essential underlying goal for large-scale systems. A key evaluation metric for exascale models will be the extent to which they *support* these goals rather than merely *enable* them.

Assessment of Current Effort Within the Community

The most common approach used within the community today is a message passing based, single process, multiple data (SPMD) configuration. Less frequently used are task-parallel approaches, based on a multiple instruction, multiple data (MIMD) configurations. The vast majority of

implementations enable parallelism by using the Message Passing Interface (MPI) and codes written in a sequential programming language (C, C++, and FORTRAN). Parallelism within a many-core node is most commonly achieved using either an *MPI-everywhere* approach or a shared memory, multi-threaded implementation using OpenMP. Emerging options that address heterogeneous/accelerator architectures range from accelerator-centric techniques such as CUDA, OpenCL and vendor centric, directive-based approaches (e.g., PGI Accelerate and HMPP). Similar accelerator-centric extensions are being considered for adoption into the OpenMP standard. To date, these techniques have limited exposure in production-ready codes.

Other efforts within the broad community include programming languages that can be categorized as a partitioned global address space (PGAS), such as Co-Array Fortran (CAF), Unified Parallel C (UPC), and the DARPA-initiated, vendor-based, languages Chapel, Fortress and X10. Although these languages attempt to address many challenges that we face in moving to exascale, they have yet to establish themselves in terms of performance in complex applications and a variety of hardware architectures. Further, their designs have failed to consider (or only considered as an afterthought) key issues that face the ASC program, including a substantial investment in existing code that uses current programming models and, thus, invalidates assumptions of performing whole program analysis of applications written exclusively in the new language.

A fundamental concern in the path towards exascale is that the current generation of programming models, and their implementations, are based on assumptions of past architectures that are fundamentally different than likely exascale system designs. In particular, message passing can force replication of data, which could prove an excessive cost on many-core architectures with reduced memory capacities. Furthermore, current shared memory models do not support the expression of data locality that node-level performance will require. Nonetheless, the path forward for future programming models will likely combine models for both inter-node and intra-node computation.

Technical Challenges

The most obvious design goal for programming models is to support massive degrees of parallelism. Although some aspects of future programming models are likely to resemble those used on today's large-scale systems, those models and their implementations must support the expression and management of a significantly greater level of concurrency within a system that is composed of nodes consisting of processors with hundreds to thousands of cores. Capturing more parallelism will almost certainly require an ability to express multiple types and levels of parallelism. For example, we anticipate the need to support MIMD parallelism directly, rather than just enabling it. Further, the models must support manageable complexity to achieve the unprecedented level of parallelism. Thus, we expect future programming models to use a hierarchical representation of parallelism.

Another critical challenge is to support the heterogeneous processing environment of systems with both high-throughput and general-purpose cores that at least some large-scale systems will exhibit. Programmers will require mechanisms to express which code regions are most likely to perform well on which processor type. Further, those mechanisms must reflect that the decision is likely input (i.e., problem) dependent and may change throughout an application's execution. While this challenge is not completely unique to exascale programming models, we expect that the level of concurrent use of the diverse resources by a single application will be.

Locality is another issue that is not unique to exascale programming models. In particular, a programming model must capture the nuances of complex memory hierarchies and support

efficient data movement. Furthermore, it must support expression of complex relationships between data accesses and must allow easy mapping of those relationships into the hierarchical structure of the underlying hardware. For example, the simple two-level notion of locality (*here* and *not here*) of many PGAS languages is insufficient. Exascale programming models are distinguished from general future programming models in that the systems will be larger with a more complex physical structure, which will require more diverse mechanisms to express locality.

Performance is a first class exascale design goal; we cannot unconditionally surrender it for productivity. Thus, implementations must directly support performance analysis tools. We envision features that support the easy evaluation of how well the application uses the available hardware. Also, experience has shown that compilers rarely succeed in attaining the level of performance required for exascale applications in terms of percentage of peak performance. Thus, exascale programming models must support coding at a lower level when it is required to obtain the desired level of efficiency. While many languages already allow the use of assembly language; we envision very high-level exascale programming models that seamlessly allow the use of implementations of intermediate, lower level, programming models. This capability would also support the previously mentioned goals of interoperability and composability.

Exascale programming models may need to consider other critical issues for exascale systems beyond the above key challenges that exascale programming models must reflect. For example, exascale systems will consist of huge numbers of components so their reliability is expected to decrease significantly compared to current large-scale systems. Clearly, this change will require new approaches to resilience. Potentially, those approaches can be hidden from the application programmer through innovations in the system software and runtime libraries. However, we expect that better support in the programming model for algorithm-based fault tolerance and other resilience approaches will prove useful. Similarly, power and energy consumption will be key aspects of exascale systems. Again, this issue may be addressed transparently to the application. However, we expect techniques that allow the application to guide the underlying mechanisms will result in the best overall performance.

Widespread adoption will likely constitute the most important challenge for exascale programming models. Frequently changing features and/or implementations has the potential to significantly reduce productivity and the reliability of key applications. Thus, although true ubiquity is not a realistic goal for exascale programming models, the models and their realizations should run well across the breadth of potential systems. Although most aspects of this goal involve more administrative issues, such as standardization, it does entail some technical challenges. Specifically, the programming models must support portability. Further, merely being able to run on the breadth of systems is not sufficient. Changing systems should not require the sacrifice of any of the preceding design goals. In particular, exascale programming models must support performance portability, which has long proven an elusive goal.

R&D Opportunities

While the high-performance computing community clearly has developed a set of successful and established programming models and supporting implementations, success for the exascale era will require substantial efforts to explore the evolution of these existing techniques, as well as the development of significant new technologies. The ability to address the set of technical challenges discussed above will require a focused collaborative research and development effort among the national laboratories, industry, and the academic community.

Systems Software

SAND Number: 2011-1400P

NNSA working group participants

- Ron Minnich, Bob Ballance, Ron Brightwell, Sue Kelly (SNL)
- Chris Dunlap, Jim Garlick, Maya Gokhale, Pam Hamilton (LLNL)
- Mike Lang, Scott Pakin (LANL)

Scope

Systems software is comprised of: kernel, communication libraries, job scheduler, systems health and systems monitoring. Out of scope are: compilers and math libraries.

General

The pragmatic software environment cobbled together in the early 1990s for the first MPPs has not evolved significantly since. The brittle "perfect machine" model, OS bypass in lieu of an OS abstraction for message passing, and user-initiated checkpoint/restart are illustrative of these early approaches still in use on the largest MPP systems. These approaches have endured because the last 1000X increase in capability has largely arisen from increased node performance without substantially increased node counts. The windfall provided by Moore's Law has mitigated the need for dramatic improvements in the scalability of the software stack.

Unfortunately, the end of Dennard scaling means clock rates have plateaued, and future systems may need to reduce them in order to curb the energy demands of dramatically larger systems. Increased capability will now require massive increases in the number of CPU cores -- a trend that has already begun in petascale systems. Exascale systems as envisioned by DARPA[citation] and others will require distributed operating system software that is fault-tolerant and capable of managing power and data movement as schedulable resources in a heterogeneous environment. The operating system will need to provide applications with the abstraction of a perfect machine, while managing the increased complexity of adaptive management and automatic tuning to optimize resource utilization.

Plans for evolutionary change, or incremental modifications to existing software, should face much sharper scrutiny than plans based on revolutionary change. We plan to keep this in mind for SSW.

Assessment of current effort within ASC program (and/or community)

There are several Systems Software efforts of note funded within the NNSA labs:

- The Kitten LWK has been written at Sandia and runs on the Cray XT series machines. It is the latest in a series of LWKs.

Non-NNSA funding for systems software at the NNSA labs includes:

- The HARE project includes Sandia, IBM, and Bell Labs. It is exploring the use of a non-Linux, non-LWK, "lightweight general purpose" kernel and has been ported to the Blue Gene series of machines. HARE ends this year.
- The ZeptOS project at ANL explores the use of Linux on Blue Gene machines, and has shown very good results with a modified Linux kernel.
- The FOX project, which includes SNL (lead), PNL, LLNL, IBM, Bell Labs, BU, and OSU, is exploring new models of application runtime, fault tolerance, and operating systems concurrently. One of the operating systems is the new Osprey operating system, which replaces the Unix open/close/read/write model with a process IPC model.

Specific technical challenges to get to exascale

- Much of DOE HPC work to date has focused on the adaptation of consumer-grade software and hardware to HPC needs. It is not clear that this trend will easily continue to the exascale. Our exascale programs envisage parallelism for single programs on a scale thousands of times larger than other uses, including large-scale commercial applications such as Google.

