INTERNATIONAL EXASCALE SOFTWARE PROJECT

ROADMAP 1.1

Jack Dongarra
Pete Beckman
Terry Moore
Patrick Aerts
Giovanni Aloisio
Jean-Claude Andre
David Barkai
Jean-Yves Berthou
Taisuke Boku
Bertrand Braunschweig
Franck Cappello
Barbara Chapman
Xuebin Chi

Alok Choudhary
Sudip Dosanjh
Thom Dunning
Sandro Fiore
Al Geist
Bill Gropp
Robert Harrison
Mark Hereld
Michael Heroux
Adolfy Hoisie
Koh Hotta
Yutaka Ishikawa
Fred Johnson

Sanjay Kale
Richard Kenway
David Keyes
Bill Kramer
Jesus Labarta
Alain Lichnewsky
Thomas Lippert
Bob Lucas
Barney Maccabe
Satoshi Matsuoka
Paul Messina
Peter Michielse
Bernd Mohr

Matthias Mueller
Wolfgang Nagel
Hiroshi Nakashima
Michael E. Papka
Dan Reed
Mitsuhisa Sato
Ed Seidel
John Shalf
David Skinner
Marc Snir
Thomas Sterling
Rick Stevens
Fred Streitz

Bob Sugar
Shinji Sumimoto
William Tang
John Taylor
Rajeev Thakur
Anne Trefethen
Mateo Valero
Aad van der Steen
Jeffrey Vetter
Peg Williams
Robert Wisniewski
Kathy Yelick

SPONSORS

Office of Science
U.S. Department of Energy

NSF

ANR

cea

CRAY
THE SUPERCOMPUTER COMPANY

eDF

EPSRC
Engineering and Physical Sciences
Research Council

FUJITSU

INRIA

KYOTO UNIVERSITY
FOUNDED 1897

GENCI

NVIDIA.

RIKEN

東京大学
THE UNIVERSITY OF TOKYO

筑波大学

Table of Contents

The International Exascale Software Project Roadmap

1. Introduction

The technology roadmap presented here is the result of more than a year of coordinated effort within the global software community for high-end scientific computing. It is the product of a set of first steps taken to address a critical challenge that now confronts modern science and is produced by a convergence of three factors: (1) the compelling science case to be made, in both fields of deep intellectual interest and fields of vital importance to humanity, for increasing usable computing power by orders of magnitude as quickly as possible; (2) the clear and widely recognized *inadequacy* of the current high end software infrastructure, in all its component areas, for supporting this essential escalation; and (3) the near complete lack of planning and coordination in the global scientific software community in overcoming the formidable obstacles that stand in the way of replacing it. At the beginning of 2009, a large group of collaborators from this worldwide community initiated the *International Exascale Software Project* (*IESP*) to carry out the planning and the organization building necessary to solve this vitally important problem.

With seed funding from key government partners in the United States, European Union and Japan, as well as supplemental contributions from some industry stakeholders, we formed the IESP around the following mission:

> *The guiding purpose of the IESP is to empower ultra-high resolution and data-intensive science and engineering research through the year 2020 by developing a plan for (1) a common, high-quality computational environment for petascale/exascale systems and (2) catalyzing, coordinating, and sustaining the effort of the international open source software community to create that environment as quickly as possible.*

There exist good reasons to think that such a plan is urgently needed. First and foremost, the magnitude of the technical challenges for software infrastructure that the novel architectures and extreme scale of emerging systems bring with them is daunting [5, 8]. These problems, which are already appearing on the leadership-class systems of the US National Science Foundation (NSF) and Department of Energy (DOE), as well as on systems in Europe and Asia, are more than sufficient to require the wholesale redesign and replacement of the operating systems, programming models, libraries, and tools on which high-end computing necessarily depends.

Second, the complex web of interdependencies and side effects that exist among such software components means that making sweeping changes to this infrastructure will require a high degree of coordination and collaboration. Failure to identify critical holes or potential conflicts in the software environment, to spot opportunities for beneficial integration, or to adequately specify component requirements will tend to retard or disrupt everyone's progress, wasting time that can ill afford to be lost. Since creating a software environment adapted for extreme-scale systems (e.g., NSF's Blue Waters) will require the collective effort of a broad community, this community must have good mechanisms for internal coordination.

Third, it seems clear that the scope of the effort must be truly international. In terms of its rationale, scientists in nearly every field now depend on the software infrastructure of high-end computing to open up new areas of inquiry (e.g., the very small, very large, very hazardous, very complex), to dramatically increase their research productivity, and to amplify the social and economic impact of their work. It serves global scientific communities who need to work together on problems of global significance and leverage distributed resources in transnational configurations. In terms of feasibility, the dimensions of the task—totally redesigning and recreating, in the period of just a few years, the massive software foundation of computational science in order to meet the new realities of extreme-scale computing—are simply too large for any one country, or small consortium of countries, to undertake on its own.

The IESP was formed to help achieve this goal. Beginning in April 2009, we held a series of three international workshops, one each in the United States, Europe, and Asia in order to work out a plan for doing so. Information about and the working products of all these meetings can be found at the project website, www.exascale.org. In developing a plan for producing a new software infrastructure capable of supporting exascale applications, we charted a path that moves through the following sequence of objectives:

1. *Make a thorough assessment of needs, issues and strategies*: A successful plan in this arena requires a thorough assessment of the technology drivers for future peta/exascale systems and of the short-term, medium-term, and long-term needs of applications that are expected to use them. The IESP workshops brought together a strong and broad-based contingent of experts in all areas of HPC software infrastructure, as well as representatives from application communities and vendors, to provide these assessments. As described in more detail below, we also leveraged the substantial number of reports and other material on future science applications and HPC technology trends that different parts of the community have created in the past three years.
2. *Develop a coordinated software roadmap*: The results of the group's analysis have been incorporated into a draft of a coordinated roadmap intended to help guide the open source scientific software infrastructure effort with better coordination and fewer missing components. This document represents the current version of that roadmap.
3. *Provide a framework for organizing the software research community*: With a reasonably stable and complete version of the roadmap in hand, we will endeavor to develop an organizational framework to enable the international software research community to work together to navigate the roadmap and reach the appointed destination—a common, high quality computational environment that can support extreme-scale science on extreme-scale systems. The framework will include elements such as initial working groups, outlines of a system of governance, alternative models for shared software development with common code repositories, and feasible schemes for selecting valuable software research and encouraging its translation into usable, production-quality software for application developers. This organization must also foster and help coordinate R&D efforts to address the emerging needs of users and application communities.
4. *Engage and coordinate with the vendor community in cross-cutting efforts*: To leverage resources and create a more capable software infrastructure for supporting exascale science, the IESP is committed to engaging and coordinating with vendors across all of its other objectives. Industry stakeholders have already made contributions to the workshops (i.e.. objectives 1 and 2 above) and we expect similar, if not greater participation, in the effort to create a model for cooperation as well as coordinated R&D programs for new exascale software technologies.
5. *Encourage and facilitate collaboration in education and training*: The magnitude of the changes in programming models and software infrastructure and tools brought about by the transition to peta/exascale architectures will produce tremendous challenges in the area of education and training. As it develops its model of community cooperation, the IESP plan must, therefore, also provide for cooperation in the production of education and training materials to be used in curricula, at workshops and on-line.

This roadmap document, which focuses objectives 1 and 2 above, represents the main result of the first phase of the planning process. Although some work on tasks 3–5 has already begun, we plan to solicit, and expect to receive in the near future, further input on the roadmap from a much broader set of stakeholders in the computational science community. This version of the roadmap begins that process by including more extensive input from the science application community, international funding agencies, and vendor partners. The additional ideas and information we gather as the roadmap is disseminated are likely to produce changes that need to be incorporated into future iterations of the document as plans for objectives 3–5 develop and cooperative research and development efforts begin to take shape.

2. Destination of the IESP Roadmap

The metaphor of the roadmap is intended to capture the idea that we need a representation of the world, drawn from our current vantage point, in order to better guide us from where we are now to the destination we want to reach. Such a device is all the more necessary when a large collection of people, not all of whom are starting from precisely the same place, need to make the journey. In formulating such a map, agreeing on a reasonably clear idea of the destination is obviously an essential first step. Building on the background knowledge that motivated the work of IESP participants, we define the goal that the roadmap is intended to help our community reach as follows:

> *By developing and following the IESP roadmap, the international scientific software research community seeks to create a* ***common, open source software infrastructure for scientific computing*** *that enables leading-edge science and engineering groups to develop applications that exploit the full power of the exascale computing platforms that will come on-line in the 2018–2020 timeframe. We call this integrated collection of software the extreme-scale/exascale software stack, or* ***X-stack****.*

Unpacking the elements of this goal statement in the context of the work done so far by the IESP reveals some of the characteristics that the X-stack must possess, at minimum:

- *The X-stack must enable suitably designed science applications to exploit the full resources of the largest systems*: The main goal of the X-stack is to support groundbreaking research on tomorrow's exascale computing platforms. By using these massive platforms and X-stack infrastructure, scientists should be empowered to attack problems that are much larger and more complex, make observations and predictions at much higher resolution, explore vastly larger data sets, and reach solutions dramatically faster. To achieve this goal, the X-stack must enable scientists to use the full power of exascale systems.

- *The X-stack must scale both up and down the platform development chain*: Science today is done on systems at a range of different scales, from departmental clusters to the world's largest supercomputers. Since leading research applications are developed and used at all levels of this platform development chain, the X-stack must support them well at all these levels.

- *The X-stack must be highly modular, so as to enable alternative component contributions*: The X-stack is intended to provide a *common* software infrastructure on which the entire community builds its science applications. For both practical and political reasons (e.g., sustainability, risk mitigation), the design of the X-stack should strive for modularity that makes it possible for many groups to contribute and accommodate more than one choice in each software area.

- *The X-stack must offer open source alternatives for all components in the X-stack*: For both technical and mission oriented reasons, the scientific software research community has long played a significant role in the open source software movement. Continuing this important tradition, the X-stack will offer open source alternatives for all of its components, even though it is clear that exascale platforms from particular vendors may support, or even require, some proprietary software components as well.

3. Technology Trends and Their Impact on Exascale

The design of the extreme-scale platforms that are expected to become available in 2018 will represent a convergence of technological trends and the boundary conditions imposed by over half a century of algorithm and application software development. Although the precise details of these new designs are not yet known, it is clear that they will embody radical changes along a number of different dimensions as compared to the architectures of today's systems and that these changes will render obsolete the current software infrastructure for large-scale scientific applications. The first step in developing a plan to ensure that appropriate system software and applications are ready and available when these systems come on line, so that leading edge research projects can actually use them, is to carefully review the underlying technological trends that are expected to have such a transformative impact on computer architecture in the next decade. These factors and trends, which we summarize in this section, provide essential context for thinking about the looming challenges of tomorrow's scientific software infrastructure; therefore, describing them lays the foundation on which subsequent sections of this roadmap document builds.

3.1 Technology Trends

In developing a roadmap for the X-stack software infrastructure, the IESP has been able to draw on several thoughtful and extensive studies of impacts of the current revolution in computer architecture [5, 7]. As these studies make clear, technology trends over the next decade – broadly speaking, increases of 1000X in capability over today's most massive computing systems, in *multiple* dimensions, as well as increases of similar scale in data volumes – will force a disruptive change in the form, function, and interoperability of future software infrastructure components and the system architectures incorporating them. The momentous nature of these changes can be illustrated for several critical system-level parameters:

- *Concurrency*– Moore's law scaling in the number of transistors is expected to continue through the end of the next decade, at which point the minimal VLSI geometries will be as small as five nanometers. Unfortunately, the end of Dennard scaling means that clock rates are no longer keeping pace, and may in fact be reduced in the next few years to reduce power consumption. As a result, the exascale systems on which the X-stack will run will likely be composed of hundreds of millions of ALUs. Assuming there are multiple threads per ALU to cover main-memory and networking latencies, applications may contain ten billion threads.

