7th IESP Meeting Working Group Report on:

Revolutionary Approaches: Punctuated Equilibrium or Continuous Evolution

Thomas L. Sterling, Indiana University
Bronis R. de Supinski, Lawrence Livermore National Laboratory
version-4

1. Introduction

Projection of the possible path forward for HPC towards Exascale computing is guided by two histories. One expresses the success of incremental techniques of improvement over the last two decades in advancing delivered computing performance, at least for some applications and their algorithms. The second exposes as many as half a dozen paradigm shifts over the extended period of 60 to 70 years when progress in underlying device technologies catalyzed dramatic changes in the way computing was organized and conducted and computing systems were designed and operated. The evolutionary path successfully supported an era, *Pax MPI*, of stability with almost dependable improvements through incremental changes. The latter revolutionary path over the extended history also provided marked events of dramatic improvement but through disruptive change that demanded redefinition of structures and methods. These two histories, both valid, provide contradictory lessons and present the HPC community with the challenge of determining which applies to the potential accomplishment prior to the end of this decade of realizing sustained Exaflops performance. It is well understood that if an incremental sequence of enhancements to the status quo can satisfy the needs of such an achievement then it is far preferable to the alternative more risky and disruptive strategy. However, if an evolutionary trajectory is inadequate to this goal, then a revolutionary although uncertain path is essential even in the presence of incurred costs to refactoring systems, applications, and methods. While the broad consensus favors an evolutionary strategy, a minority is addressing the more radical alternative. At the 7th meeting of the International Exascale Software Project (IESP) conducted in Cologne Germany, a working group was organized from among the meeting participants to consider the potential needs and approaches of a revolutionary strategy for Exascale software. This brief note documents the initial findings of this forum.

The international HPC community dedicated to the realization of Exascale computing by the end of this decade is divided on the issue of the approach and strategy of achieving this goal. Four dominant issues prevail: These are the socialization and controversy of change, the Exascale challenges that may demand revolution, the candidate system areas where revolution may be achieved, and the disruptive impact of such revolutionary changes and how they can be mitigated. A détente is required between those demanding cautionary progress and those insisting on change in spite of fracturing the means of scalability beyond current practices. More than gap filling may be required but only through responsible methods that include continuity with existing legacy codes, models, and skills. Unfortunately a degree of bipolarization has been insinuated into the community-wide dialog that inhibits the formulation of a shared constructive middle ground. Fear of one extreme or the other has precluded the development of a rational all encompassing strategy to address the daunting challenges while maintaining the continuity of productive capability from passed software investment in applications and environments. This note attempts to begin the dialog that may bridge the current chasm and seek to derive the needed techniques.

2. Opportunities for Possible Revolutionary Approaches

Due to shortness of time the working group on "Revolutionary Approaches" focused on those areas that appeared to require necessary change beyond incremental extensions of current and recent practices. These were identified in part by the admittedly subjective criterion of what scared the participants most about trying to make it possible to reach an Exaflops performance before 2020. Each issue as it was put forth was discussed and is reported here in the cases where there was not strong disagreement. In that sense, this is a conservative take on controversial ideas; perhaps an oxymoron in itself. Six issues are highlighted here.

2.1 Execution models

The disruptive events in the history of supercomputing that mark the phase changes of HPC in each case resulted in a new paradigm of computing. A form of reflecting a computing paradigm is an "execution model." An execution model is a set of guiding principles that govern the roles and relationships of the integral component layers of a computing system and guide their interoperability and possible co-design. Each execution model responds to advances in underlying enabling technologies that provide new opportunities for progress in performance while also imposing some challenges that the execution model must address to achieve its potential. Among the performance factors are four that every execution model addresses, presented here as performance degradation. These are starvation, latency, overhead, and waiting due to contention for shared resources. Starvation requires the model to exhibit sufficient parallelism to keep system resources productively busy. Latency needs to avoid blocking for the period of access requests whether to local storage or for remote services. Overhead is the amount of critical time required to manage parallel resources and concurrent tasks. Every execution model attacks each of these problems and with different approaches that depend on the implications of the base-level technologies and possible hardware architectures that may be derived using them. The execution model impacts co-design across programming models and architectures as well as the supporting system software. While the current predominant execution model is Communicating Sequential Processes it is not clear what is the right one to best address such challenges as multi-core sockets or GPU accelerators. The right answer must come from the semantics of the application information graph and not just the hardware side of the system definition.