- System software that manages millions of nodes is fundamentally different from that which manages thousands. We question whether replicating a complex operating system like Linux a million times will be feasible.
- We need a tighter integration to the programming model needs rather than providing a generic interface that may not be appropriate to application needs. Put another way, SSW should provide abstractions that are useful to the applications.
- Power management is another area of concern. The power subsystems information is not available to the parallel runtime. We need power management interfaces that are efficient, far faster than the multi-second power management interfaces available today on COTS PC systems, and capable of providing both global (whole machine) and local (per-node, per-board, per-rack) power information and control.
- Hardware features may be offline for power reasons, and a runtime should take this into account in the scheduler.
- A unified framework for event collection and propagation needs to be standardized upon to allow data from disparate components to be accumulated and synthesized by the RAS subsystems.
- The RAS subsystems function in an out-of-sight manner: information is gathered and forwarded to a central facility, and disseminated from there to programs that need it. This loop is too slow, and does not provide for fast reaction to local events. We feel that it will be essential to extend RAS to allow local decisions to be made and managed by the runtime.
- How do you keep the system running and the target computation(s) making forward progress in the presence of constant failures? Reliability and resilience issues will play a huge role at exascale. A change in resilience may affect everything: programming models, languages, runtimes, file systems, and operating systems. The RAS subsystems may allow for the proactive detection of select failures and the reallocation of resources.
- We will need to work out OS interfaces to allow runtime and application to expect and react to failures. The current mechanisms are extremely heavy-weight, usually involving the use of signal(). The OS must support lower-overhead paths.
- Dynamic management of resources figures heavily into the future. We need algorithms, applications, runtimes, and systems software that can respond rapidly to changing environments. One unknown question is just how rapid that has to be -- measured in milliseconds, seconds, or minutes?
- Current systems are perfectly uniform. In the future we will see heterogeneity everywhere, in both processors and network. It is probably time to plan for runtimes that can deal with "lumpy", i.e. non-uniform systems.
- Better access to hardware info is essential, but it must also be more efficient, faster access. Hardware notification of some events either does not happen at present or is vectored through the high-overhead firmware-based mechanisms.
- File system APIs have been driven by the Unix model. This model has led to the development of parallel file systems that, in turn, drive new operating systems to supporting the Unix API to support the file systems that ... It is time to get off this treadmill. We need to work out new, scalable file system APIs that are not tied to a 40-year-old API.
- Memory management has been stuck on the Unix model for decades: unique address spaces per process. This model in turn has led to the need for pinned virtual memory, and a host of other problems that plague our OS and runtime. It is time to look at richer models, including Single Address Space systems, which can allow for more efficient, lower-overhead network I/O.
- 3D Memory may be in our future, as may other even more complex memory systems. What is a workable model for this memory? Do we treat it as another level of cache, and consider DRAM as a set of cache lines? Is it application managed? Is it coherent? We need some strawman APIs, as well as simulation efforts and early hardware access, to figure this out.
- Threading is viewed as useful, but the Posix/Pthreads API is not seen to be a good fit for exascale. Something better is needed, possibly inspired by new efforts such as Google's concurrency model for the new Go language.
- Increased complexity and decentralization will necessitate “defense-in-depth” concepts and security features being built into many components of the system in a manner that does not impinge on performance.

R&D Opportunities for the near term (next 3 years)

- It might be worth porting a research kernel such as Kitten to BG/P to allow explorations of its capabilities; a partnership with IBM to explore extensions to its Compute Node Kernel with some HARE and/or Kitten capabilities could yield valuable results.
- Set up simulators to allow booting OSes and testing ideas.
- Use virtualization on systems such as Jaguar to support booting SSW on a large scale.
- Open up HPC resources for research -- INCITE has been useful in the past but few if any systems software projects made it into INCITE this year.
- We should continue to encourage development of mini applications and synthetic workloads amenable to testing in simulation.
- Need to provide interfaces for tools and programming models.
- Need to prototype and test APIs for data movement, resilience, power, and control systems.
- Need well defined requirements for visualization and analysis. Will these require multiple process support per node? Will we need to partition on a node basis?
- In any event, the more risky research started with FAST-OS, which ends this year, should built open to support exploration of new OS models that will have a better chance of scaling to future needs.

Solvers, Algorithms, and Libraries

Members: D. Knoll (LANL), J. Wohlbier (LANL), M. Heroux (SNL), R. Hoekstra (SNL), R. Hornung (LLNL), U. Yang (LLNL), P. Hovland (ANL), R. Mills (ORNL), S. Li (LBNL)

Scope: The Solvers, Algorithms, and Libraries working group (SAL) addresses technical challenges and near-term R&D topics necessary for successful execution of PEM and IC application codes on future architectures:

- SAL and application team efforts are complementary and collaborative.
- Solvers include the full range of linear and nonlinear solvers and preconditioners.
- Algorithms include multiphysics time integration schemes, scale-bridging, load balancing, mesh generation, embedded UQ, and parallelization strategies.
- Libraries provide high-quality reusable software components that can be leveraged across multiple applications.

Assessment of current effort within ASC program:

SNL, LLNL, and LANL have several existing efforts addressing future architectures. These efforts represent only a small fraction of the SAL activities that will be required in the near future.

- New data structures, APIs, and programming models to support a broad range of linear and nonlinear solver methods for evolving modern architectures (SNL).
- Advanced linear solver capabilities for modern architectures including exploration of hybrid threading under MPI, new smoothers and other algorithm components for multigrid, manycore (CPU and GPU) capable preconditioners and block recycling and communication-avoiding iterative methods (LLNL, SNL).
- Miniapps representing key IC and PEM needs to enable performance studies and parallel algorithm exploration for new architecture/programming models (LANL, SNL).
- Exploration of coupled physics algorithms in current codes, such as ALE hydro and multispecies thermal transport, and assessment of their suitability to perform well on future platforms (LLNL).

- New algorithms to improve the mapping of IC time integration approaches to modern architectures (potentially less sequential access of unit physics). This same algorithmic development direction can lead to more self-consistent scale-bridging to many single physics PEM simulation tools on modern architectures (LANL).
- Fault tolerance and remediation in basic linear solver operations (LLNL, SNL).

Specific technical challenges to get to exascale:

Common technical challenges across all SAL activities: (i) migration to scalable manycore (where *scalable* refers to the system design including network, distributed OS, file system, etc.), (ii) increased need for data locality and data placement, (iii) use of hybrid programming models, such as MPI+threading, instead of MPI-only approach, (iv) communication-reducing or avoiding approaches, (v) redistribution of data for better load balance, (vi) development of appropriate data structures for better data locality and easier access to parallelism, (vii) fault tolerance built into algorithms, and (viii) development of performance models to predict potential bottlenecks.

Specific issues related to Krylov solvers, preconditioners, and multigrid methods:
Most (if not all) of the successful applications on scalable manycore systems have a global SIMT (single instruction multiple thread) parallel execution pattern. This pattern is sufficient for many data parallel computations found in explicit methods and simple implicit problems. However, many NNSA applications have more complicated computational patterns and rely on solvers with sophisticated preconditioners. Right now we do not have algorithms that efficiently map to manycore nodes for these applications. Some specific issues for multigrid libraries include the effect of massive parallelism (millions or billions of cores) on multigrid convergence (e.g. some of the best smoothers are highly sequential, while parallel variants are often less effective), communication issues on coarser levels in algebraic multigrid (increasing stencil sizes require additional neighbor processors but less data per core on coarser levels), and the optimal use of shared memory nodes in the multilevel hierarchy, e.g., larger coarse grids resolved using a shared memory sparse direct solver.

Specific algorithmic challenges for current multiphysics IC code time integration algorithms:
Mapping unit physics packages / libraries onto an entire modern architecture and using standard sequential operator splitting will most likely result in inefficient use of exascale resources. We must re-think current, highly sequential, operator-splitting approaches for efficient implementation of IC codes in the Exascale Initiative. A related algorithmic challenge results from the goal to utilize modern architectures to increase IC and PEM code predictive capability. This goal can be achieved by using more first principles physics, and less phenomenological "modeling" on the system scale. We need to invest in the computational co-design of consistent scale-bridging algorithms.

Other technical challenges: We need to understand when computing data "on the fly" is more beneficial than using pre-computed data tables. We must motivate and consider the development and use of embedded UQ approaches. Validating existing miniapps against real codes and generating additional miniapps for use in development of new SAL methods will be challenging, and will require close interaction with the Apps and PM working groups. Additionally, C++ template meta-programming is currently the most effective way to write portable manycore software. However, this may not be sufficient for all computations in complex multiphysics codes. Regardless of the programming model, the software refactoring effort may be huge and the tools to do it are limited. This is a big risk factor, especially for Fortran based development. Emerging open standards for writing portable manycore software, such as OpenCL, must also be considered.