- *Reliability* – System architecture will be complicated by the increasingly probabilistic nature of transistor behavior due to reduced operating voltages, gate oxides, and channel widths/lengths resulting in very small noise margins. Given that state-of-the-art chips contain billions of transistors and the multiplicative nature of reliability laws, building resilient computing systems out of such unreliable components will become an increasing challenge. This cannot be cost-effectively addressed with pairing or TMR; rather, it must be addressed by X-stack software and perhaps even scientific applications.

- *Power consumption* – Twenty years ago, HPC systems consumed less than a megawatt. The Earth Simulator was the first such system to exceed 10 MW. Exascale systems could consume over 100 MW, and few of today's computing centers have either adequate infrastructure to deliver such power or the budgets to pay for it. The HPC community may find itself measuring results in terms of power consumed, rather than operations performed. The X-stack and the applications it hosts must be conscious of this situation and act to minimize it.

Similarly dramatic examples could be produced for other key variables, such as *storage capacity*, *efficiency*, and *programmability*.

More important, a close examination shows that changes in these parameters are interrelated and not orthogonal. For example, scalability will be limited by efficiency, as are power and programmability. Other cross correlations can be perceived through analysis. The DARPA Exascale Technology Study [5] exposes power as the pacesetting parameter. Although an exact power consumption constraint value is not yet well defined, with upper limits of today's systems on the order of 5 megawatts, increases of an order of magnitude in less than 10 years will extend beyond the practical energy demands of all but a few strategic computing environments. A politico-economic pain threshold of 25 megawatts has been suggested (by DARPA) as a working boundary. With dramatic changes to core architecture design, system integration, and programming control over data movement, best estimates for CMOS-based systems at the 11-nanometer feature size is a factor of 3 to 5 times this amount. One consequence is that clock rates are unlikely to increase substantially in spite of the IBM Power architecture roadmap with clock rates between 0.5 and 4.0 GHz a safe regime and a nominal value of 2.0 GHz appropriate, at least for some logic modules. Among the controversial questions is how much instruction-level parallelism (ILP) and speculative operation is likely to be incorporated on a per processor core basis and the role of multithreading in subsuming more of the fine-grained control space. Data movement across the system, through the memory hierarchy, and even for register-to-register operations will likely be the single principal contributor to power consumption, with control adding to this appreciably. Since future systems can ill afford the energy wasted by data movement that does not advance the target computation, alternative ways of hiding latency will be required in order to guarantee, as much as possible, the utility of every data transfer. Even taking into account the wastefulness of today's conventional server-level systems and the energy gains that careful engineering has delivered for systems such as Blue Gene/P, an improvement on the order of 100X, at minimum, will still be required.

As a result of these and other observations, exascale system architecture characteristics are beginning to emerge, though the details will become clear only as the systems themselves actually develop. Among the critical aspects of future systems, available by the end of the next decade, which we can predict with some confidence are the following:

- Feature size of 22 to 11 nanometers, CMOS in 2018
- Total average of 25 picojoules per floating point operation
- Approximately 10 billion-way concurrency for simultaneous operation and latency hiding
- 100 million to 1 billion cores
- Clock rates of 1 to 2 GHz
- Multithreaded, fine-grained concurrency of 10- to 100-way concurrency per core
- Hundreds of cores per die (varies dramatically depending on core type and other factors)
- Global address space without cache coherence; extensions to PGAS (e.g., AGAS)
- 128-petabyte capacity mix of DRAM and nonvolatile memory (most expensive subsystem)
- Explicitly managed high-speed buffer caches; part of deep memory hierarchy
- Optical communications for distances > 10 centimeters, possibly intersocket
- Optical bandwidth of 1 terabit per second
- Systemwide latencies on the order of tens of thousands of cycles
- Active power management to eliminate wasted energy by momentarily unused cores
- Fault tolerance by means of graceful degradation and dynamically reconfigurable structures
- Hardware-supported rapid thread context switching
- Hardware-supported efficient message-to-thread conversion for message-driven computation
- Hardware-supported, lightweight synchronization mechanisms
- 3-D packaging of dies for stacks of 4 to 10 dies each including DRAM, cores, and networking

Because of the nature of the development of the underlying technology most of the predictions above have an error margin of +/-50% or a factor 2 independent of specific roadblocks that may prevent reaching the predicted value.

3.2 Science Trends

A basic driver of the IESP is the fact that the complexity of advanced challenges in science and engineering continues to outpace our ability to adequately address them through available computational power. Many phenomena can be studied only through computational approaches; well-known examples include simulating complex processes in climate and astrophysics. Increasingly, experiments and observational systems are finding that not only are the data they generate exceeding petabytes and rapidly heading toward exabytes, but the computational power needed to process the data is also expected to be in exaflops range.

A number of reports and workshops have identified key science challenges and applications of societal interest that require computing at exaflops levels and beyond [1-3, 6, 9]. Here we summarize some of the significant findings on the scientific necessity of exascale computing; we focus primarily on the need for the software environments needed to support the science activities. DOE held eight workshops in the past year that identified science advances and important applications that will be enabled through the use of exascale computing resources. The workshops covered the following topics: climate, high-energy physics, nuclear physics, fusion energy sciences, nuclear energy, biology, materials science and chemistry, and national nuclear security. The US National Academy of Sciences published the results of a study in the report "The Potential Impact of High-End Capability Computing on Four Illustrative Fields of Science and Engineering" [6]. The four fields were astrophysics, atmospheric sciences, evolutionary biology, and chemical separations.

Likewise, NSF has embarked on a petascale computing program that has funded dozens of application teams through its Peta-Apps and PRAC programs, across all areas of science and engineering, to develop petascale applications, and is deploying petaflops systems, including Blue Waters, expected to come on-line in 2011. It has commissioned a series of task forces to help plan for the transition from petaflops to exaflops computing facilities, to support the software development necessary, and to understand the specific science and engineering needs beyond petascale.

Similar activities are seen in Europe and Asia, all reaching similar conclusions: significant scientific and engineering challenges in both simulation and data analysis already exceed petaflops and are rapidly approaching exaflop-class computing needs. In Europe, the Partnership for Advanced Computing in Europe (PRACE) involves twenty partner countries, supports access to world-class computers, and has activities aimed at supporting multi-petaflops and eventually exaflops-scale systems for science. The European Union (EU) is also planning to launch projects aimed at petascale and exascale computing and simulation. Japan has a project to build a 10-petaflop system and has historically supported the development of software for key applications such as climate. As a result, scientific and computing communities, and the agencies that support them in many countries, have been meeting to plan joint activities that will be needed to support these emerging science trends.

To give a specific and timely example, a recent report[1] states that the characterization of abrupt climate change will require sustained exascale computing in addition to new paradigms for climate change modeling. The types of questions that could be tackled with exascale computing (and cannot be tackled adequately without it) include the following:

- ¨How do the carbon, methane, and nitrogen cycles interact with climate change?
- ¨How will local and regional water, ice, and clouds change with global warming?
- ¨How will the distribution of weather events, particularly extreme events, determine regional climate change with global warming?
- ¨What are the future sea-level and ocean circulation changes?

[1] *Science Prospects and Benefits of Exascale Computing*, ORNL/TM-2007/232, December 2007, page 9, http://www.nccs.gov/wp-content/media/nccs_reports/Science%20Case%20_012808%20v3__final.pdf

Among the findings of the astrophysics workshop and other studies are that exascale computing will enable cosmology and astrophysics simulations aimed at the following:

- Measuring the masses and interactions of dark matter
- Understanding and calibrating supernovae as probes of dark energy
- Determining the equation of state of dark energy
- Measuring the masses and interactions of dark matter
- Understanding the nature of gamma-ray bursts

Energy security. The search for a path forward in assuring sufficient energy supplies in the face of a climate-constrained world faces a number of technical challenges, ranging from issues related to novel energy technologies, to issues related to making existing energy technologies more (economically) effective and safer, to issues related to the verification of international agreements regarding the emission (and possible sequestration) of CO_2 and other greenhouse gases. Among the science challenges are the following:

- Verification of “carbon treaty” compliance
- Improvement in the safety, security, and economics of nuclear fission
- Improvement in the efficiency of carbon-based electricity production and transportation
- Improvement in the reliability and security in the (electric) grid
- Nuclear fusion as a practical energy source

Computational research will also play an essential role in the development of new approaches to meeting future energy requirements (e.g., wind, solar, biomass, hydrogen, and geothermal), which in many cases will require exascale power.

Industrial applications, such as simulation-enhanced design and production of complex manufactured systems and rapid virtual prototyping, will also be enabled by exascale computing. To characterize materials deformation and failure in extreme conditions will require atomistic simulations on engineering time scales that are out of reach with petascale systems.

A common theme in all of these studies of the important science and engineering applications that are enabled by exaflops computing power is that they have complex structures and present programming challenges beyond just scaling to many millions of processors. For example, many of these applications involve multiple physical phenomena spanning many decades of spatial and temporal scale. As the ratio of computing power to memory grows, the “weak scaling,” which has been exploited for most of the last decade, will increasingly give way to “strong scaling,” which will make scientific applications increasingly sensitive to overhead and noise generated by the X-stack. These applications are increasingly constructed of components developed by computational scientists worldwide, and the X-stack must support the integration and performance portability of such software.

3.3 Key Requirements Imposed by Trends on the X-Stack

The cited trends in technology and applications will impose severe constraints on the design of the X-stack. Below are cross-cutting issues that will affect all aspects of system software and applications at exascale.

- **Concurrency**: A 1000x increase in concurrency for a single job will be necessary to achieve exascale throughput. New programming models will be needed to enable application groups to address concurrency in a more natural way. This capability will likely have to include “strong scaling” because growth in the volume of main memory will not match that of the processors.

This in turn will require minimizing any X-stack overheads that might otherwise become a critical Amdahl fraction.

- **Energy**: Since much of the power in an exascale system will be expended moving data, both locally between processors and memory as well as globally, the X-stack must provide mechanisms and APIs for expressing and managing data locality. These will also help minimize the latency of data accesses. APIs also should be developed to allow applications to suggest other energy saving techniques, such as turning cores on and off dynamically, even though these techniques could result in other problems, such as more faults/errors.
- **Resiliency**: The VLSI devices from which exascale systems will be constructed will not be as reliable as those used today. All software, and therefore all applications, will have to address resiliency in a thorough way if they are to be expected to run at scale. Hence, the X-stack will have to recognize and adapt to errors continuously, as well as provide the support necessary for applications to do the same.
- **Heterogeneity**: Heterogeneous systems offer the opportunity to exploit the extremely high performance of niche market devices such as GPUs and game chips (e.g., STI Cell) while still providing a general-purpose platform. An example of such a system today is Tokyo Tech's Tsubame, which incorporates AMD Opteron CPUs along with Clearspeed and Nvidia accelerators. Simultaneously, large-scale scientific applications are also becoming more heterogeneous, addressing multiscale problems spanning multiple disciplines.
- **I/O and Memory**: Insufficient I/0 capability is a bottleneck today. Ongoing developments in instrument construction and simulation design make it clear that data rates can be expected to increase by several orders of magnitude over the next decade. The memory hierarchy will change based on both new packaging capabilities and new technology. Local RAM and NVRAM will be available either on or very close to the nodes. The change in memory hierarchy will affect programming models and optimization.