2.2 Changing the way we think

More broadly an area considered potentially in need of revolution is changing the way we think about the merger of systems and programming. Conventionally the user believes that he/she controls the machine. It is recognized that in the future we may have to relinquish this potentially over simplistic assumption. To exploit dynamic behavior to benefit from runtime information may have to accept non-determinacy and variability of the path to execution but still producing dependable answers, at least within bounded error bars. Thus a change of culture and attitude may become necessary where we do not control every cycle but rather influence the execution to find a path to the right answer, while delivering substantially greater performance than conventional practices.

2.3 Incorporating Intelligent Methods

As systems become more complex, embrace heterogeneity, exhibit highly varying latencies and overheads, where contention at hot spots may not be predictable, the use of on-the-fly selection among alternatives at multiple levels may become essential through the integration of intelligent techniques. Intelligent control involves detection and measurement of conditions on a continuing basis throughout a computation, goal driven objective functions that determine what is to be achieved and how to determine progress towards such goals, policies for selecting among alternative approaches or paths, and mechanisms for supporting introspective operation. Such methods must operate in real time but not impose significant overhead or any potential benefits will be dissipated. Objective functions are reflected at multiple levels from simple autonomic low-level responses to high-level complex decisions reflecting difficult choices. For future HPC, intelligent on-the-fly control must reflect a spatially as well as temporal awareness from a data perspective and may even require some hardware support for operational knowledge and predictive control. One approach may involve the implementation of an Information Backplane that provides a protocol between system layers for mutual dynamic introspective behavior.

2.4 Operating System

The vast majority of deployed supercomputing systems employ one of several versions of the Unix operating system with Linux dominating the Top-500 list. While continually evolving, the basic architecture of this critical software was established more than 30 years ago. The challenge of managing a billion cores is unprecedented as well as the scale and layers of memory and their interconnection. Because of the inadequacies of conventional Unix-like operating systems a number of variations have been pursued. Among these are lightweight kernels and user runtime systems for improved efficiencies. But many other challenges such as name space management, fault tolerance through graceful degradation, protection for security, dynamic resource management, and energy control all add new factors requiring integration. During much of the last two decades on clusters and MPPs systems have really been collectives of many operating systems running side by side instead of a single system image with some umbrella scheduling package. In addition, system and core architectures are undergoing significant change with both multicore and heterogeneous computing elements emerging of significant interest. The role and methods of future OS software in addressing these additional complexities has yet to be resolved. Future parallel programming languages may require new classes of service from the underlying OS as programming models change in the light of scaling and efficiency requirements. Changes in operating systems may be proactive with new concepts in system control or they may be reactive in support of dramatic changes in architecture, programming paradigms, and runtime systems all of which rely on the OS in some form. In either case, the OS may experience revolutionary change as it continues to play a fundamental role in HPC systems now and in to the future.