R&D Opportunities for the near term (defined as the next 3 years):

It should be clear that an increased R&D effort and subsequent progress in SAL activities is required for successful transition of IC and PEM activities to exascale resources. Given the broad possibilities in the Exascale Initiative architecture landscape, it is imperative to embark on an equally broad SAL R&D program.

Manycore preconditioners: We must develop non-SIMT preconditioners (esp. smoothers for multigrid) that are more robust than polynomial or Jacobi scaling. This is an important area where we must have success in the next few years. Specifically for multigrid libraries, we must investigate approaches that reduce communication such as combining more local smoothing steps, aggregating messages to reduce latency overhead, using idle processors to accelerate convergence, and non-Galerkin approaches for algebraic multigrid to control stencil growth. Multigrid in time is also of interest.

Physics-based approaches: Solvers and preconditioners tailored to particular sets of equations and discretizations: rather than focus on making general solvers work at exascale, it might be better to have a suite of specialized solvers. By exploiting knowledge of physics and discretizations solvers may scale better and be more robust. Discretization-specific solvers have been shown to be necessary for E&M and MHD problems.

Multiphysics time integration: Co-design new algorithmic approaches for multiphysics time integration in IC codes. Current operator splitting approaches are highly sequential and require mapping each unit physics library / package across the entire architectures. Research is required to develop algorithms with less sequential operator splitting, co-designed with application code developers for modern architectures.

Scale-bridging: Co-design two-way, consistent scale-bridging algorithms. This area has been highlighted more by PEM, but has the potential to be very synergistic with multiphysics time integration research. The goal of this work is the collaborative development with app code teams of algorithmic capabilities that allow system-scale simulation with fine-scale physics fidelity. This is a natural candidate for effective use of exascale resources and reduced reliance on approximate closure models.

Embedded UQ: There is a significant R&D opportunity in the development, acceptance, and implementation of embedded UQ methods.

Fault tolerance: Develop techniques to validate the results of numerical algorithms without the use of replication and to correct errors, including random and fail-stop errors. Methods must maintain high performance, using load balancing and related techniques.

Miniapps: All efforts discussed in the section will benefit from development and use of miniapps that represent key aspects of PEM and IC efforts. We (SNL) have existing miniapps for unstructured implicit finite element (MiniFE), explicit dynamics (PhdMesh), molecular dynamics (MiniMD) and circuit modeling (MiniXyce). We (LANL) also have the MAMA project (miniapps on modern architectures) which uses C++ and OpenCL to create an environment for flexible task and data parallelism. MAMA will enable rapid miniapp development for modern architectures. We expect to develop more in order to provide coverage of important large-scale application need

I/O, Networking, and Storage

Members: Hal Armstrong (LANL), John Bent (LANL), Alok Choudhary (Northwestern), Matt Crawford (Fermilab), Mark Gary (LLNL), Gary Grider (LANL), Luis Martinez (SNL), Chris Morrone (LLNL), Rob Ross (Argonne National Lab), Lee Ward (SNL, Lead)

Scope

We are responsible for the high performance IO and file system within the multi-program, parallel machine as well as the networking infrastructure used to move data and user sessions within the NNSA complex.

The IO portion of our domain can be any portion of the software stack, from the application libraries and supporting middleware to the support file systems found within the operating system. We also are responsible for working with hardware storage vendors in support of our mission. This can be a wide range of engagement. We meet frequently with vendors to discuss hardware changes that better support our mission and we have prototyped hardware solutions in order to gauge mission impact with experimental solutions.

The networking portion of our domain is typically restricted to hardware. We work with the vendors to suggest improvements in support of our mission, evaluate and test new vendor offerings, and provide support to our operations infrastructure in identifying and deploying state of the art enterprise and long-haul networking technologies.

Current State of the Art

By far, the greatest demand on our IO systems, deployed in high performance computing, is for the purpose of checkpoint and restart of long running jobs with compute needs that outlive current mean-time-to-interrupt times for the machines we deploy. Because of the highly synchronized manner in which our applications function the IO pattern for checkpoint-restart is bulk-synchronous, the distributed application dumps state from all it's component nodes at the same time and to the same globally shared storage.

Currently, to support this activity we primarily utilize the Lustre file system, an open-source and community supported project, the PanFS file system, from Panasas corporation. We have in the recent past used the GPFS file system, from IBM corporation, and may do so again in the future. All of these file systems function in a roughly similar fashion. They centrally coordinate, and manage, a large set of storage devices hosted on the high-performance machine, or near by.

To date, these solutions have worked well, particularly when the applications do not attempt to coordinate access to shared files. Some applications do share files, though, and cannot, typically, achieve better than ~50% of the potential IO bandwidth available. As we move toward the exascale computing realm our classic solutions will be hard-pressed to address our needs, unaided. The non-shared files approach that delivers our best performance, now, will be impacted by the sheer number of compute clients that are predicted for the exascale machine. The controllers, in some cases, do not scale well

and the control protocols, in others, look to be challenging at scale. For the shared-file case, the same problem is present but it is exacerbated by the need to manage a globally-coherent address space at an unprecedented scale.

Application middleware in our complex is, primarily, a small collection of scientific data file management and abstraction libraries. Typically, the Hierarchical Data Format (HDF version 5) library, or some variant of the Network Common Data Form (NetCDF) interface, or the hom-grown EXODUS format in one case. Adoption of these abstractions is scattered within the applications. Developers seem to welcome the useful abstractions but sometimes are unwilling to pay the performance cost of maintaining them. They can be troublesome to tune for a particular machine and file system. As we move to exascale, those that require globally coherent address spaces will be particularly challenging to keep attractive.

Our infrasturcture networking challenge is highly constrained by a need to remain compatible with commonly deployed technologies. This form of networking is available, and must interconnect with, our external enterprise networks which is always based on commodity protocols and hardware solutions for both economic and technological reasons. We could not afford to generate, maintain, or motivate external support for the translation equipment or the long haul routing and transmission hardware if we turned to custom solutions. We would be challenged to design and deliver competitive solutions for the enterprise and long-haul as such function has never been within our scope of responsibilities in the past. We must allow the market to dictate our options, though we can influence the high-end of that market.
As we progress toward exascale we intend to track the market and purchase the most cost-effective solutions for our needs. We cannot, now, often and reasonably move our datasets between the various locations in our complex easily and we do not foresee a better scenario as we approach exascale.

Research Needs

The current file system solutions were designed to work in a hierarchy that had, never great but better, ratios of bandwidth at their interfaces. The petascale realm has altered those for the worse, exponentially so. Everything indicates this will continue as we approach exascale. In the main, we must address this problem as well as the predicted shortcomings of the current file system's need for centralized management.

We believe this is best done by researching file systems that can smoothly tier and avoid, or mitigate the need to manage more than a few storage components at any time, irrespective of the number of clients that are simultaneously accessing the storage. Newly arrived, and attractively priced, solid state storage may be incorporated as a "burst" buffer for our bulk-synchronous checkpoints. We can quickly write a checkpoint into this storage and then, when the application has re-entered the long compute phase, drain the sold state store to higher capacity, cheaper, more classical rotating magnetic stores. We must address the centralized control problem as well. For this, modern peer-to-peer and "cloud" services would seem to offer an inspiring model. While not directly applicable to our environments they do contain interesting and, undeniably, scalable alternatives that we might leverage as part of our solutions.

Such a file system represents an architecturally integrated hierarchy of storage devices. As such, the opportunity to integrate what has been the traditionally separate data archive capability will occur. This seamless integration should be attempted as well since we already contemplate support for a high degree of heterogeneity and robustness in the inherently unreliable network and stores.

Our application developers have strong need for functional, fast, intuitive scientific data management libraries. More, strong interest in data provenance and an ability to make use of data sets simultaneously with creation and population motivates us to take a deeper look at our middleware stack. Perhaps something new, perhaps only significant modification and extension of the existing libraries. It is unclear, yet, which or how.

We estimate initial prototypes will be available by approximately 2015 with reasonable funding, and subsequent hardening and production quality implementations by 2020.

Development Needs

For file systems and storage at exascale, at least, our only answer cannot be simply a wonderful new file system that relies on the answers to questions nobody has even thought to ask until recently. We must not plan on an indeterminate future in that way. It would be irresponsible.