3.4 Relevant Politico-Economic Trends

The HPC market is growing at approximately 11 percent per year. The largest-scale systems, those that will support the first exascale computations at the end of the next decade, will be deployed by government computing laboratories to support the quest for scientific discovery. These capability computations often consume an entire HPC system and pose difficult challenges for concurrent programming, debugging and performance optimization. Thus, publicly funded computational scientists will be the first users of the X-stack and have a tremendous stake in seeing that suitable software exists, which is the raison d'être for IESP.

In the late 1980s, the commercial engineering market place, spanning diverse fields such as computer aided engineering and oil reservoir modeling, used the same computing platforms and often the same software as the scientific community. This is far less the case today. The commercial workload tends to be more capacity oriented, involving large ensembles of smaller computations. The extreme levels of concurrency necessary for exascale computing suggests that this trend may not change, so it is not clear how much demand for those features of the X-stack unique to exascale computing from commercial HPC users. On the other hand, the HPC vendor community is eager to work with, and leverage the research and development effort of, the IESP software community. To that end, plans for cooperation and coordination between the IESP software and the HPC vendor community are being developed; we summarize the current state of this discussion in Section 6.

4. Formulating Paths Forward for X-Stack Component Technologies

In this section of the roadmap, the longest and most detailed, we undertake the difficult task of translating the critical system requirements for the X-stack, presented in Section 3, into concrete recommendations for research and development agendas for each of the software areas and necessary components of the X-stack. The roadmapping template we used roughly follows the approach described in the excellent study from Sandia National Laboratories by Garcia and Bray [4]. Accordingly, the discussion of each component or area is divided into the following parts:

- **Technology and science drivers**: The impacts of the critical technology trends and science requirements must be described and analyzed for each software area and/or component of the X-stack. These impacts represent technology and science drivers for each such area/component of the X-stack, and each must be evaluated in terms of how well or poorly current technologies address the target requirements and where the obstacles to progress lie.
- **Alternative R&D strategies**: Once the technology and science drivers are identified and studied, the different possible lines of attack on the problems and challenges involved, insofar as we can see them today, need to be described and explored.
- **Research and development agenda recommendations**: Alternative R&D strategies in each area need to be evaluated and ranked, and actual plans, including specific milestones, must be drawn up. Clearly these plans must take into account a variety of factors, many of which have been (or should be) described elsewhere in the roadmap.
- **Cross-cutting Considerations**: Many of the parts of the X-stack will have interdependencies and cross-cutting effects related to other component areas; allusions to these effects are likely to be laced or scattered through the previous three subsections. In many cases it will be desirable to break out a summary of these considerations as a separate section in order to highlight gaps or to ensure that activities are suitably coordinated. This version of the roadmap focuses on four such cross-cutting areas: resiliency, power/total-cost-of-ownership, performance, and programmability.

4.1 System Software

The system software list is often described as that software that manages system resources on behalf of the application but is usually transparent to the user. For the purposes of mapping the road to a viable X-stack, we include under this heading the operating system, runtime system, I/O system, and essential interfaces to the external environment (e.g., data repositories, real-time data streams, and clouds). Each of these areas is treated in turn below.

4.1.1 Operating Systems

4.1.1.1 Technology Drivers for Operating Systems

Increasing importance of effective management of increasingly complex resources – Exascale systems will increase the complexity of resources available in the system. Moreover, in order to attain the benefits offered by an exascale system, effective management of these resources will be increasingly important.

As an example, consider the execution environment presented by an exascale system. Current systems provide hundreds of thousands of nodes with a small number of homogeneous computational cores per node. Exascale systems will increase the complexity of the computational resource in two dimensions. First, the core count per node will increase substantially. Second, the cores most likely will be heterogeneous (e.g., combining stream-based cores with traditional cores based on load/store). In addition to increasing the complexity of the computational resources, the resources shared between the computational resources (e.g., the memory bus) can have a far greater impact on performance.

Besides the changes in the resources provided by an exascale system, the programming models will undergo an evolution. In particular, non-MPI programming models will undoubtedly have increasing presence in exascale systems. The only trends clear at the present time are that there will be an increasing emphasis on datacentric computations and that programming models will continue to emphasize the management of distributed-memory resources. Given the evolution in programming models, we can also expect that individual applications will incorporate multiple programming models. For example, a single application may incorporate components that are based on MPI and other components that are based on shared memory. The particular combination of programming models may be distributed over time (different phases of the application) or space (some of the nodes run MPI; others run shared memory).

The purpose of an operating system is to provide a bridge between the physical resources provided by a computing system and the runtime system needed to implement a programming model. Given the rapid change in resources and programming models, a common operating system must be defined for the exascale community. This will provide the exascale community with a common set of APIs that can be used by a runtime system to support fully autonomic management of resources, including adaptive management policies that identify and react to load imbalances and the intermittent loss of resources (resilience). In order to achieve this goal, the APIs supported by the operating system must expose low-level resource APIs, and the runtime must be aware of the context (within the application) of a specific computation.

4.1.1.2 Alternative R&D Strategies for Operating Systems

Several approaches could be adopted in the development of a community operating system for exascale systems. One approach is to evolve an existing OS, for example, Linux, Plan 9, or IBM's Compute Node Kernel. An alternative approach is to start with a new design to address the specific needs of exascale systems. The first approach has the advantage that the APIs provided by the OS have already been defined, and many runtime implementations have already been developed for the APIs. Moreover, these operating systems also provide drivers for many of the devices that will be used in exascale systems (e.g., the PCI bus). However, because the APIs are based on the resources provided by previous systems (many of these operating systems were defined nearly a half-century ago), they may not provide the appropriate access to the resources provided by an exascale system. In the end, it is likely that a hybrid approach, which builds on APIs and existing code bases and redesigns and modifies the most specialized components, will prevail.

The operating system must maintain a high degree of flexibility. This flexibility can be accomplished only by minimizing the resource management strategies that are required by the operating system.

4.1.1.3 Recommended Research Agenda for Operating Systems

The first step in the development of a common OS for the exascale community is to develop a framework for the OS. This should be undertaken by a small collection of researchers who have significant experience in implementing HPC operating systems.

One of the critical challenges in developing HPC operating systems is our inability to study the impact of resource management decisions "at scale." To remedy this problem, we will need to develop a full system simulation capability. A number of efforts are addressing parts of the full-system simulation capability; however, these efforts need to be coordinated to ensure that they provide the needed capability.

The most critical APIs provided by the community OS will include APIs to support inter- and intranode communication, inter- and intranode thread management, and explicit management of the memory hierarchy provided by the entire system. APIs to support energy management and resilience will also be critical; however, these APIs require more experience; and, as such, their final definition should be deferred until the final stages of this research activity.

The critical research areas in which substantial, if not groundbreaking, innovations will be required in order to reach this goal are the following:

- Fault tolerant/masking strategies for collective OS services
- Strategies and mechanisms for power/energy management
- Strategies for simulating full-scale systems
- General strategies for global (collective) OS services

Timeframe	Targets and Milestones – Operating Systems
2010-11	Community-defined framework for HPC operating systems that defines a set of core components and coarse-grained APIs for accessing the resources provided by an HPC system.
2012-13	Scalable, full-system simulation environment that can be used to evaluate resource management mechanisms at scale.
2014-15	APIs for fine-grained management of internode communication, thread management, and memory hierarchy management.
2016-17	APIs for fine-grained management of power (energy) and resilience.
2018-19	At least one runtime system that provides global, autonomic management of the resources provided by an HPC system. This runtime system should provide for transparent resilience in the presence of failing resources.

4.1.2 Runtime Systems

4.1.2.1 Technology and Science Drivers for Runtime Systems

The role of a runtime system is to act on behalf of the application in matching its algorithm's characteristics and requirements to the resources that the system makes available in order to optimize performance and efficiency. By programming to the runtime system's interface, application developers are freed from the mundane but often difficult jobs of task scheduling, resource management, and other low-level operations that would force them to think about the computer rather than the science they are trying to do. As the description of the technology trends and science requirements above suggests, it will be extremely challenging to create runtime systems that can continue to fulfill this role. The design of tomorrow's runtime systems will be driven not only by dramatic increases in overall system hierarchy and high variability in the performance and availability of hardware components but also by the expected diversity application characteristics, the multiplicity of different types of devices, and the large latencies caused by deep memory subsystems. Against this background, two general constraints on design and operation of X-stack runtime systems need to be highlighted: power/energy constraints and application development cost. The first constraint establishes the objective for X-stack runtimes as *maximizing the achieved ratio of performance to power/energy consumption*, instead of raw performance alone. The second constraint means that X-stack runtimes must focus on supporting the execution of the *same program* at all levels of the platform development chain, which is in line with the basic criteria for X-stack success (Section 2).

The runtime system is the part of the software infrastructure where actual and more accurate information is available about system resource availability and performance; thus this component has the potential to make better-informed decisions on behalf of the application. To achieve this goal, however, and successfully insulate application programmers from the complexities of extreme scale platforms, X-stack runtimes will have to incorporate much more intelligence than current technologies support. The real challenge will be to use this added intelligence effectively in the limited timeframe that is typically available while the application runs.

4.1.2.2 Alternative R&D Strategies for Runtime Systems

Several directions can and should be tried in order to create X-stack runtimes that achieve the targeted scale. The most obvious division of alternatives is in terms of degree of hierarchy, namely, a flat runtime model (e.g., message passing) and a hierarchical model (e.g., shared memory within a node and message passing across nodes). In the latter case, the runtime hierarchy can have the same underlying model at different levels or use different models at different levels. Flat and hierarchical alternatives are not totally opposed in direction, and a hybrid approach can certainly benefit from the flat approach pushing its capabilities to the limits. Another set of alternatives to explore are general-purpose runtime systems, on the one hand, and application type- or area-specific (or customizable) runtime systems, capable of more effectively exploiting platform resources relative to special sets of needs, on the other.

4.1.2.3 Recommended Research Agenda for Runtime Systems

Challenging research topics include heterogeneity, asynchrony, reduction of process management and synchronization overheads, provision of shared naming/addressing spaces, optimization of communication infrastructure, scheduling for parallel efficiency and memory efficiency, memory management, and application-specific customizability. These topics can be grouped into four priority research directions:

- **Heterogeneity:**
  - Research challenge: X-stack runtime systems will have to work on several different platforms, each of them heterogeneous, and this will certainly prove challenging. The objective will be to optimize the application's utilization of resources for best power/performance by helping the application adapt to and exploit the level of granularity supported by the underlying hardware.
  - Anticipated research directions: Anticipated research includes unified/transparent accelerator runtime models; exploitation of systems with heterogeneous (functionality/performance) nodes and interconnects; scheduling for latency tolerance and bandwidth minimization; and adaptive selection of granularity. This type of research is also expected to be useful for homogeneous multicores.
  - Impact: Research in this area broadens the portability of programs, decoupling the specification of the computations from details of the underlying hardware, thereby allowing programmers to focus more exclusively on their science.
- **Load balance:**
  - Research challenge: A key challenge is to adapt to the unavoidable variability in time and space (processes/processors) of future applications and systems. This will have to be done with the objective of optimizing resource utilization and execution time.
  - Anticipated research directions: Directions include general-purpose, self-tuned runtimesthat detect imbalance and reallocate resources (e.g., cores, storage, DVFS, bandwidth) within or across processes and other entities at the different levels; virtualization-based mechanisms to support load balancing; minimization of the impact of temporary resource shortages, such as those caused (at different granularity levels) by OS noise; and partial job preemptions.
  - Impact: Research in this area will result in self-tuned runtimes that will counteract, at fine granularity, unforeseen variability in application load and availability and performance of resources, thus reducing the frequency at which more expensive application-level rebalancing approaches will have to be used. Globally, this will significantly reduce the effort requested of the programmers to achieve efficient resource utilization and ensure that the resources that cannot be profitably used are returned to the system to be reallocated.