2.5 Programming Models

Parallel programming models change in response to advances in system structure to represent parallelism, data structures, and control flow in response to technology progress. This may be seen as a broader set of system changes commensurate with a new execution model or simply an alternative means of crafting parallel programs for a new class of architecture. In either case, a programming model provides a necessary level of abstraction, coupling the demands of an application with the resource capabilities and characteristics of the physical system resources. Nowhere is there more controversy than in the future of programming methodologies for Exascale computing. A core but by no means only requirement is to devise a set of semantics of parallelism that will yield concurrency at a sustained level in excess of a billion-way for Exascale. This involves forms of parallelism, means of sequencing operations and the conditions under which they may be performed, synchronization constructs. But this also entails the interrelationship of control flow with data structures and their distribution. While it is expected that some application algorithms may be extended through conventional practices or their incremental advancement to satisfy this criterion, there are many others that probably can't as demonstrated by some of today's important strong-scaled problems. But there is much more to a programming environment than just concurrency, as important as this is. And much has been gleaned over the last two decades that should inform, perhaps even support, future HPC programming. Of particular importance will be a migration path for legacy codes and programming models to new classes of Exascale systems to run seamlessly and without alteration. While it is not expected that such applications will benefit fully from innovations, they must run as well or better than native implementations of the programming model. Also critical is the means of interoperability between new and old program modules and libraries to support incremental improvements of conventional programs to new modalities, but one piece at a time. This also permits new applications to build on top of established and proven libraries where necessary initially. Far from discarding everything in the past and slowly replacing it, a bridge should be built to cross from the developments of the past to the possibilities of the future.

2.6 Correctness and Debugging

One objective widely thought to require a revolutionary approach is the critical challenges of achieving correctness through methods of verification and debugging. Frankly, this does not require Exascale performance to impose this need; it is with us today. But the idea of managing billions of quite possibly distinct threads in an asynchronous computational soup is daunting. Even the concept of correctness may have to be replaced with boundedness and quasi reproducibility rather than absolute exactness. Tools for verification and validation, error detection and debugging, and confidence in answers at varying scales requires new models, quite possibly revolutionary, for attacking this problem.

3. Related Factors and Issues

Several related issues were discussed concerning factors for realizing the potential revolution in computing systems and techniques responsibly in support of continued growth in computing capability. Some of the most prominent are touched on here:

3.1 Managing Revolution

Whatever happens, the work has to continue to get done. If supercomputers of the future fail to deliver increased performance on user applications, they will not be deployed and the field will stall, perhaps irrevocably as did commercial aviation in the 1960s from which flight times have not significantly improved in half a century. The notion of the oxymoron of an "incremental revolution" may be essential by which the workload continues to be performed but for which significant improvements in performance are achieved concomitant with investment in code refactoring. The employment of a discipline incorporating gradual steps will permit planning over time by which organizations, agencies, and nations can project their respective paths across HPC generations. Before change is imposed, a destination has to be identified; this possibly through preparatory research and proof-of-concept development. At launch time of the revolutionary system concepts, it is imperative that the process begins with something credible. It is unlikely to expect that there is not much of current practice that will convey to future methodologies and these are to be retained. Nonetheless, for something new experience genesis, something has to die. Those practiced in the arts of HPC use over time will adopt new practices and adapt their respective problems to them in order to exploit the orders of magnitude benefits in science, technology, commerce, and security. This will require a culture change but one that can be managed and transitioned rather than disruptive through bridging methods and education. Many of the benefits may be achieved early through the rewriting of widely used libraries by groups of experts to the service of the broad community. High level programming models, even declarative or domain specific programming interfaces may also mitigate the challenge of transition. All of these approaches and others constitute means of managing the possibly essential HPC revolution to Exascale capability.

3.2 Intelligence in Systems

Where do we put intelligence in future Exascale systems? There is little doubt that except in some special cases future systems will rely, perhaps heavily, on runtime functionality realized through new runtime system software in cooperation with OS and architecture driven by compiler and user programming interfaces. With the incorporation of dynamic control brings the realization of the opportunity for imbuing systems with higher order intelligent policies of operation. Many methods are already being pursued to self-adapt codes to underlying core architectures, for work scheduling, and for dynamic load balancing. Other areas such as fault management and energy optimization are potential candidates. A new layer of intelligence managing all of these now disparate goals supported at many levels may provide a quasi self-aware environment through introspective means capable of optimizing potentially conflicting requirements to achieve overall “best” operation.

