LLNL-TR-471682

NNSA ASC Exascale Environment Planning, Applications Working Group, Report February 2011

C. H. Still, A. Arsenlis, R. B. Bond, M. J. Steinkamp, S. Swaminarayan, D. E. Womble, A. E. Koniges, J. R. Harrison, J. H. Chen

February 28, 2011

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

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).