- **Flat runtimes:**
  - Research challenge: A major challenge is to increase the scalability of existing and proposed models with respect to the resources required for their implementation and the overheads they incur. This includes the need to optimize the utilization that is currently achieved of internal resources such as adaptors and communication infrastructure. Also, typical practices today where globally synchronizing calls (barriers, collectives) represent big limitations at large scale will have to be addressed.
  - Anticipated research directions: Research will be needed in optimization of resources and infrastructure for implementing the runtime (e.g., memory used by message-passing libraries, overheads for process management and synchronization) and increased usage of prediction techniques to accelerate the runtime, or at least introduction of high levels of asynchrony and communication/computation overlap (i.e., asynchronous MPI collectives, APGAS approaches, data-flow task based approaches). Also needed will be hierarchical implementations of flat models (e.g., thread based MPI, optimization of collective operations) and adaptation of communication subsystems to application characteristics (routing, mapping, RDMA, etc.)
  - Impact: Research in this area will result in increased scalability of basic models. Techniques developed here will also be beneficial for the hierarchical approach. Globally, this will extend the lifespan of existing codes and will help absorb the shock that the transition to exascale represents.
- **Hierarchical/hybrid runtimes:**
  - Research challenge: A key challenge is how to properly match the potentially different semantics of the models at different levels as well as to ensure that the scheduling decisions taken at each of them have positive interference. This matching between models must also consider the actual matching of the execution to the underlying hardware structure and ensure efficient utilization of the resources for any target machine. One of the challenges that motivates the hierarchical approach is constraining the size of the name/address spaces (i.e., ranks, amount of shared state) while still providing a fair level of concurrency and flexibility within each level.
  - Anticipated research directions: Anticipated research includes experimentation on different hierarchical integrations of runtimes to support models, such as MPI+other threading or task based models, threading models+accelerators, MPI+threading+accelerators, MPI+PGAS, and hierarchical task-based models with very different task granularities at each level; techniques to support encapsulation, modularity, and reuse; selection of appropriate number of entities (processes/threads) at each level in the hierarchy and the mapping to actual hardware resources; automatic memory placement, association, and affinity scheduling.
  - Impact: Research in this area will result in effectively matching the execution to the available resources, enabling smooth migration paths from today’s flat codes.

Timeframe	Targets and Milestones – Runtime Systems
2010-11	Asynchrony/overlap: Demonstrate for both flat and hierarchical models 3x scalability for strong scaling situations where efficiency would otherwise be very low (i.e., 30%) Why: Fighting variance is a lost battle: learn to live with it. Synchronous behavior is extremely sensitive to variance and does not forgive communication delays.
2012-13	Heterogeneity: Demonstrate that the "same" code can be run on different heterogeneous systems. Locality-aware scheduling: demonstrate that automatic locality aware scheduling can get a factor of 5x in highly NUMA memory architectures. Why: By then, everybody will have experienced that rewriting the same application for every new platform is not a viable alternative. Machines will have deep, noncoherent memory hierarchies, and we have to demonstrate we know how to use them.
2014-15	Optimizing runtime: general-purpose runtime automatically achieving load balance, optimized network usage, and communication/computation overlap, minimization of memory consumption at large scale, maximization of performance to power ratio, malleability, and tolerance to performance noise/interference on heterogeneous systems. Why: Complexity of systems will require automatic tuning support to optimize the utilization of resources, which will not be feasible by static, user-specified schedules and partitionings.
2016-17	Fault-tolerant runtime: tolerating injection rates of 10 errors per hour (cooperating with application provided information and recovery mechanisms for some errors). Why: By then systems will have frequent failures, and it will be necessary to anticipate and react to them in order that the application delivers useful results.
2018-19	Fully decoupling runtime: dynamically handling all types of resources such as cores, bandwidth, logical and physical memory or storage (i.e., controlling replication of data, coherency and consistency, changes in the layout as more appropriate for the specific cores/accelerators). Why: Underlying system complexity and application complexity will have to be matched in a very dynamic environment.

4.1.2.4 Cross-Cutting Considerations

The runtime functionality interacts with all cross-cutting areas.

- Power management: The runtime will be responsible for measuring the application performance and deciding the appropriate setups (frequency and voltage, duty cycles, etc.) for the knobs that the underlying hardware will provide.
- Performance: The runtime will have to be instrumented to provide detailed information to monitoring systems such that they can report appropriate measurements to upper levels of the resource management infrastructure (i.e., job scheduler) or to the user. The runtime will also need monitoring information about the performance of the computational activity of the application to select the most appropriate resource for them or to choose the appropriate power mode.
    - Resilience: The runtime will be responsible for implementing some fine-grained mechanisms (i.e., reissue failed tasks, control speculative state) as well as for deciding when to fire coarse-grained mechanisms and the actual amount of state they should handle.

- Programmability: The runtime will have to implement the features needed to support the various programming models used on exascale systems.

Global coordination between levels (architecture, runtime, compiler, job schedulers, etc.) is needed.

4.1.3 I/O Systems

4.1.3.1 Technology and Science Drivers for I/O Systems

Technology and science drivers for I/O systems include architectural alternatives for I/O systems, the underlying application requirements or purpose for doing I/O, I/O software stack, the expected capabilities of the devices, and fault resiliency. The data management (discussed in detail in the Scientific Data Management section), life cycle, and its future usage and availability also influence how I/O system software should be designed. Given the current state of I/O and storage systems in petascale systems, incremental solutions in most aspects are unlikely to provide the required capabilities in exascale systems. I/O architectures, when designed as separate and independent components from the compute infrastructure, have already been shown not to be scalable as needed. That is, traditionally I/O has been considered as a separate activity that is performed before or after the main simulation or analysis computation, or periodically for activities such as checkpointing, but still as separate overhead. This mindset in designing architectures, software, and applications must change if the true potential of exascale systems is to be exploited. I/O should be considered an integral activity to be optimized while architecting the system and the underlying software. File systems, which have mainly been adapted from the legacy (sequential) file systems with overly constraining semantics, are not scalable. Traditional interfaces in file systems and storage systems, or even in some cases higher-level data libraries, are designed to handle the worst-case scenarios for conflicts, synchronization, and coherence and mostly ignore the purpose of the I/O by an application, which is an important source of information for scaling I/O performance when millions of cores simultaneously access the I/O system. Emerging storage devices such as solid-state disks or SCMs have the potential to significantly alter the I/O architectures, systems, performance, and software system. These emerging technologies also have significant potential to optimize power consumption. Resiliency of an application under failures in an exascale system will depend significantly on the I/O systems—its capabilities, capacity, and performance—because saving the state of the system in the form of checkpoints is likely to continue as one of the approaches.

4.1.3.2 Alternative R&D Strategies for I/O Systems

Many R&D strategies at different levels of the architecture and software stack can potentially address the above technology drivers and for exascale systems. The metrics of I/O systems are performance, capacity, scalability, adaptability of applications, programmability, fault resiliency, and support for end-to-end data integrity.

1. Delegation and Customization within I/O Middleware: The best place for optimizing and scaling I/O is the middleware within user space because that is where most semantic data distribution, data usage, and access pattern information are available. The middleware is not only for the single-user space; it also cooperates with other user file I/O activities on the machine so that systemwide optimization can be performed. The concept of delegation within I/O middleware entails the use of a small fraction of the system on which the middleware exists and runs within the user space to perform I/O-related functions and optimizations on behalf of the applications. Using the application requirements, it can perform intelligent and proactive caching, data reorganization, optimizations, and smoothing of I/O accesses from burst to smooth patterns. This approach can provide services to the application in such a way that the application can customize the resources used based on its requirements. The delegation and customization approach also has the opportunity to perform various functions on data while it is being produced or to preprocess the data before it is consumed. The availability of multicore nodes provides the opportunity to use

one or more cores on each node, to perform I/O services, to use an exclusive set of select nodes, and to provide a range of customization options including locality enhancements.

2. Active Storage and Online Analysis: The concept of active storage is based on the premise that modern storage architectures might include usable processing resources at the storage nodes that can be exploited for performing various important tasks including data analysis, organization, and redistribution. This concept has significant potential to improve performance and knowledge discovery by exploiting the significant processing power within the caching and delegate nodes or within the storage system. The potential use of both significantly more memory and GPGPUs, as well as FPGA types of accelerators for data reformatting, subsetting, analysis, and searching, make it even more attractive. However, the potential for developing these should be explored within the runtime middleware (e.g., MPI-IO or higher-level libraries) or at the file system layer. These layers should be modified to provide appropriate interfaces to enable this capability. Online analytics can potentially reduce the need to store certain types of data if all the necessary information and knowledge from this data can be derived while it is available.

3. Purpose-driven I/O Software Layers: The traditional homogeneous I/O interfaces do not explicitly exploit the purpose of an I/O operation. A checkpointing I/O activity is different from an I/O activity, which stores data for future analysis using some other access pattern. An example of the latter is the use of data in analyzing a subset of variables along a time axis. Optimizations in the two activities may require different approaches by the software layers. The software layers from file systems, middleware, and higher should be modified by incorporating these capabilities and by exploiting the purpose of I/O.

4. Software Systems for Integration of Emerging Storage Devices: Emerging storage devices such as solid-state devices and storage class memories (SCMs) offer significant potential to improve performance, reduce power consumption, and improve caching; such devices can potentially reduce or eliminate explicit I/O activities and traffic on traditional disks if they are transparently incorporated within the I/O software layers. Research and development of newer I/O models and different layers of software systems, including file systems and middleware, is important for the exploitation of these devices. Various approaches must be investigated along with the various options for using these devices in the exascale architecture (e.g., an SCM device being part of each node's memory hierarchy or them being part of a separate section of the architecture that have these devices). These systems have implications in how various layers are designed and optimized and should be topics for research and development. Furthermore, power optimizations approaches in software layers should be explored.

5. Extension of Current File Systems: Efforts may be made to extend current file systems to address the parallelism and performance needed. However, given the current capabilities and performance of these files systems, which are derived from conservative and reactive designs and with strict sequential semantics, the chances of success of this approach are limited.

6. New Approach to Scalable Parallel File Systems: Research is needed into newer models, interfaces, and approaches that are not limited by sequential semantics and into consistency models that incorporate newer and highly scalable metadata techniques and that can exploit information available from user and higher levels as well as can incorporate newer storage devices and hierarchies.

7. Incorporation of I/O into Programming Models and Languages: Important research areas include language features and programming model capabilities in which users can use the programming models and language to provide the I/O requirements, access patterns, and other high-level information. Ideally, it should be possible for compilers to use these enhanced models to optimize I/O, pipeline I/O, and intelligently schedule I/O to maximize overlap with other computations. Moreover, the models should be usable on multicore architectures, where they can

be exploited to utilize cores for enhancing I/O performance and specify online analysis functions on delegate systems of active storage.

8. Wide-Area I/O and Integration of External Storage Systems: Scalable techniques are needed in which parallelism in accessing storage devices is integrated with parallelism for network streaming. Also important is integrating parallel streaming of data over the network, using similar principles as those in parallel I/O.

4.1.3.3 Recommended Research Agenda for I/O Systems

The recommended research agenda for I/O systems is all items above except item 5.