Therefore, we believe we must develop and deploy proven and likely technologies from our, and others, current research in order to ease the blow to existing file system solutions. Much of the non-scalable control problem with these file systems can be mitigated by making the real machine appear as something far smaller, far more like what existing file system were developed for. A project called the IO Forwarding Software Layer can accomplish this, and has been used in current production petascale environments with promising results. We believe that we can mitigate much of the single-shared-file issue with a layer of software that translates this mode of access into, what the underlying file system believes to be, many non-shared files. This solution is known as the Parallel Log-structured File System (PLFS) has been in use within our complex for a little more than a year and, again, with promising results. Finally, we believe relatively minor modifications to our Scalable Checkpoint and Restart (SCR) low-level library will allow us to address the bulk-synchronous nature of our checkpoints using solid state storage as a "burst" buffer, minimizing the storage budget for the anticipated exascale machine.

These technologies do not make a modest research program irrelevant, though. Some of us believe we can reach low exascale with judicious work on and deployment of these technologies. Some do not. All agree, though, that they buy us needed time and allow a significant, perhaps imperfect, risk mitigation strategy for the research we propose.

Our current archive solution, the High Performance Storage System (HPSS) will require continued maintenance and thought as we move to exascale. It's architecture accommodates a virtually unlimited number of parallel channels to and from the archive. The problem will be in making the current implementation better meet that virtual without requiring a major rewrite or refactoring.

The middle ware scientific data management libraries must be heavily worked on to address performance issues. Even today, many applications shy away from these libraries for performance reasons. Not surprising given that they were designed and implemented for the single workstation on the desktop. It is no stretch to imagine they will be completely unacceptable at exascale.

We estimate that the identified portions of our development will be useful by 2015, some already are in early stages.

Co-Design Needs

IO and infastucture networking are completely cross-cutting, we will need help gathering requirements and evaluating how well we meet application needs throughout. We will necessarily place demands on the compute and service operating system as well as the machine interconnect.

In particular though, we will need to work with the folks who specify and design the hardware and the operating systems on both the compute and service nodes in order to have our needs met, and the applications developers, tools and environment, and visualization folks to generate acceptable solutions for them.

Co-Design Opportunities

We can immediately identify opportunities to work with the DOE Office of Science ASCR program in pursuit of a common solution for both. As well, we should exmaine partnerships with the Hierarchical Data Format (HDF) folks to address our needs relative to performance of the HDF5 library. A previous partnership has generated the IO Forwarding Software Layer and, potentially, we might wish to extend it to include support for coming platforms in some fashion.

Potential Partnerships

Our scope is common within the HPC community and we believe we could meaningfully partner with DoD, academics, and DARPA in pursuing our goals. We have already begun a partnership with the DOE Office of Science ASCR program.

Visualization and Data Analysis

ASC Leads: James Ahrens[1], David Rogers[2], Becky Springmeyer[3],

ASC Participants: Eric Brugger[3], Patricia Crossno[2], Cyrus Harrison[3], Ming Jiang[3], Laura Monroe[1], Bob Tomlinson[1], Dino Pavlakos[2]

ASCR Liaisons: Hank Childs[4], Scott Klasky[5], Kwan-Liu Ma[6]

Los Alamos National Laboratory[1], Sandia National Laboratories[2], Lawrence Livermore National Laboratory[3], Lawrence Berkeley National Laboratory[4], Oak Ridge National Laboratory[5], University of California at Davis[6]

Scope

The scope of our working group is scientific visualization and data analysis. Scientific visualization refers to the process of transforming scientific simulation and experimental data into images to facilitate visual understanding. Data analysis refers to the process of transforming data into an information-rich form via mathematical or computational algorithms to promote better understanding. We share scope on data management with the Storage group. Data management refers to the process of tracking, organizing and enhancing the use of scientific data. The purpose of our work is to enable scientific discovery and understanding. Visualization and data analysis has a broad scope as an integral part of scientific simulations and experiments; it is also a distinct separate service for scientific discovery, presentation and documentation purposes. Our scope includes an exascale software and hardware infrastructure that effectively supports visualization and data analysis.

Assessment of current effort with the ASC program and community

By the late nineties, it was becoming increasingly difficult to efficiently and effectively visualize the largest datasets with existing tools. This was a significant concern to the scientific simulation community, because large-scale results were being generated and needed analysis. The Advanced Simulation and Computing (ASC) Computational Systems and Software Environment (CSSE) program changed this by supporting the development of multi-platform parallel visualization applications and toolkits. This software suite includes open-source visualization applications, ParaView and VisIt, an open source visualization library, the Visualization Toolkit (VTK), and a commercial package, CEI's EnSight. These solutions use parallel and distributed computing methods to offer visualization, imaging, and rendering algorithms. The result of these efforts significantly changed how large-scale, scientific visualization is carried out. Scientists today use parallel supercomputers and commodity visualization clusters to routinely visualize terascale and petascale sized datasets that could have never been effectively analyzed. The ASC visualization and data analysis solutions have become key elements in the computational science efforts supported by many government programs including Office of Science (OSC) Advanced Scientific Computing Research (ASCR) and Biological and Environmental Research (BER), the National Science Foundation (NSF) and the Department of Defense (DOD).

Data analysis seeks to characterize data using higher-level abstractions that can be used to find associations between data elements. Data analysis approaches recognize anomalies, find correlations, categorize data, make predictions, and assist in decision making. There are a broad range of techniques used in data analysis, including statistical methods, dimensionality reduction techniques such as principal component analysis (PCA), vector space modeling, machine learning, and clustering. The choice of analysis technique depends upon not only the type of data being analyzed, but also the intent of the user. Some methods permit exploration of data with a target of discovery, others assist in hypothesis testing, and some produce descriptive summaries. Unlike in visualization, there is little packaged scalable software infrastructure for large-scale parallel data analysis. Currently, the Titan Informatics Toolkit and VTK provide open source implementations of parallel statistics (bivariate correlative statistics, bivariate contingency statistics) and parallel canonical correlation analysis. Another open source project, R, provides some parallel support for statistical analysis, though only a subset of R's algorithms are fully parallelized. Commercially, Mathworks and SAS provide parallel support for a subset of their statistical analysis software packages.

Specific technical challenges to get to exascale

Technical challenges to get to exascale in the visualization and data analysis area are intertwined with challenges in other areas of exascale research, including storage, I/O, programming models, and hardware. In particular, visualization and data analysis at the exascale must be tightly coupled with applications development. While there will continue to be a need for the current post-processing capabilities, analysis at the exascale will require co-design with scientists who develop applications in order to enable a completely new class of exascale analysis.

A new in-situ exascale visualization and data analysis approach is needed since the petascale approach of storing a full-range of results for later analysis becomes impossible due to exascale storage technology trends - The rate of performance improvement of rotating storage is not keeping pace with compute. Provisioning additional disks is a possible mitigation strategy, however, power, cost and reliability issues will become a significant issue. This trend suggests that our ability to generate data on future supercomputing architectures will significantly exceed our ability to store this data. In addition, it is becoming clear that data movement from the chip out through memory hierarchies to storage has significant power costs. At the exascale, visualization and analysis will need to occur as the simulation is run; the option of writing full-scale results to storage will not work. The creation of an effective in-situ visualization and analysis infrastructure is an

important technical challenge. A related challenge is the design and addition of data-intensive hardware to the exascale supercomputer, such as memory buffers and analysis-enabled storage systems, to support visualization and data analysis.

Exascale simulation results must be distilled with quantifiable data reduction techniques – In order to achieve scientific understanding from massive simulations, visualization and data analysis techniques reduce the amount of data to a smaller understandable representation. Visualization techniques transform data proportional in size to the scale of the platform, (10^{18}) for exascale, to a visual representation shown on a display that typically has (10^{6}) pixels. Today, parallel graphics algorithms are primarily used to achieve this massive data reduction. This approach provides the foundation for our current successes in handling massive data, however, it is a brute force approach that requires significant computing resources to reduce the data and it is difficult to quantify the bias of this approach. Approaches that quantifiably reduce data as it is generated need to be explored.
New exascale-enabled statistical, parametric, multi-physics, multi-scale simulation approaches require corresponding new visualization and data analysis approaches – Simulation scientists are using new approaches to model more complex phenomenon on exascale resources. Parametric studies record how a simulation responds in a parameter space of possibilities. Multi-physics approaches simulate a linked model of different related phenomena such as a linked physics and chemistry simulation. Multi-scale approaches simulate phenomena at different spatial and temporal scales. Understanding and presenting both summarized and highlighted results from multiple sources such as parameter studies, multi-physics or multi-scale simulations are an important technical challenge that needs to be addressed.

Just like exascale simulations, visualization and data analysis approaches will need to run efficiently on exascale platform architectures and take advantage of a very high degree of parallelism – Technical challenges include achieving portability, efficiency and integration flexibility with simulation codes.

