LLNL-TR-472494

Tools and Tool Support for the Exascale Era

M. Schulz

March 8, 2011

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

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.

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.