Timeframe	Targets and Milestones – I/O Systems
2010-11	▪ I/O delegation concepts in various I/O software layers ▪ New abstractions and approaches to parallel file systems ▪ Protocols for parallel data transfers for wide-area I/O
2012-13	▪ Initial I/O runtime and file systems for SCM/SSD devices ▪ Develop purpose-driven I/O software layers ▪ I/O delegation optimizations, including analytics and data-processing capabilities ▪ Programming language and model constructs for I/O integration
2014-15	▪ Active storage alternatives in runtime and file systems ▪ Customizable I/O APIs and implementations ▪ Tuned I/O API implementations demonstrated with new memory hierarchy components that include SCM ▪ Scalable tools with parallel I/O and parallel streaming for wide-area I/O
2016-17	▪ Newer programming models and languages capabilities enabled for active storage ▪ Fault resiliency and low-power capabilities added in the I/O software layers ▪ Integration of online analysis within active storage architecture with new storage devices (SCM) ▪ Protocol conversion capabilities for wide-area I/O
2018-19	▪ File systems and runtime software layers for exascale I/O optimized for new storage devices ▪ Power-performance optimization capabilities in I/O software layers ▪ Scalable software layers for wide-area I/O integrated with schedulers with special-purpose protocols for external networks

4.1.3.4 Cross-Cutting Considerations

The architecture of the systems in general, and for storage and I/O systems and their use of emerging devices in particular, will influence the I/O system software. Architectures should consider the issues outlined above in designing I/O systems. I/O-related communication and storage device usage will significantly influence power optimizations. The I/O system software clearly has implications for resiliency, the schedulers, the operating systems, and programming models and languages.

4.1.4 Systems Management

Systems management comprises a broad range of technical areas. We divided these topics into five categories to be able to more tightly describe the challenges, research directions, and impact of each: (1)

"resource control and scheduling," which includes configuring, start-up, and reconfiguring the machine, defining limits for resource capacity and quality, provisioning the resources, and workflow management; (2) "security," which includes authentication and authorization, integrity of the system, data integrity, and detection of anomalous behavior and inappropriate use; (3) "integration and test," which involves managing and maintaining the health of the system and performing continuous diagnostics; (4) "logging, reporting, and analyzing information," where the data consists of a static definition of machine (what hardware exists and how it is connected), the dynamic state of the machine (what nodes are up, what jobs are running, how much power is being used), RAS (Reliability, Availability, Serviceability) events (warning or error conditions, alerts), and session log information (what jobs ran, how long, how much resource they consumed); and (5) "external coordination of resources," which is how the machine coordinates with external components (e.g.,how the HPC machine fits in a cloud) and comprises a common communication infrastructure, reporting errors in a standardized way, and integrating within a distributed computing environment.

4.1.4.1 Technology and Science Drivers for System Management

In addition to the fundamental drivers mentioned above (scale, component count failure rates, etc.) there are additional technical challenges for system management. The first challenge is the fact there is a "real-time" component to all system management tasks, with the time periods ranging from microseconds to weeks. Whether it is running the right task at the right time, getting the right data to the right place at the right time, getting an exascale system integrated and tested in a timely manner, or responding to attempted security compromises, all system management tasks have to be responsive. On exascale systems the tasks also have to be automatic and proactive in order to stay within response limits.

Another driver for exascale system management is that the limited resources that have been used in system resource control and scheduling for the gigascale to petascale—processors and computational operations—are no longer the most constrained resource. DARPA studies listed in this report document that data movement, rather than computational processing, will be the constrained resource at exascale. This is especially true when power and energy are taken into account as limiting design and total cost of ownership criteria. Hence, resource control and management—and the utilization logs for resources—have to change focus to communications and data movement. Today, most of the data movement components of a system are shared and not scheduled, while most of the computation resources are controlled and dedicated to an application. That may not be the best solution going to exascale, but we do not know.

System management also has to ensure system integrity, a major factor of which is system security (security is used here in the sense of open-system cyber security). Exascale systems will be so varied and complex that in order to protect their correct operation, security features (such as authentication and authorization, intrusion detection and prevention, and data integrity) will have to be built into the many components of the system. The "defense-in-depth" concepts that are successful for facilitywide security will have to be extended throughout the exascale system without impinging on performance or function.

System complexity is another driver at exascale. HPC systems are exceedingly complex and susceptible to small perturbations having extraordinary impact on performance, consistency, and usability. Taking the number of transistors multiplied by the number of lines of code simultaneously in use as a measure of complexity, exascale systems will be four orders[2] of magnitude more complex than their petascale predecessors. The system manager's job is to manage this complexity in order to provide consistent high performance and quality of service. Without the reinvention of many of the tools used today and the invention of new tools, system managers will not be able to meet those expectations.

[2] Estimates of today's vendor-supplied system software contain between 3 and 18 million lines of code. If one assumes that each line of code generates 10 machine instructions, that is 30–180 million instructions. Further assume that OS functions use 1/30th of a second (and applications the rest)), there are 1 – 6 million instructions per second in every node. Today's machines have 1,000 to 10,000 OS images, with some having closer to 100,000. A simplistic complexity value might be considered as number of instructions * number of images. Today this is $6*10^{14}$. At exascale, there may be 10,000,000 nodes. If the code complexity only doubles for exascale, the complexity is $1.2*10^{14}$, four orders of magnitude more complex in the simplest case.

4.1.4.2 Alternative R&D Strategies for System Management

The obvious alternative is to take an evolutionary approach to extending terascale and petascale system management practices. This will result in significant inefficiencies in exascale systems, extended outages, and low effectiveness. As a metric, one can extend the Performability (Performance * Reliability) measure to also include the effectiveness of resource allocation and consistency (PERC). Given the evolutionary approach, it is likely that exascale systems will have a PERC metric within an order of magnitude of petascale because of much less efficient resource management, much less consistency, and much less reliability.

Another approach could be to import technical approaches from other domains such as the telecommunications industry, which provisions data movement and bandwidth as key resources. Another domain that has technology to offer is real-time systems, which use control theory, statistical learning techniques, and other methods to manage limited resources in a proactive manner. As a final example, some cyber-security intrusion detection technology also has potential to offer stateful, near-real-time analysis of activities and logs. Data mining and data analytics also have potential to offer point solutions to managing large amounts of event data and identifying key factors that need to be addressed at high levels.

4.1.4.3 Recommended Research Agenda for System Management

Here we present a representative list of research problems that will need to be addressed in order to achieve the goals of exascale system management presented above.

Category 1: "Resource control and scheduling" and "External coordination of resources"

- Better characterize and manage nontraditional resources such as power and I/O bandwidth
- Determine how to manage and control communication resources – provision and control, different for HPC than for WAN routing
- Determine and model real-time aspects of exascale system management and feedback for resource control
- Develop techniques for dynamic provision under constant failure of components
- Coordinate resource discovery and scheduling with exascale resource management

The first area for research in Category 1 is obtaining a better characterization of non-traditional resources such as power and I/O data motion. Related is research into how to control that data motion. As part of that study, the community needs to identify whether additional hardware enhancements should be designed such as network switches that allow multiplexing streams by percentage utilization. In part, the control will need to build on the results of the ability to better characterize the data motion, but it may also proceed somewhat independently. Another research initiative that must be undertaken is determining how to integrate the characterization and perform the control in real time. The most challenging piece of research is determining how to keep the system running in the presence of constant failures. System management in the exascale timeframe ideally must be able to proactively determine failures and reallocate resources. If a failure is not predetected, the system management infrastructure must be able to detect, isolate, and recover from the failure, by allocating additional equivalent resources. While effort is underway in the application space to handle failures, system management research should target presenting applications with machines where failures are corrected transparently by reallocating working resources to replace the failed ones. Moreover, in order to integrate the HPC machine into a larger infrastructure, research should be undertaken to provide standardized reporting of machine definitions and capabilities that exist in a globally scheduled environment.

Category 2: "Security"

- Provide fine-grained authentication and authorization by function/resources
- Provide security verification for software built from diverse components
- Provide appropriate "defense in depth" within systems without performance or scalability impact
- Develop security-focused OS components in X-stack
- Assess and improve end-to-end data integrity
- Determine guidelines and tradeoffs of security and openness (e.g., grids)

For a system as complex as an exascale system, the risk of undetected compromise is too high to rely on traditional security at the borders (login nodes). Fine-grained authentication and authorization by function and for each resource are needed through all software and hardware components of the system. This has to be lightweight so as not to restrict or slow authorized use or limit scalability, while at the same time comprehensive enough to assure as complete protection as possible. The security model should be to monitor and react rather than restrict, as much as possible, and to enable open, distributed ease of use.

Because the system is expected to be built from diverse components, created by different communities, security verification of software components will have to be done efficiently. This will require a means to verify correct functioning, but the challenge will be to accommodate the scale and the diversity of use of an exascale resource.

Since other needs point to creating a novel HPC operating system, a critical feature to be considered is making a security focused OS. There may also be hardware assist features that can combine finer-grained control and access management. Security requires integrity, so end-to-end data integrity has to be included. Moreover, new analysis to provide the right balance between security and openness for distributed computing (e.g., grid, web services) needs to be explored.

Category 3: "Integration and test" and "Logging, reporting, and analyzing information"

- Determine key elements for exascale monitoring
- Continue mining current and future petascale failure data to detect patterns and improvements
- Determine methods for continuous monitoring and testing without affecting system behavior
- Investigate improving information filters; provide stateful filters for predicting potential incorrect behavior
- Determine statistical and data models that accurately capture system behavior
- Determine proactive diagnostic and testing tools

The first research initiative that must be undertaken to reach the end goal of proactive failure detection is determining the key elements that need to be monitored. Much work has already occurred in this area. Thus, determination of what will be required for exascale is needed, with potentially new items identified. Additional research must be encouraged in the field of mining failure data to determine patterns and develop methodologies for doing so. Because the amount of collected data will be vast in the exascale era, investigations for filters and statistical models must occur. In both cases, it is critical to significantly reduce the volume while accurately capturing system behavior and not losing critical events. For filtering, there is a critical need to develop stateful techniques, where the dynamic state of the machine determines what events the filter provides. Techniques must be researched to allow this monitoring, filtering, and analysis to occur in real time without affecting application behavior running on the system. These research initiatives need to feed research of proactively determining where failures will occur by monitoring and analyzing filtered data.

Timeframe	Targets and Milestones – Systems Management
2010-11	Category 1: Creation and validation of an analytic model and simulation capability for exascale resource management that spans different implementations of job and resource management systems. This work will enable experimentation of alternative designs that will accelerate implementation in the later timeframes. Category 2: Fine-grained authentication—being able to provide access to individual or classes of resources to a single user or to groups of users.
2012-13	Category 1: Dynamic provisioning of traditional resources—being able to provide applications with more nodes and memory on the fly. Category 3: Unified framework for event collection: providing a community-agreed-upon standard format for events across machines and subsystems within a machine.
2014-15	Category 1: Expanded analytic model and simulation capability for exascale resource management to include external coordination of services. Category 2: Security validation of diverse components, providing a methodology for the different components in a system to ensure that security is maintained across the components. Category 3): Model and filter for event analysis, using the data produced by the above unified framework to produce models representing the system for understanding how different policies would impact the system, and providing filters, some of which should be stateful (dependent on the dynamic state of the machine).
2016-17	Category 1: Integrated nontraditional resources, such as bandwidth and power: by using the above models and filters, and the dynamic provisioning of resources, providing the ability to manage new important resources such as power and data motion. Category 3: Continual monitoring and test so that, by building on the unified framework for collecting data and filters, real-time monitoring and testing of the machine are provided.
2018-19	Category 1: Continual resource failure and dynamic reallocation—using the above proactive failure detection as input, and the above described dynamic provisioning of traditional and nontraditional resources to provide the ability to keep the machine running in the presence of continual failures by reallocating resources. Category 2: Hardware support for full system security. “Defense in depth” security is needed so that security does not rely solely on access control to the machine. Also needed is development of end-to-end methodologies including integrated hardware to protect all components of the machine. Category 3: Proactive failure detection—building on the above continual monitoring and analysis tools to provide the ability to predict failures.

