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.