Research and Development Opportunities for the Near Term

For the near term, it is important to prototype, test and deploy a variety of approaches to address the identified technical challenges and evaluate the advantages and disadvantages of these solutions. A list of opportunities is presented below. An evaluation metric and linkages to other areas are identified for each of these opportunities to track progress towards exascale.

1. **In-situ visualization and data analysis software infrastructure** - As we progress to exascale we must move to a model of visualization and data analysis that occurs as part of the simulation run to avoid costly data movement. Visualization and analysis results will be generated as part of the simulation and decisions about what data can be saved will be made dynamically as the simulation progresses. This project addresses the infrastructure required for in-situ analysis. Linkages to the application teams are critical since this infrastructure needs to be collaboratively designed. Success metrics include improving time-to-insight, reducing storage costs and improving result quality.

2. **Advanced data reduction techniques including statistical sampling, compression, multi-resolution and science-based feature extraction approaches** - The massive data that is generated while the simulation is executing needs to be prioritized and reduced immediately. Statistical sampling techniques, information theory and compression approaches are techniques that reduce data size and identify elements of interest. Collaborative design of these approaches with applied mathematics groups and applications groups is critical. Success metrics will be tied to appropriate application milestones.
3. **Visualization and data analysis techniques to help understand advanced exascale physics approaches** – Processing collections of simulation results and providing summarized insight from an ensemble of answers is an important opportunity to pursue. Identifying how results from different aspects of a simulation suite relate to each other, that is, how data relates across scales and how data relates across different joint-physics simulations, is a key aspect of this opportunity. Applied mathematics and applications are vital partners. Success metrics will be tied to appropriate application milestones.
4. **Implement core visualization and data-analysis capability using a scalable parallel infrastructure** – Our visualization and data analysis solutions need to work on the exascale supercomputers. We will partner with the programming models, tools and applications groups to produce a scalable code base. Our success will be measured by our readiness for applications as machine delivery milestones are met.

5. **Exascale visualization and data analysis hardware infrastructure** – We have the opportunity to create an efficient data-intensive hardware infrastructure for the exascale platform. Examples include exploring the use of memory buffers for staged analysis and storage, as well as analysis-enabled storage systems. Success metrics include improving time-to-insight, reducing storage costs, and improving result quality.

6. **Knowledge infrastructure** – Tracking and using knowledge about the scientific goals makes visualization and data analysis more effective. Our opportunity is to integrate the knowledge from workflow tracking and data management systems to improve the quality of the visualization and data analysis solutions we create.

NNSA ASC Exascale Environment Planning
Applications Working Group

C. H. Still (LLNL), lead; A. Arsenlis (LLNL); R. B. Bond (SNL);
M. J. Steinkamp (LANL); S. Swaminarayan (LANL); D. E. Womble (SNL);
A. E. Koniges (LBNL); J. R. Harrison (ORNL); J. H. Chen (SNL)

February 2011

Working Group Scope

The scope of the Apps WG covers three areas of interest: Physics and Engineering Models (PEM), multi-physics Integrated Codes (IC), and Verification and Validation (V&V). Each places different demands on the exascale environment. The exascale challenge will be to provide environments that optimize all three.

PEM serve as a test bed for both model development and "best practices" for IC code development, as well as their use as standalone codes to improve scientific understanding. Rapidly achieving reasonable performance for a small team is the key to maintaining PEM innovation. Thus, the environment must provide the ability to develop portable code at a higher level of abstraction, which can then be tuned, as needed. PEM concentrate their computational footprint in one or a few kernels that must perform efficiently. Their comparative simplicity permits extreme optimization, so the environment must provide the ability to exercise significant control over the lower software and hardware levels.

IC serve as the underlying software tools employed for most ASC problems of interest. Often coupling dozens of physics models into very large, very complex applications, ICs are usually the product of hundreds of staff-years of development, with lifetimes measured in decades. Thus, emphasis is placed on portability, maintainability and overall performance, with optimization done on the whole rather than on individual parts. The exascale environment must provide a high-level standardized programming model with effective tools and mechanisms for fault detection and remediation.

Finally, V&V addresses the infrastructure and methods to facilitate the assessment of code and model suitability for applications, and uncertainty quantification (UQ) methods for assessment and quantification of margins of uncertainty (QMU). V&V employs both PEM and IC, with somewhat differing goals, i.e., parameter studies and error assessments to determine both the quality of the calculation and to estimate expected deviations of simulations from experiments. The exascale environment must provide a performance envelope suitable both for capacity calculations (high through-put) and full system capability runs (high performance). Analysis of the results place shared demand on both the I/O as well as the visualization subsystems.

The sections below provide current assessments of exascale efforts within ASC, list technical challenges, and suggest near-term targets for research and development.

Assessment of Current ASC Effort

NNSA Laboratories have begun exploratory studies to assess which ASC tools might be viable, and exploring what issues will be most critical in the development of new methods. These explorations have included work performed on surrogate hardware such as ASC Roadrunner

and ASC Dawn, and Linux based clusters hosting multiple Graphical Processing Units (GPU's) per node. Using these systems, code developers can address exascale-relevant issues at both the node and full system levels. Each Lab has companion research efforts associated with the other working groups, and each Lab has research teams involved in ASCR co-design centers, thus connecting to the larger exascale research community.

Specific Technical Challenges

In addition to exascale systems, applications must also support other architectures. Portability and high-level abstractions in the programming model will be critical, both for IC because of the complexity of the application, and for PEM because of the smaller development team. Mechanisms for expressing data hierarchies and optimization thus far have been closer to machine level programming than high-level abstractions. As architectural complexities increase, more assistance is needed from compilers and tools. This will be particularly true at the exascale.

The most significant restrictions on exascale design are due to power constraints. An exascale system will have thousand-fold performance increase at approximately a ten-fold power increase. The dominant power consumer is data motion, so data locality and memory bandwidth will be critically important. Another main concern will be resilience/reliability. Mean time to failure scales inversely to the number of components in a system, so for an exascale class computer, fault tolerance and mitigation are critical, as well as resilience and ensuring correctness.

Current high-performance computers generally have a few processing elements (cores) per node, with relatively high memory per core, and just as importantly, relatively high memory bandwidth per core. This allows work to be streamed into the core efficiently. The core operates efficiently until the pipeline stalls waiting for slow memory hardware. Minimizing stalls becomes the critical path for optimization. Slow memory is scheduled for less redundant access, floating-point operations are managed. Integer performance, necessary for efficient indexing into data structures, is also quite good. The individual nodes are coupled by an interconnect which has high enough bandwidth to deliver significant amounts of data from off-node via explicit message passing without imposing too heavy a burden on the algorithms. The accompanying global parallel filesystem is relatively inefficient, emphasizing reliability and robustness -- balancing read/write operations for multiple jobs over a large range of data request sizes.

Hardware and software reliability is generally very good. Error-correcting memory is available, and the software stack is able to catch and report or correct most faults. The number of uncorrectable (fatal) faults is generally low enough that task pre-emption (i.e., application job crash) is an acceptable solution. Application software mitigation techniques are fairly straightforward, and further insurance is provided via full-application checkpointing to the parallel filesystem. When a fatal fault occurs, the calculation is restarted from the most recent checkpoint. These restart sets are often also used for code result analysis.

Future exascale systems will be quite heterogeneous. The number of nodes in an exascale system will increase, but the more dramatic change will be in the number of cores within a node, and the increased number of levels of data hierarchy. Both total memory and memory bandwidth per node may be somewhat greater, but the number of cores per node will increase hundred-fold. Hence, memory per core, and memory bandwidth per core, will be substantially smaller. Thus, algorithms must change to manage data motion and locality. Data motion induces delays. Like today, best efficiency will occur by computing on available local data as long as possible. Unlike today, the penalty for fetching non-local data will be significantly greater.

Some level of mitigation is possible if fast context-switching hardware threading is implemented. A thread in ready-state can begin processing while the prior thread waits on a memory fetch. However, managing and scheduling hundreds of threads present a much greater challenge than codes face today. Pure explicit message passing will be replaced by a shared-memory hybrid incorporating threads. Future hardware node boards may include I/O and visualization nodes integrated into them, providing opportunities for inline data analysis, and embedded algorithms for UQ.

Fault tolerance will be a much bigger issue at exascale, and the software stack must either catch and handle faults, or provide information to the application allowing fault handling. Advanced features such as task migration seem desirable, but allowing the application to detect that a migration has happened will be needed for benchmarking and performance analysis. If checkpointing is no longer feasible because of I/O limitations, another mitigation technique will be needed for recovery from fatal crashes.

Research and Development Near Term Opportunities

Primary areas of concern for the applications (APPS) are data structure and memory locality, availability and quality of tools, emergence of a suitable programming model, and issues of resilience and correctness. Additional areas for concern are inline data analysis, non-determinism not associated with resilience, and ensuring the hardware can support application needs. Each of these provides opportunities for cooperative R&D.