3.3 Over Reliance on Compiler Technology

Historically compilers have dictated system usage. Remarkable progress in advanced compiler technology over several decades has bred a culture of strong reliance on this system layer. But it has also limited progress in efficiency and scalability to those issues with which compiler could effectively deal. With limited predictability, compilers are often conservative in their choice of operation. Problems like inter-procedural analysis and automatic parallelization are still topics of research even after decades of admittedly good work. With the emergence of new runtime systems applied to the field of HPC, an entirely new class of opportunities is becoming available. Yet the common culture of many to rely on compilers alone is hindering future directions to deliver superior capabilities. Compilers cannot alone provide the necessary understanding to fully proscribe how large-scale systems must operate. But with runtime systems and even some advances in hardware support compilers will continue to play a crucial role, although in some ways different from the past, in guiding the computation of Exascale systems and their applications.

3.4 Persistent Storage

There is an entire array of application challenges that are largely data-intensive and bound by the capabilities of the large-scale storage systems. Our field has relied on the basic model of mass storage in the form of file systems derived from the 1960s with some semantics that go back to the 1940s (e.g., rewind). Even with important strides such as MPI-IO, HPF5, and RAID mass storage is still treated as an entirely separate system than the in-memory computation. Indeed, in many cases it is an entirely separate system. The problem will only be aggravated as the imbalance between compute throughput capability and main memory capacity continues to degrade and future applications demand the processing of enormous data sets. Once a specialty area, out of core computation may become increasingly prevalent. Vertical movement of data, especially with the likely addition of NVRAM technology adding yet another layer to the hierarchy, may be seen to increase with respect to today's practices. The principal distinguishing characteristic between to two classes of storage is that of ephemeral data (in-memory) versus persistent data (spinning bits). Far more lightweight and agile transition between the two storage technologies may be required and new programming methods unifying these classes in a single name space may be essential to achieve dynamic reactive data movement. Here too, revolutionary strategies may be needed to fully embrace the potential of Exascale computing for the widest array of applications.

4. Conclusions

We, the International community considering the necessary capabilities of Exaflops (and other measures) performance, are uncertain at this time of the software architecture and the software components of which it is comprised for in support of Exascale systems anticipated by the end of this decade. While not proven, it is expected by some that some elements of such a software system may reflect revolutionary methods in order to support part or all of the workload applied. This brief note has summarized the tentative conclusions of a self-selected representative group on behalf of the IESP as a product of the 7th IESP meeting working-group on "Revolutionary Approaches." Four questions were considered important to answer by IESP as it establishes its long-term strategy including: 1) socialization of the topic to eliminate the current climate of polarization of the issue, 2) the areas of functional need most likely to demand some form of revolutionary technique, 3) possible revolutionary approaches to address requirements, and 4) means of mitigating the potentially disruptive effects of revolutionary approaches. This report due to limited time addressed questions 2 & 3 with an emphasis on the latter. Although revolutionary in direction, the deliberations reflected in this note were conservative in that they stressed the importance of responsible paths to realizing Exascale including means of mitigating the potential disruptive consequences. Also, only those topics, issues, and directions that had a sense of working group consensus are included in this report. If this group has correctly represented the need for revolutionary approaches as part of the path to Exascale, it also urges that there is much more to be learned through targeted research before any such approach can with confidence be undertaken by the international HPC community.

Contributors:

- Pete Beckman, Argonne National Laboratory
- Ron Brightwell, Sandia National Laboratories
- Barbara Chapman, University of Houston
- Jack Dongarra, University of Tennessee, Knoxville
- Anshu Dubey, University of C
- Al Geist, Oak Ridge National Laboratory
- Andrew Jones, NAG
- Alice Koniges, Lawrence Berkeley National Laboratory
- Jesus Labarta, Barcelona Supercomputing Centre
- Bob Lucas, USC Information Sciences Institute
- Paul Messina, Argonne National Laboratory
- Hiroshi Nakamura, University of Tokyo
- Hiroshi Nakashimi, Kyoto University
- Thomas Sterling, Indiana University (Co-Chair)
- Shinji Sumimoto, Fujitsu Inc.
- Bronis de Supinski, Lawrence Livermore National Laboratory (Co-Chair)
- Kenjiro Taura, University of Tokyo
- Rajeev Thaker, Argonne National Laboratory
- Anne Trefethen, Oxford University
- Vladimir Voevodin, Moscow State University