4.1.4.4 Cross-Cutting Considerations

System management functionality crosses all aspects of the vertical integration—performance, usability/programmability, resilience, and power. System management directly impacts consistency and total cost of ownership as well. In addition, system management relies heavily on accumulating, integrating, and analyzing disparity data from all system components as well as all applications wanting to use the system. Multilevel analysis of system usage, subsystem activities, and component and subsystem health are needed to provide dynamic resource provision and to facilitate consistent and correct execution of application tasks.

4.1.5 External Environments

The term *external environments* refers to the essential interfaces to remote computational resources (e.g., data repositories, real-time data streams, high-performance networks, and computing clouds) that advanced applications may need to access and utilize. The use of such resources is already typical for many high-end applications, and they form a critical part of the working environment for most, if not all, major research communities.

In the following, "distributed data repositories" are discussed. This discussion complements the views presented in, for example, Section 4.3.3, Application Support: Scientific Data Management. In particular, while in Section 4.3.3 the main focus is on data management issues and challenges in the data center, this section discusses data management issues (i.e., data access/integration) with regard to external data repositories (data grids/clouds) and how the exascale roadmap can pave the way toward a transparent, efficient. and integrated management of scientific databases distributed across data centers, data grids, data clouds, and other external data repositories. Cross-references with other parts of this roadmap can be identified in Section 4.3.2 (with special regard to metatools and new data analysis approaches), Section 4.4 (cross-cutting dimensions such as resilience, performance. and programmability), and Section 4.1.2 (I/O systems with special regard to active storage and online analysis as well as scalable file systems).

4.1.5.1 Technology and Science Drivers for External Environments

Exascale cyber infrastructures will face important and critical challenges, both from computational and data perspectives. Increasingly complex and parallel scientific codes will lead to the production of a huge amount of data. For instance, climate change scientists are expected to generate hundreds of exabytes of data (distributed across several centers) through heterogeneous storage resources (located in data centers as well as in external environments such as data grids and data clouds) for access, analysis, post-processing, and other scientific activities. Collections of data will be stored at different sites and made available to users for further analysis.

The large volume of data and the time needed to locate, access, analyze, and visualize this data will greatly impact the scientific productivity. Significant improvements in the data management field therefore will be critical to increase research productivity in solving complex scientific problems.

Since external environments will play an important role in the scene, several challenges must be taken into account in developing the exascale roadmap context. The first challenge at such large scale is to provide efficient, scalable, resilient and transparent access to the external (with regard to the data center) and distributed (from a geographical point of view) data repositories. Exascale applications will have to efficiently manage and access data inside/outside the data center with a high level of performance and through common interfaces able to decouple fabric/middleware layers from the application one. Data centers will increasingly need to access to external data repositories to take advantage of a wide set of data collections. This should be made transparent, and this transparency represents a key challenge because the heterogeneity of the data environments is expected to further increase as it's directly connected with technology evolution.

Since external environments will play an important role in the scene, several challenges must be taken into account in developing the exascale roadmap context.

- The first challenge at such large scale is to provide efficient, scalable, resilient, and transparent access to the external (with regard to the data center) and distributed (from a geographical point of view) data repositories. Exascale applications will have to efficiently manage and access data inside/outside the data center with a high level of performance and through common interfaces able to decouple fabric/middleware layers from the application one. Data centers will increasingly need access to external data repositories to take advantage of a wide set of data collections. This should be made transparent, and this transparency represents a key challenge because the heterogeneity of the data environments is expected to further increase as it's directly connected with technology evolution.

- Related challenges that will become critical will be replication and distribution. At exascale, huge data repositories will be replicated and distributed across several sites to increase data availability, provide higher levels of fault tolerance and locality. For example, in the climate change domain, the CMIP5 data repositories will be replicated across the United States and Europe, and future scenarios will strongly rely on replication needs and schemas. Distribution and replication are expected to be strongly exploited in the near future; and, because of the scale and evolution of future exabyte systems, they represent a relevant challenge.
- Considering the wide variety of external data repositories available worldwide, uniform access in terms of common interfaces will be fundamental. The wide set of interfaces to data services is already a challenge. Because of the large-scale environment, the heterogeneity of the platforms, and the complexity of the exascale system, interoperability will play an important role in making highly feasible, transparent, and productive the interaction among all the involved components and services available inside data centers, data grid environments, and data clouds.
- Data portals are today the entry points to vast data collections for several institutions, data centers, and data clouds. In the exabyte era, stronger support and integration of scientific, collaborative, and social aspects are expected in the context of new scientific gateways. Social networking capabilities, poorly exploited today for scientific purposes, are strongly needed to increase the level of discussions, feedback, exchange of scientific results, and dissemination among groups. What is missing today is low-level and pervasive interoperability to enable data repositories in data centers, data grids, and data clouds to be transparently accessed and easily integrated in order to exploit new multidimensional and multidisciplinary research opportunities.
- Data knowledge and discovery will play a critical role as the number of data collections and the volume of data stored in distributed (heterogeneous) repositories becomes larger. A high number of (heterogeneous) metadata/ontologies sources (from different institutions/centers) are anticipated, which describe the available data collections with regard to different domains. Metadata provenance will increasingly become fundamental, in order to identify, trace, and record the history of data and the related processing and analysis steps in such a multifaceted environment. Automatic metadata extraction needs to be improved to support the data publication process at exascale data production rates. Semantic interoperability needs to be further addressed to make data integration a reality.
- Open access will become the key for effective sharing of data. At present, several restrictions and access policies make real sharing and easy access to the available data collections complicated, creating several nonconnected (isolated) islands of data repositories. This problem must be solved, while taking into account that access and usage policies must be preserved as well. What is missing is transparent and uniform management of such aspects across several countries and institutions.

4.1.5.2 Alternative R&D Strategies for External Environments

Access to data repositories in grid and cloud environments raises numerous challenges. In most cases, an evolutionary approach seems adequate if we consider the status of existing middleware and technologies and the production environments that have been built on top of them in several international initiatives in Europe, the United States, and Japan. Obviously, the scale and the requirements in the exabyte era will need a reengineering, extension, and improvement of several modules to make the integration feasible.

New efforts must be devoted to the intermediate layers (e.g., middleware) to have more interoperable, robust, and complete support to access the external data environments at exascale.

Since access to data grids and data clouds is a key element for external environments, the design of *common* interfaces (for middleware components) will be fundamental. What is crucial is the coexistence of standards and de facto standards and scientific and commercial actors, which makes more complex the entire realm. Stronger efforts in interoperability and standardization need to be globally sustained with a co-design approach supported by commercial and scientific partners. Such an approach will enable effective access to a larger set of external data repositories and environments. Metadata standards, domain-based ontologies. and the associated standardization and discussion processes must be strongly addressed. Such efforts will allow us to better describe, at exascale, data related to different scientific domains through a widely accepted, known, and adopted set of information.

Metadata standardization will be an enabling process for effective access and sharing of data, since it addresses search and discovery of data collections across different data sources. It is also a driving factor for interoperability, obviously implying the need to develop new tools, software. and services able to deal with such a new standard at exascale.

Also critical is further investigation into new algorithms, protocols, replication schemas, placement strategies, consistency protocols, lifetime issues, and dynamic aspects. At this layer, a standardized access to the external data environments will be needed, access that can be exploited to decouple replication aspects from the access ones.

4.1.5.3 Recommended Research Agenda for External Environments

The recommended research agenda focuses on three areas:

1. *Access to external data repositories*
   - Stronger effort in data delivery mechanisms, parallel data transfer, compression algorithms, efficient data protocols and data access services
   - More pervasive use of new and higher performance networks
   - Further activities on standard interfaces that will provide a stronger level of interoperability among different data repositories—an effective collaboration and co-design between industrial and scientific partners is recommended
   - Further work to make the middleware more robust, to transparently access heterogeneous data environments in data centers, data grids, and data clouds
2. *Replication and distribution of data*
   - Further investigation on new algorithms, protocols, replication schemas, and placement strategies, which are crucially needed at such large scale
   - Dynamic replication strategies based on historical information and usage patterns
   - Stronger need to deal with several kinds of transient failures (e.g., network and storage failures) providing efficient recovery procedures in case of faults, and better addressing resilience
3. *Scientific data gateways*
   - Collaborative, easy-to-use, integrated, social-based features, tailored on user access patterns and levels that are highly configurable
   - Complex and distributed dataflow support

- Knowledge mining and discovery, starting from advanced and integrated decision support systems
- Ability to represent the virtual place where people can work together; create communities; exploit a wide set of tools; and analyze, visualize, and compare data coming from data centers, grids, or cloud environments.

In short, the roadmap for distributed data repositories must move toward *extremely integrated, interoperable, and interdisciplinary data environments,* where the transparent integration of heterogeneous data sources (inside and outside the data center) will allow, at exascale, a better and deeper understanding of complex phenomena and problems.

Timeframe	Targets and Milestones – Distributed Data Repositories
2010-11	Workshops focused on the main topics of the Research Agenda for distributed data repositories. Metadata management, harvesting capabilities, ontology management, dynamic replica management, improved search and discovery capabilities, standardization activities on data services
2012-13	Advanced web access and workflow capabilities for scientific data portals, federated data management, interoperability among data services, semantic data integration services
2014-15	Resilient services for distributed data repositories, advanced ontology management, operational data gateways integrated, collaborative and community-oriented, stronger level of interoperability, new data analysis services, advanced support for semantic and scalable search and discovery across distributed scientific databases, integrated (cross-domain) data platforms. Distributed, efficient, and resilient data-mining support.
2016-17	Operational interoperability related to heterogeneous data-oriented environments, production level data services, social collaborative virtual environments, and distributed knowledge-based systems.
2018-19	Full data integration and interoperability among heterogeneous environment (data centers, data grids, clouds environments). Cross-domain, real-time, and interactive data and knowledge discovery, access, processing, mining, analysis, and visualization.

4.1.5.4 Cross-Cutting Considerations

Four cross-cutting considerations have been identified.

Performance: Efficient access to external environments is crucial; especially if this step is part of complex workflows that start/run inside the data centers and exploit external data sources to enrich their processing and analysis. To have data grids or clouds as part of the system, high-performance network connections are strongly needed, as well as high-performance data transfer protocols.

Resilience: External environments relates to distributed environments (i.e., data grids) are characterized by many software (i.e., services) and hardware (i.e., routers, switches, storages) components. Consequently there could be transient and permanent errors and issues everywhere in the global scenario to be addressed at runtime. Making hardware and software components resilient is a strong challenge for external data environments.

Scalability: At such large scale the number of potential users and actors in this milieu, as well as the number of data collections, will be high. This situation implies the need to have a scalable architecture able to deal with a growing community and an increasing volume of data, without decreasing the level of quality of service and efficiency.

Programmability: The applications developers cannot be expected to manage, at a low level, distribution, replication, load balancing, and other issues explicitly in their codes. Complex aspects of distributed services need to be available as high-level APIs to allow end users to optimize their code, perform tuning operations, and improve their applications.