As mentioned previously, memory bandwidth will be constrained, and managing data locality will be particularly critical. Programming at a high level will be important in order to obtain portability and high-level concurrency, but mechanisms are needed to selectively expose the underlying memory complexity, and ensure data access paths are minimized. Programming models (PM) must specify that level of control without sacrificing the higher-level abstractions. In addition, the ability to share common data structures or threading models with a solver or library (SAL) could lead to higher performance. To develop either the PM or SAL, APPS will need tools to evaluate the effectiveness of data placement and suggest avenues for optimization and tuning – and metrics for exascale performance, which are not yet clear.

To address reliability, resilience, and mitigation techniques, APPS needs to work with hardware architects (HW), software stack developers (SSW) and I/O developers (IONS). The ability to know what is correct, and to reproduce answers for debugging has been a cornerstone of code development for some time, and retaining that capability will be essential. UQ is generally predicated on having some level of determinism. APPS will also need to work with HW on mechanisms for indirect addressing and ensuring adequate memory per core for 3D neighborhoods. These are critical capabilities for a number of the APPS computational models.

Finally, another avenue for cooperative R&D is in the development of mini-apps to be used as test and development platforms (surrogate applications), but they must meet somewhat conflicting criteria: coarse-grained enough to examine multimode issues, fine-grained enough to examine on-node issues, and large enough data sets with sufficiently complex physics module interactions as to ensure relevance to ICs. If such mini-apps can be developed, other working groups can use them to evaluate research designs in simulator environments.

The Applications Working Group would like to gratefully acknowledge contributions from Andrew F. Nelson (LANL).

Tools and Tool Support for the Exascale Era
For the NNSA Workshop on Exascale Computing Technologies
LLNL-TR-472494

The ASC Working Group on Exascale Tools

ASC Working Group Members and ASCR Participants

Martin Schulz, LLNL (WG lead)

Atinuke Arowojolu, DOE
Sean Blanchard, LANL
James Brandt, SNLs
Scott Futral, LLNL
John Mellor-Crummey, Rice University
Barton Miller, University of Wisconsin
David Montoya, LANL
Mahesh Rajan, SNLs
Kenneth Roche, PNNL
Mary Zosel, LLNL

1 Goals and Scope of this White Paper

The goal of this paper is to highlight the challenges in providing scalable tool support on exascale class machines and to identify key research and development needs as well as opportunities to meet these challenges. In this context we define tool support very broadly as software that helps programmers to understand, optimize and fix their codes as well as software that facilitates interaction between application, run-time, and hardware. This includes tools for performance analysis, static and run-time optimization, debugging, correctness verification and program transformation.

This paper is intended as background material for the ASC exascale workshop, held in March 2011 in San Francisco, to stimulate discussion both within the tools area and across working groups. Further, this paper targets both the issues and requirements users have on tools in the exascale era as well as the requirements that need to be fulfilled by exascale systems to build scalable tools.

2 Background and State of the Art

Exascale class machines will exhibit a new level of complexity: they will feature an unprecedented number of cores and threads, will most likely be heterogeneous and deeply hierarchical, and offer a range of new hardware techniques (such as speculative threading, transactional memory, programmable prefetching, and programmable accelerators), which all have to be utilized for an application to realize the full potential of the machine. Additionally, users will be faced with less memory per core, fixed total power budgets, and sharply reduced MTBFs. At the same time, it is expected that the complexity of applications will rise sharply for exascale systems, both to implement new science possible at exascale and to exploit the new hardware features necessary to achieve exascale performance.

LLNL-TR-474731

While several tool sets have been successfully deployed on Petascale machines, in most cases this support is rather limited. Scaling is often achieved by applying brute force and tools are restricted to single programming paradigms. Furthermore, current generation tools mostly focus on the data collection combined with post mortem analysis and visualization and have only limited support for online or in situ analysis and evocation of response.

To overcome these limitations and provide the users with the necessary tool support to reach exascale performance, we need a new generation of tools that help users address the bottlenecks of exascale machines, that work seamlessly with the (set of) programming models on the target machines, that scale with the machine, that provide the necessary automatic analysis capabilities, and that are flexible and modular enough to overcome the complexities and changing demands of the exascale architectures.

3 Exascale Challenges and R&D Opportunities

To address the challenges posed by exascale systems and to meet the high-level requirements outlined above, significant research and development in the area of tools and tool support is needed. These efforts will need to focus both (1) on developing new tools capabilities that users will need to scale their applications to exascale and (2) on novel infrastructure that make implementing such tools feasible. Further (3), research and development of software tools has a significant overlap with all other areas of exascale software and hardware design; these must be addressed as part of Co-Design efforts. Finally (4), after tools have been developed we need concrete support models that ensure the availability of the tools in the long run and across multiple hardware generations. In the following, sections we will highlight issues and research opportunities for these four areas.

User Expectations on the Usability of Exascale Tools

Integrated simulation codes are large, often complex, and sometimes pushing limits of everything including language features. This is particularly true for many of the NNSA codes. The tools deployed must be robust enough to handle these codes. This is in contrast to the compact applications or benchmark codes frequently used to test tools developed in research settings. This provides a challenge to exascale tool design, development, and deployment since one or two full-featured, highly responsive tools will not exist. Instead there will be a suite of more special purpose tools. An important part of tool deployment will include establishment of a usage model for the tools at exascale to establish expectations, limitations, and a guide to their applicability for problem solving.

3.1 User Requirements for Tools

Exascale machines are expected to look significantly different from previous machine generations. They will feature significantly larger core counts, less memory per core, new hardware features like programmable prefetching or speculative threading, software controlled accelerators, etc. Users will expect tools to help them cope with these new features and the challenges they introduce. Additionally, while HPC tools have traditionally focused on debugging and computational speed, exascale tools must also support the measurement and analysis of other metrics of interest such as memory utilization, temperature, reliability, and power consumption.

New Debugging Techniques

The increased complexity and core counts of exascale systems, will diminish the effectiveness of traditional interactive debuggers. To cope with the complexity of exascale executions, application developers will need additional tools that can help users to either automatically or semi-automatically reduce the problem to smaller core counts or to detect the problem itself. Tool support for debugging at exascale can range from simple approaches that cluster processes into similar groups to automatic root cause analysis tools that directly point users to the most probable causes for observed behaviors.

In addition to traditional reactive debugging approaches, users will also need proactive or defensive approaches that can check codes (statically or dynamically) before the actual debugging runs and that can identify and mitigate potential errors before they become fatal. Such verification tools have been successfully constructed for analyzing

MPI programs at smaller scales; however, to be broadly useful for exascale applications, such tools will need to be extended to large core counts as well as broadened to other programming models and libraries.

Automatic Correlations and Data Analysis in Performance Tools

One of the core challenges for performance tools at exascale will be the scalable collection and analysis of performance data. With millions of cores and billions of threads, tools will have to manage a flood of data and as a consequence, comprehensive execution tracing to a central storage location for post mortem analysis will be infeasible. To measure long-running executions in their entirety, only tools that record compact execution profiles will be practical.

Rather than simply presenting performance measurement data to application developers for them to explore, the complexity and scale of exascale executions will require tools to do more to direct attention to problems and phenomena of interest. Techniques for this must include adaptively and selectively recording of performance data, in situ or online analysis as well as data compression. Further, application developers will need new approaches to visualize the gathered data in way that is intuitive to them, e.g., by mapping performance data to data structures, or by translating it to the physical domain known to the application.

Tools for Memory Efficiency and Optimization

With rising core counts, the amount of available memory per core will be drastically reduced forcing application programmers to rethink their memory usage and forcing them to use memory more efficiently. This will drive the need for novel memory tools that provide insight into how well memory is being used, where memory is either wasted or unnecessarily replicated, and where memory allocations are not scalable (e.g., by requiring array sizes linear in the number of processors). In addition, we envision runtime systems that help applications keep track of memory usage dynamically or that dynamically optimize memory behavior, e.g., by reducing redundancies or by eliminating unnecessary transfers and temporary buffering.

Tools for Threading

At exascale, the use of threading will no longer be optional for applications. Without threads it will not be possible to reach the concurrency levels needed for exascale while staying within the confines of the limited node memory. Further, new architecture paradigms, like GPGPUs, explicitly rely on threading. However, tool support for threading is currently still weak and we need new approaches to provide the user with support for threaded programming models. This includes, but is not limited to, loop overhead measurements, detection of synchronization bottlenecks, and the startup time of threaded regions.

Tools for Power Optimization

Power will be one of the most constraining factors for building and running exascale machines. Compared to current machines, we need to increase the power efficiency by several orders of magnitude. While hardware advances are expected to contribute a substantial amount of these savings, it is also expected that system software and potentially applications, through a set of high-level annotations, will have to be power-aware and actively monitor, control, and reduce their power consumption.

To make the latter a reality, users and system software/runtime designers will require feedback on the power consumption of their codes. We need tools that gather information about power consumption and correlate the results to the application source code. These tools should be able to use both system wide monitors available at the board or rack level, as well as processor or chip set internal sensors.

Tools for Transformation to Accelerators

Exascale architectures will most likely feature some form of hardware acceleration, either in the form of highly threaded execution units as on GPUs or specialized vector units. In both cases a manual transformation of code to utilize such hardware accelerators is complex, tedious, and error prone. Users will therefore need automatic or semiautomatic tools that help with this process by identifying code regions suitable for acceleration, outlining them

into separate code pieces and transforming them into specialized code for the accelerator hardware. This process can be handled at compile time, at run time, or both. This model may be supported by specialized source code annotations embedded in the parallel programming model, such as OpenMP pragmas.

3.2 Requirements on Tools Infrastructures

In addition to new capabilities that tools need to offer to the user community, the exascale target systems also pose significant challenges to building and maintaining the tools.

Scalability

Tools themselves must be scalable and must be able to execute efficiently across the entire machine. Each tool component must be scalable by itself and cannot contain algorithms or data structures that scale linearly (or worse) with the number of processors. Further, tools must use the parallelism available in the system for their own processing. As a consequence, tools and their analysis algorithms will have to be distributed systems themselves.

Asynchronous Analysis Capabilities

In many cases tools will have to rely on online and in situ analysis techniques in order to preprocess any data that is being collected. Processing data for analysis within the application process itself can either be difficult due to limited OS support on some architectures or can lead to significant perturbations of the application's execution. It could therefore be advantageous to offload the processing of performance data to resources outside the compute nodes. This leads to asynchronous event processing with minimal perturbation.

One approach that is currently taken in many tools is the use of tree-based overlay networks. In this case a set of (possibly separate) nodes is used to create a second processing and communication structure with a tree-shaped communication topology. Such a structure lends itself well to hierarchical aggregation options executed within the hierarchy of the tree based overlay network. However, other communication mechanisms are also possible and which structure is most suitable will depend on the analysis algorithms. Given that communication of tool related information may compete with application traffic, it may also be advantageous to investigate the possibility of an out of band fabric or channel for communicating such information.

Analysis Response

In order to enable run-time application optimization in the face of changing application needs and platform state, the tools' framework must include the ability to feed analysis results back to the application and/or system software. This will require not only the mechanisms for invoking response, but also the logic of the decision-making process for response. Due to the complex nature of the platform and the interdependencies of its resources, response logic is not straightforward. However, concurrent monitoring and analysis can provide the information to be used in evaluating response logic and efficacy.

Fault Tolerance

At scale, tools will face similar reliability and fault tolerance challenges as the applications and hence have to protect themselves against it in a similar way. In many cases, e.g., for debugging tools that are intended to be used in application failure scenarios, the tools need to be even more robust and able to withstand system failures, since they are required to deliver useful information even after the application has failed.

Additionally, tools must be able to coordinate with the application fault tolerance mechanisms. In case of application faults during runs under tool control, it must be possible to restart both the application and the corresponding tool without a loss of state. Since this will most likely not work transparently—due to the application driven nature of most fault tolerance approaches—we must provide the necessary coordination APIs that tools can use to save their own state and to provide the necessary restore capabilities.

The Need for Componentization

The discussion above shows that we will need sophisticated tools to address the complexities of the target applications and systems. Each tool will itself be a highly distributed system and require substantial effort to implement and tune. On the other hand, no single tool will be able solve all problems - instead we will need the ability to create and maintain custom tools for particular problems or target platforms.

The use of generic and separable components will be key to overcoming these challenges: each functionally separable part of a tool should be implemented as its own component, which then is made available as part of a component library. Tools can assemble these components into a full end-to-end solution with minimal glue code. In the ideal case, tools may even be assembled directly from components alone using a tool construction specification (e.g., implemented as an XML file).

Overall, component architectures for tools will not only avoid stovepipe solutions and enable interoperability between tools, but it will also enable quick tool prototyping and the creation of custom or even application specific tools. This will allow tool providers to quickly react to new, unpredictable problems and provide users with quick and direct support without having to create specialized tools from scratch.

3.3 Crosscutting issues

Many of the above challenges cannot be solved by the tools community alone; rather development of effective tools will require a close collaboration and interaction with the entire system stack. In fact, to fulfill many of the requirements posed by the user community, tools need access to more in depth information across all system layers than is currently available.

It will be important to clearly define these cross cutting issues and API requirements as well as responsibilities for their implementation. Further, these APIs need to be system independent to allow for portability of tools across architectures and even across the two different swimlanes of exascale architectures.

Interfaces with Programming Models

It is essential that tools allow the users to relate any information that is being gathered back to the application, its source code, and its data structures. This will only be possible if tools gain access to abstractions provided by the respective programming models. Current approaches only provide such information in a very limited way making it hard for tools to interpret the performance data. We therefore need new interfaces that allow both the compilers and runtime libraries for programming models to deliver this information to tools.

Further, defining appropriate APIs would enable tool analysis results to be used by applications for run-time optimization. This targets not only current applications with inherent rebalancing and reconfiguration capabilities, but also encourages co-development of programming models and algorithms that can leverage such information. This additionally requires understanding and definition of dynamic application resource requirements and the impact of the resource state on the application.

Interfaces with Hardware/Architectures

Current architectures only provide limited insight into system characteristics, typically through performance counters. To understand the performance implications we will require new counters (e.g., for network traffic, for energy consumption, or for accelerators) and other introspection capabilities (such as memory reference tracing or external environment control). Further, these capabilities must be accessible to tools through standardized cross platform interfaces and must support the full breadth of tools. In particular, it must be possible to use performance counters safely in both caliper and sample based tools.

Interfaces with OS and System Software

Tools require access to a wide range of OS and system software information. This includes access to debug interfaces, symbol tables, thread allocation and scheduling data as well as resource utilization. To provide tools with access to

this kind of information, we need clearly defined and portable interfaces, even on machines with specialized compute node kernels.

Further, many of the features discussed above, in particular the offload capabilities for asynchronous processing of performance data, require additional resources. Tools need interfaces to allocate and control these resources that are provided by the system software in a scalable manner. For example, tools need to be able to locate process and thread information, launch tool daemons on suitable nodes and request additional processing nodes for online analysis and data aggregation.

Finally, where tool analysis results will be used for run-time optimization, tools must be able to feed results and response options to the OS and system software. For instance, if an analysis leads to a discovery of application imbalance, such information must be able to used to invoke a coordinated response by system resource managers, applications, and system response capabilities such as process migration.

Commonalities with Data Analysis/Visualization and I/O

Tools have infrastructure requirements similar to the data analysis, visualization, and I/O software stacks. Like tools for performance monitoring or correctness checking, data analysis tools will increasingly need to rely on in-situ or concurrent processing on an external set of nodes to avoid streaming unwieldy amounts of data to storage. Common infrastructure for tools, data analysis, visualization, and I/O should be explored.

3.4 Long term support models

In addition to the technical issues discussed so far, there must be a plan for long-term maintenance and support of valuable tools. As tools get developed and deployed we must ensure their long term existence and support, including continued testing, support for debugging and extensions, interoperability checks, and ports to new platforms and architectures.

While these basic problems already exist in the current software landscape, they will become even more important as we progress towards exascale. The software needed to provide the required support is itself a major investment and we cannot afford to reengineer it over and over again.

The solution to these challenges has both a technical and a managerial component: on the technical side the use of a component framework allows each component to be maintained and tested separately, which reduces maintenance complexity. Further, the ability to compose components that have been implemented by different authors distributes responsibilities and allows a broad engagement from the overall tools community.

On the managerial side, we need to ensure that funding is available for such long term maintenance beyond the core research. This could be done in the form of explicit maintenance contracts or through efforts like ASC's CCE or OASCR's ESC.

4 Ending Thoughts

To make exascale computing tractable, users will need sophisticated tools to maneuver the increasingly complex architecture and application space. This will include more scalable approaches for debugging and performance analysis, but will also reach into new areas such as memory efficiency analysis and optimization and power reduction. To provide these capabilities, tools themselves must face a series of challenges, be highly scalable, and be fault tolerant.