4.2 Development Environments

The application development environment is the software that the user has to program, debug, and optimize programs. It includes programming models, frameworks, compilers, libraries, debuggers, performance analysis tools, and, at exascale, probably fault tolerance.

4.2.1 Programming Models

4.2.1.1 Technology and Science Drivers for Programming Models

Several challenges have been identified, and possible approaches for addressing these challenges have been suggested.

- Exascale systems are expected to have a *huge number of nodes*. Even within the node, much parallelism will exist in many core architectures and accelerators such as GPGPU. Programming models and languages should support the use of such huge levels of parallelism.
- Exascale systems may consist of *several kinds of components*, including conventional multicore CPUs, many-core chips, and general and application-specific accelerators, resulting in heterogeneity. Programming models and languages should alleviate the programming difficulties arising from such heterogeneity.
- Exascale systems will consist of a *huge number of components*, which will increase the failure rate. Programming models can provide a way to handle such failures with fault resilience mechanisms.
- *Memory bandwidth* will be important in exascale systems. Programming models and languages should provide models to exploit the data locality to make use of complex memory hierarchies.
- The programming model will need to address emerging and on-going applications trends. For example, *algorithms and applications are increasingly adaptive*. Exascale computations will perform massive amounts of I/O; the programming model will need to enable highest levels of I/O performance. New application domains may require new programming models.
- The use of *deep, large software stacks* require the capability to detect and isolate errors at various stages (code development, production, compile time, runtime) and report them at an appropriate level of abstraction.

4.2.1.2 Alternative R&D Strategies for Programming Models

The following strategies are proposed:

- Hybrid vs. uniform: A hybrid programming model is a practical way to program exascale systems that may have architectural heterogeneity. Uniform programming models provide a uniform view of the computation. They reduce the need for the application developer to be aware of the details of the architectural complexity and are often considered to be more productive. Their provision is a challenge, however.

- Evolutionary vs. revolutionary approaches: Specification of incremental improvements to the existing models is a safe approach. Revolutionary approaches may be attractive, but risky.
- Domain specific vs. general programming models: For some application areas, domain-specific models may provide performance and portability with higher productivity than general purpose programming models offer.
- Widely embraced standards vs. single implementations: While the latter have the advantage of rapid development and implementation, the former are based on the experience of a wider community and often required by application groups.

4.2.1.3 Recommended Research Agenda for Programming Models

Research is needed into a variety of promising programming models for exascale computing, including systemwide models that provide a uniform approach to application development across an entire platform, as well as hybrid programming models that combine two or more programming APIs. Such models will need to provide a range of means for the expression of high levels of concurrency and locality and may be capable of supporting application-specific fault tolerance. Enhancements to existing programming interfaces as well as new programming approaches should be explored. For new models, interoperability with existing HPC programming interfaces is highly desirable. Programming models that facilitate productive application development are to be encouraged. Other desirable characteristics are performance transparency and the ability to support incremental application migration.

Timeframe	Targets and Milestones – Programming Models
2010-11	Interoperability between established programming models for HPC (MPI, OpenMP in particular) Initial workshops to discuss potential exascale programming models
2012-13	Fault-tolerant MPI Standard programming model for heterogeneous nodes Systemwide programming model(s) for petascale platforms available
2014-15	Candidate programming models for exascale systems defined
2016-17	Candidate programming models for exascale systems implemented
2018-19	Exascale programming model(s) adopted

4.2.1.4 Cross-Cutting Considerations

Major characteristics of exascale architectures will have a significant impact on the nature of the programming models that are designed to facilitate the creation of exascale-level applications. Hence major departures from the envisaged range of system architectures may necessitate a rethinking of the dominant features of an exascale programming model.

The programming model must facilitate efficient support for massive levels of I/O by applications and must enable the application developer to write fault-aware applications.

The implementation technology will need to be developed to realize the programming models that are defined for exascale computing. The compiler translation will be critical and will need to be of exceptional quality. The runtime system will be expected to provide significant support to the compiler by providing features for managing compute threads, implementing a variety of mechanisms for synchronization, scheduling computations, supporting efforts to balance the workload, executing correctness checks that have been deferred to runtime, collecting performance data, and more.

Applications and libraries will be created using the programming models defined for exascale computing. The programming model will be expected to provide a sufficient range of features to enable the expression of their concurrency and locality and the orchestration of the actions of different threads across the system. The model also must facilitate the composition of different modules and library routines.

A variety of programming-model-aware tools will be required to enable productive application development, translation, and deployment. For instance, tools to support application development might reduce the effort involved in identifying portions of code suitable for execution on certain system components. Tools for debugging will need to be created that are aware of the model's semantics; performance analysis and tuning tools will need to be created that reduce the effort involved in program optimization and are aware of the specific factors that influence program performance under a given programming model. In addition, user annotations may need to be defined to support the actions of the compilers and tools.

4.2.2 Frameworks

4.2.2.1 Technology and Science Drivers for Frameworks

Effective use of exascale systems will place many new demands on application design and implementation. Left alone, each application team will face a daunting collection of infrastructure requirements, independent of the science requirements. Frameworks (when properly developed) have successfully provided a common collection of interfaces, tools, and capabilities that are reusable across a set of related applications. In particular, challenging computer science issues—which are often orthogonal to science issues—can be encapsulated and abstracted in a way that is easy for applications to use, while still maintaining or even improving performance.

A focused effort on frameworks for exascale systems is needed for the following reasons.

- We have a large body of existing scalable applications that we want to migrate toward exascale.
- Many novel exascale-class applications are expected.
- Frameworks provide the best cost and time approach to application development.
- Exascale computing provides a new opportunity for multiscale, multiphysics, and multidisciplinary applications.

4.2.2.2 Alternative R&D Strategies for Frameworks

Two R&D strategies are considered for frameworks.

No frameworks: Most successful frameworks are constructed in response to substantial experience developing individual components, where these components have substantial common requirements, natural interoperability relationships, or both. It is certainly possible to ignore the commonalities and relationships and focus on one-of-a-kind applications. Initially this strategy may appear attractive because it provides the shortest path to single application completion. As more applications are developed, however, this strategy produces redundant, incompatible, and suboptimal software that is difficult to maintain and upgrade, ultimately limiting the number of exascale applications, their quality, and their ability to be improved over their lifetime.

Clean-slate frameworks: If exascale systems eventually require a completely new programming model, the approach we will use to establish exascale frameworks will differ from the case where existing applications are refactored. In this case, the framework will be best constructed to solve a minimally interesting problem. Then existing applications will be mined for their useful software fragments. This strategy was required for many applications when making the transition from vector multiprocessors to MPI.

4.2.2.3 Recommended Research Agenda for Frameworks

Successful development of exascale-class frameworks will require a decade of effort. Among the critical research topics that must be addressed to achieve this goal are the following:

- Identification and development of cross-cutting algorithm and software technologies: For the existing scalable application base and for new applications, there will be common requirements for moving to exascale systems. For example, partitioning and load-balancing algorithms for exascale systems and usage of many-core libraries are common needs.
- Refactoring for many-core: In anticipation of many-core programming model decisions, we must still make progress in preparing for exascale systems by understanding the common requirements of many-core programming that will be true regardless of the final choice in programming model.

The table below, which gives the initial timeline for major activities and deliverables, focuses on the following elements:

Workshops: The computational science and engineering communities have many existing frameworks, some multi-institutional but most centered at a single institution. As a result, the practices, tools, and capabilities of each framework vary greatly, as does the scope of visibility outside the host institution. The first priority for successful exascale framework development must be a series of workshops. The first workshop will bring together people from existing framework efforts, developers of enabling technologies (programming models, algorithms and libraries), and application stakeholders who must ultimately use and develop within the proposed frameworks to perform analyses of capabilities and gaps. Subsequent workshops will focus on specific R&D issues necessary for success.

Breadth-first frameworks: The next major effort will be the development of two to three frameworks: one for libraries and one or two specific application domains. Although programming models, libraries, and fault-resilient capabilities will probably not be mature, this initial breadth-first approach will facilitate co-design of the framework with these enabling tools to ensure compatibility. This effort will also focus on mining capabilities from existing applications as appropriate as well as provide a first definition of the common tool-chain.

Full-scope, additional frameworks: In subsequent years, the programming model, libraries, and fault-resilient strategies should mature, allowing the initial frameworks to solidify these aspects of the design and implementation. Shortly after, or perhaps concurrently, several new domain-specific frameworks can begin, utilizing the design decisions and tool-chain established by the first frameworks.

Deployment: In the first years of exascale capabilities, all frameworks should be in a state to demonstrate exascale capabilities on the first available exascale-class systems.

Timeframe	Targets and Milestones —- Frameworks
2010-11	Workshops: 2010, 2011, regularly after. • Bring together members from key existing framework efforts, algorithm/library developers, programming models. • Workshop 1: – Capabilities/gaps analysis. – First opportunities for multi-institutional frameworks. – Best practices from existing efforts. – Common tool chain requirements. – Possible win-win scenarios.

	• Workshop 2: – Plan for programming model evaluations. – Development of library data model semantics. • Workshop 3: – Applications-driven resilience models.
2012-13	Develop first two applications and first library frameworks, 2013. • Mining of components from existing capabilities. – Implementation of common tool chain, programming model, first resilience harness, library interfaces. • Breadth-first approach.
2014-15	Full development of exascale-specific framework features: – Mature framework-library data layout semantics. – Fully capable fault resilience capabilities. – Fully defined common toolchain.
2016-17	Development of two to three additional appllication frameworks, 2017. • Leveragiing of infrastructure/design knowledge from first efforts. • Development of intercomponent coupling capabilities (e.g., data sharing).
2018-19	Demonstration of full-scale application capabilities across all frameworks on exascale system, 2019.

4.2.2.4 Cross-Cutting Considerations

Framework efforts will be greatly affected by evolving programming models, libraries, and new algorithm development, as well as fault-resilient requirements and capabilities. Although MPI will likely be part of the picture, with a node programming model underneath, a radical new programming and execution model maybe needed. In all cases, a framework will be important for rapidly deploying a critical mass of application capabilities.

Ultimately, any frameworks we develop must have buy-in from application development teams, those domain scientists who are encoding the physics and engineering models. Without their full support, our frameworks will be irrelevant. Computational domain scientists must be part of the framework development process as needed to obtain this support.

Frameworks and the libraries they provide must be part of the software stack for petascale, trans-petascale, and exascale systems. This approach is essential for providing application developers with a common software environment at several scales of computing.

4.2.3 Compilers

4.2.3.1 Technology and Science Drivers for Compilers

Compilers will be a critical component of exascale software solutions. Not only will they be required to implement new and enhanced programming models and to generate object code with exceptional quality, but they will also need to support the process of program adaptation, tuning, and debugging. The high number of potentially simpler (in-order) cores and the existence of specialized components will increase the importance of the compiler.

Compilers for uniform programming models that span entire systems will need to manage the distribution of data, locality of computation, and orchestration of communication and computation in such a manner that all components of the machine perform useful computations. With substantial support from the runtime library, they may also be required to balance the workload across the system components.

Compilers for node programming models may be required to generate code that runs across a large collection of general-purpose cores or across a node that may be configured with general-purpose cores along with one or more specialized accelerators.

Memory hierarchies will be highly complex; memory will be distributed across the nodes of exascale systems and will be NUMA within the individual nodes, with many levels of cache and possibly scratchpad memory. Compilers will be expected to generate code that exhibits high levels of locality in order to minimize the cost of memory accesses. Compilers also] may need to explicitly manage the transfer of data between different subcomponents within nodes.