Delivering the sophisticated tools that will be required for exascale platforms will require a united effort by the tools community. The community can no longer afford to create vertically integrated stovepipe implementations; instead the community will need to establish components that can be shared across multiple tools, support the rapid development of new tools, and enable tool capabilities to be dynamically tailored in response to the system and application state and the problems at hand.

Exascale Hardware Architectures Working Group

For the NNSA Workshop: From Petascale to Exascale: R&D Challenges for HPC Simulation Environments

Group Lead: Scott Hemmert
Participants and Contributors: Jim Ang, Brian Carnes, Patrick Chiang, Doug Doerfler, Sudip Dosanjh, Parks Fields, Ken Koch, Jim Laros, Matt Leininger, John Noe, Terri Quinn, Josep Torrellas, Jeff Vetter, Cheryl Wampler, Andy White

Introduction and Scope of Working Group

The ASC Exascale Hardware Architecture working group is challenged to provide input on the following areas impacting the future use and usability of potential exascale computer systems: processor, memory, and interconnect architectures, as well as the power and resilience of these systems. Going forward, there are many challenging issues that will need to be addressed. First, power constraints in processor technologies will lead to steady increases in parallelism within a socket. Additionally, all cores may not be fully independent nor fully general purpose. Second, there is a clear trend toward less balanced machines, in terms of compute capability compared to memory and interconnect performance. In order to mitigate the memory issues, memory technologies will introduce 3D stacking, eventually moving on-socket and likely on-die, providing greatly increased bandwidth but unfortunately also likely providing smaller memory capacity per core. Off-socket memory, possibly in the form of non-volatile memory, will create a complex memory hierarchy. Third, communication energy will dominate the energy required to compute, such that interconnect power and bandwidth will have a significant impact. All of the above changes are driven by the need for greatly increased energy efficiency, as current technology will prove unsuitable for exascale, due to unsustainable power requirements of such a system.

These changes will have the most significant impact on programming models and algorithms, but they will be felt across all layers of the machine. There is clear need to engage all ASC working groups in planning for how to deal with technological changes of this magnitude. The primary function of the Hardware Architecture Working Group is to facilitate codesign with hardware vendors to ensure future exascale platforms are capable of efficiently supporting the ASC applications, which in turn need to meet the mission needs of the NNSA Stockpile Stewardship Program. This issue is relatively immediate, as there is only a small window of opportunity to influence hardware design for 2018 machines. Given the short timeline a firm co-design methodology with vendors is of prime importance.

Technology Challenges

Improving energy and power efficiency is the most formidable challenge facing exascale architectures. Exascale machines must be approximately three orders of magnitude more energy efficient than current machines, with a target of delivering an exa-op at 20 MW. This number implies the delivery of 50 giga operations per watt. In other words, the average energy consumption per operation must reach 20 pico-Joules (pJ). In comparison, Intel's Core Duo mobile processor (circa 2006), consumed more than 10,000 pJ per instruction average. The power issue also contributes to many other challenge areas. Power concerns have limited the increase in clock frequency and led to the multicore era of microprocessors, which is increasing the amount of parallelism in supercomputers at a faster rate than what has historically been seen. Issues with power and energy have also driven, at least in part, a shift to systems that cannot adequately balance

compute capability with memory and interconnect performance. In addition to these four areas, increasing part counts is causing growing concerns in the area of resilience. Each of these areas is discussed in more detail below.

Power Large machines spend most of the energy transferring data from or to remote caches, memories, and disks. Minimizing data transport energy, rather than only looking at arithmetic logic unit (ALU) energy, is the critical challenge. Several evolutionary approaches to attaining more energy-efficient architectures exist. At the circuit level, these approaches involve designing circuits for energy and power efficiency, rather than for speed, as in most current approaches. Such designs include on-chip interconnection network circuits for low-swing, and new memory layouts and bank organizations that minimize the amount of capacitance switched per access. Future memory designs must minimize the energy spent charging and discharging lines, possibly through memory designs that include hierarchical bit-line organizations. At the microarchitecture level, evolutionary approaches involve simplifying the cores, making their pipelines shallower and their execution engines less speculative. Finally, at the machine architecture level, a popular approach is to augment the processing nodes with accelerators that are energy-efficient for some operations. Unfortunately, attaining three orders of magnitude higher efficiency in energy and power requires all of this and much more. In particular, it calls for the investment on several research areas that we list next.

Possible near terms areas of investment include: non-silicon memory (see memory section below) and silicon photonics (see interconnect section). Additionally, one of the most effective approaches for energy-efficient operation is to reduce the supply voltage. For example, one possible research area is to set Vdd to a value only slightly higher than the transistor threshold voltage (Vth). This is called Near-Threshold Voltage (NTV) operation, and can lead to a 10x increase in energy efficiency, at the cost of throughput. However, it can cause less reliable operation and research into how to mitigate this would also be needed.

Increasing Parallelism The path to exascale involves increasing the level of parallelism within a node in addition to increasing the number of nodes in a system. The level of each is currently a research question, but to some degree both will be major challenges. At the node level, parallelism will become increasingly hierarchical in the form of a significant increase in the number of computational domains, processing elements per domain, and hardware contexts per processing element. In addition, the width of the computation/vector units will continue to increase. This architectural complexity will be a challenge for the areas of algorithms, programming models, operating systems and compilers. At the system level, the number of nodes in a system will also increase substantially beyond our current platforms. Although this will be a challenge for many areas, resiliency and reliability will have to see significant improvements in order to achieve practical operational performance metrics, such as mean time to interrupt of an application and mean time to failure of the platform.

Memory Memory, not processing capability, is the fundamental challenge to building an exascale system. Projecting the current trends for today's universal memory technology (DRAM), exascale system parameters will push the boundaries of memory capacity, energy efficiency, and performance. Furthermore, disk storage, when considered as another level of the storage hierarchy, may also fail to provide adequate performance, reliability, and energy efficiency, by the end of this decade. As a result, future exascale systems may have a dramatically reduced memory capacity per core, which in turn, increases communication and lowers computational efficiencies.

Recent technological advances in memory technology and system integration, such as Spin-Torque-Transfer RAM (STT-RAM), phase-change RAM (PC-RAM), and Memristors (R-RAM), can address some of these issues by minimizing the energy required to store data persistently when compared to DRAM. Some of these technologies can also increase memory capacity. These improvements can also add new application and software capabilities, such as fault tolerance through node-distributed checkpointing, rather than flushing memory state to rotating disk. In addition, new hardware integration techniques, such as 3D stacking, can provide the opportunity for higher memory bandwidth to processing elements, while maintaining reasonable thermal densities.

Interconnect One critical problem is the growing disparity between the energy required for computation versus the energy required to move communication. At every level -- on opposite sides within a chip, between chips on a motherboard, and connecting racks in a datacenter – the energy dissipated in moving data around is not expected to improve significantly compared with the energy consumed locally in performing computation.

Further research into technological advances, such as 3D integration and silicon photonics, can address interconnect power problem by minimizing the energy required to traverse across the interconnect medium (For example, through-silicon vias result in minimal vertical distances; photonic links exhibit low channel loss for long distances). Energy-efficient transceiver circuits that actuate these novel electrical interfaces then become the dominant consumer in the link budget. Other areas include the reliability/resiliency of these new interconnect interfaces, and architecture co-design of control mechanisms that can optimize the link power for a required bandwidth

Resilience Resilience covers a broad scope ranging from hardware reliability to techniques employed to mitigate hardware and software failures. Traditionally, checkpoints have been used to minimize recalculation due to failure, but even if FIT rates remain constant, the shear number of components in an Exascale platform potentially reduces platform interrupt frequency to minutes rather than hours or days. Addressing resilience for exascale platforms requires a multi-facetted research approach. To achieve an acceptable interrupt frequency, research is necessary to increase the FIT rates for individual hardware components. Research should also be directed at new and novel approaches to application resilience including a move from bulk synchronous processing to models that are insensitive to individual component interrupts. Fault tolerant algorithms, for example, algorithms that tolerate soft errors or can specify portions of the algorithm that must produce correct results should also be investigated. Finally, advances in existing Reliability Availability and Serviceability (RAS) systems are necessary.

Conclusions

The path to exascale brings a host of new challenges, and magnifies many of the existing ones. Peak power and energy usage have become key drivers for all aspects of the system. Energy concerns are driving an accelerating pace in the increase in parallelism as we continue into the multi-/many-core era. System balance is also becoming harder to maintain as the cost of data movement begins to dominate energy usage. Successfully negotiating these challenges will require codesign across all levels of the system. However, because of long hardware design cycles, the opportunity to impact exascale component architectures is short and rapidly developing a codesign strategy with vendors is of utmost importance.

Acknowledgments

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by Los Alamos National Security, LLC, for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-TR-474731