4.2.3.2 Alternative R&D Strategies for Compilers

The alternative R&D strategies described for programming models apply equally to compilers, since they provide a major part of the implementation of the programming models. By ensuring interoperability between different languages and programming models, compilers can be key to mitigating the risk involved in selecting an emerging programming model and may increase the adoption of new models by offering an incremental path from existing or proposed models (e.g., MPI, OpenMP, UPC, X10, Chapel).

4.2.3.3 Recommended Research Agenda for Compilers

Advances in compiler technology are key to the provision of programming models that offer both performance and productivity characteristics. The following topics should be pursued:

- Techniques for the translation of new exascale programming models and languages supporting high productivity and performance, hybrid programming models, and programming models that span heterogeneous systems.
- Powerful optimization frameworks. Implementing parallel program analyses and new, architecture-aware optimizations, including power, will be key to the efficient translation of exascale programs. Improved strategies for automatic parallelization are needed, as are techniques for determining regions of code that may be suitable for specific hardware components.
- Experimentation with new optimizations and online feedback-based optimizations, benefiting from recent experiences with just-in-time compilation. Other topics include generation of multiple code versions; more aggressive, speculative optimizations; and incorporation of lightweight strategies for modifying code on the fly.
- Support of strategies for enabling fault tolerance. For example, compilers may be able to help reduce the amount of data involved in checkpointing.
- Standard interfaces facilitating interactions between the compiler and the development and execution environment. Such interfaces could enable tools or application developers to drive the translation process in new ways and enable the compiler to drive the actions of tools during runtime, for example to gather specific kinds of performance data. Compilers should be capable of automatically instrumenting code.
- Compiler-based tools for application development. Such tools could support the application development process, help interpret the impact of the compiler's translation on the application's runtime behavior, and explain how the application developer might improve the results of this translation.
- Innovative techniques. Compilers may be able to benefit from autotuning, may incorporate methods for learning from prior experiences, may exploit knowledge of suitable optimization strategies that is gained from the development and execution environments, and may apply novel techniques that complement traditional translation strategies.

Timeframe	Targets and Milestones -- Compilers
2010-11	MPI-aware compilers supporting MPI implementations. Initial interface specified to enable compilers to interact with performance and runtime correctness-checking tools.
2012-13	Compiler support for hybrid programming models
2014-15	Standard heterogeneous programming model implemented Systemwide high-level programming model implemented
2016-17	Exascale programming model implemented Standard interfaces for interactions between compilers and other tools in development and execution environment
2018-19	Refinement of architecture awareness Compilers that interact smoothly with performance and runtime tools

4.2.3.4 Cross-Cutting Considerations

Compilers must no longer be viewed as a black box but rather as open translation infrastructures that must be capable of interoperating with all elements of the development and execution environment, especially the runtime system and tools.

The runtime system will be expected to provide significant support to the compiler by providing a number of features for managing compute threads, implementing a variety of mechanisms for synchronization, scheduling tasks and other computations, and supporting efforts to balance the workload.

Compilers need to generate efficient code for the target architecture. Therefore they need to be developed in an architecture-aware manner. The use of explicit cost models may simplify the generation of code for different hardware configurations.

4.2.4 Numerical Libraries

4.2.4.1 Technology and Science Drivers for Libraries

Numerical libraries underpin any science application developed for high-performance computing and offer the potential to exploit the underlying computer systems without the application developer necessarily understanding the architectural details. Hence, science drivers are more or less automatically built in. However, we may expect new applications to emerge with exascale systems, and libraries should adapt accordingly.

The technology drivers for library development include hybrid architectures, programming models, accuracy, fault detection, energy budget, memory hierarchy and the relevant standards. Numerical libraries depend on the formation of various standards that will be needed to ensure the widespread deployment of the software components. The libraries will be equally dependent on the operating system and the computer architecture features and how they are communicated to the library level.

4.2.4.2 Alternative R&D Strategies for Libraries

The alternative research and development strategies for libraries will be driven by the operating system and software environment provided on given architectures. We can assume that we will see models such as message-passing libraries, global address space languages, and message-driven work queues. Since all three models likely will occur at some level in future systems, matching implementations need to be developed concurrently. In particular, the three programming models should be interoperable to permit the widest deployment.

4.2.4.3 Recommended Research Agenda for Libraries

Existing numerical libraries will need to be rewritten and extended in light of the emerging architectural changes. The technology drivers will necessitate the redesign of the existing libraries and will force re-engineering and implementation of new algorithms. Because of the enhanced levels of concurrency on future systems, algorithms will need to embrace asynchrony to generate the number of required independent operations.

The research agenda will need to include the following:

1. Hybrid and hierarchical based software: efficient implementations need to be aware of the underlying platform and memory hierarchy for optimal deployment.
2. Autotuning: Libraries need to have the ability to adapt to the possibly heterogeneous environment in which they have to operate.
3. Fault-oblivious and error-tolerant implementations: The libraries need to be resilient with regard to the increased rate of faults in the data being processed.
4. Mixed arithmetic for performance and energy saving: The libraries must be able to find optimal mapping of required precision in terms of speed, precision, and energy usage.
5. Architectural-aware algorithms that adapt to the underlying architectural characteristics: The libraries must be able to act on given architectural information to select or generate optimal instantiations of library routines.
6. Energy-efficient implementations to optimize the energy envelope for a given implementation: The libraries should have the ability to take the total power usage into account and optimize for this parameter.
7. Algorithms for minimizing communications: Such algorithms are essential because communications play such an important role in performance and scalability.
8. Algorithms for shared-memory architectures: These algorithms have long been a staple, but they will have a prominent role on future exascale systems as a way to mitigate the impact of increased iteration counts in Schwarz-type algorithms.
9. Fusion of library routine implementations: Libraries often introduce artificial separations into the code, based on the function of each routine. Techniques that permit the fusion of such routines (e.g., of the loops in two consecutive library calls) will be needed.

Timeframe	Targets and Milestones – Numerical Libraries
2010-12	Standards for hybrid (heterogeneous) computing are needed immediately. 2011: Milestone: Heterogeneous software libraries 2012: Milestone: Language issues addressed
2012-14	Standards required for architectural characteristics. 2013: Milestone: Architectural transparency
2014-16	2015: Milestone: Self-adapting for performance Standards required for energy awareness
2016-17	2016: Milestone: Energy awareness Standard for fault tolerance required
2018-19	2018 Milestone: Fault tolerance 2019: Milestone: Scaling to billion way

4.2.4.4 Cross-Cutting Considerations

Libraries will require standards to build on. These will include standards for power management, architectural characteristics, programming for heterogeneous environments and fault tolerance. Establishing such standards presupposes that the information regarding the underlying architecture, energy usage, and so forth, will be available as parameters to be used within the library implementations.

The libraries need to provide language bindings for existing as well as newly emerging languages. At the same time, the calling sequences for their routines should fit in with the various programming models available for exascale environments.

4.2.5 Debugging

4.2.5.1 Technology Drivers for Debugging

Historically debugging has meant the process by which errors in program code are discovered and addressed. The scale of modern parallel computers has pushed the boundaries of that definition in two ways. Massive concurrency at terascale and petascale has led to profound challenges in the ability of a software debugger to encompass the entire parallel application consisting of thousands of processes. Additionally, it has initiated the need to debug not just the code but machine and OS environments where bugs and contention outside the program code itself may be the underlying cause of faults seen at the application layer.

With exascale computing we formally broaden the scope of debugging to including finding problems in the execution of program code by identifying and addressing application incorrectness as well as application failure and critical application performance bottlenecks that may be either reproducible or transient. These faults and bottlenecks may have their origins in the code itself or may be consequences of hardware or software conditions outside the control of the application. As an example and evident already at the petascale, a failed switch adapter on a remote node may cause failures in other jobs or may bring communication to a near standstill. For bulk synchronous parallel codes it normally takes only one slow task to limit the overall performance of the code.

The following aspects of exascale technology will drive decisions in debugging:

- Concurrency-driven overhead in debugging
- Scalability of debugger methodologies (data and interfaces)
- Concurrency scaling of the frequency of external errors/failures
- Heterogeneity and lightweight operating systems

These technology drivers are specific instances of the more broadly stated technology trends in exascale of concurrency, resiliency, and heterogeneity within a node. If ignored, debugging at exascale will become more and more costly, increasing the human effort applied to debugging and diminishing the investment in HPC resources by requiring more machine hours to be devoted to costly debug sessions. The research strategy for exascale debugging therefore must aim to streamline the debugging process by making it more scalable and more reliable.

4.2.5.2 Alternative R&D Strategies for Debugging

Exascale is a regime in which the rate of hardware faults will make debugging, in the expanded context mentioned above, a persistently needed real-time activity. We therefore suggest a strategy that “plans to debug” at compile time and also addresses the data management problems presented by dramatically higher concurrencies. The utility in debugging in a separate session will be limited since a large class of bugs may not be reproducible. Exascale will require the ability to “debug without stopping.” Scalability in debugging has been addressed in previous generations of HPC systems. Research to advance the state of the art in scalability will be required.

Instead of pursuing the development of debuggers as monolithic applications capable of running other user applications in a debug environment, we propose research and development to improve the information sources from which a variety of debugging frameworks can benefit. This strategy borrows a lesson learned in the performance tools community, which has largely moved away from each tool having its own means of deriving machine function (reading counters, registers, etc.) toward development of robust APIs that deliver that information in a portable manner. For example, PAPI provides a common interface for performance information upon which performance tools may be built.

To build such scalable and reliable sources of information for debugging, we suggest vertical integration with compiler, library, runtime, OS, and I/O layers. This integration achieves two important goals at the same time. First, it expands the perspective into the application from multiple directions by providing multiple layers or contexts in which to debug. Specific aspects of codes such as just communication, I/O, specific libraries, or even user-defined quantities or data structures will allow the debugging process to zero in on the anomaly or fault in question. Composition of these data sources will allow for cross-checking and hypothesis testing as to the origin of a fault or bottleneck. This contrasts with the idea of using a debugger to step through executing code on an instruction or subroutines basis and moves in the direction of having the debugging framework become advisory and participatory in the production and execution of codes.

Second, vertical integration that delivers portable standards for gathering and acting on debug information provides efficiency in the design and maintenance of debugging tools. Instead of developing an end-to-end solution within each debugger, we imagine a lowered barrier to entry to the design of special purpose, custom-fitted debuggers that draw on reliable, scalable, and portable mechanisms for monitoring and controlling application codes. Moving from a one-size-fits-all perspective on debugging to modularly selectable approaches will enhance the ability for applications to incorporate the handling of faults and problem scenarios internally. Currently, a large mismatch exists between what the layers underlying the application tell the application about faults and what the application needs to know.

4.2.5.3 Recommended Research Agenda for Debugging

Debugging technology needs to grow away from monolithic applications toward runtime libraries and layers that detect problems and aggregate highly concurrent debugging information into a categorical rather than task based context. Pursuing this path raises a variety of research challenges whose solution will be critical to finding a successful approach to debugging at exascale:

- Methods for scalable clustering of application process/thread states – Many millions of synopses can be made understandable by clustering into types or categories. Debuggers will need to have the ability to search through this volume of data to find the "needle in the haystack" in order to speed root cause determination.
- Debugging without stopping (resilient analysis of victim processes) – Support for debugging will be needed in cases where one node has died and OS and runtime methods are able to migrate and/or reschedule failed tasks, keeping the application alive. Debuggers will need interoperability with system and runtime fault tolerance technologies.
- Vertical integration of debug and performance information across software layers – It will be necessary to find ways to move debugging into multiple levels of application development, build, and execution in order to get a fuller picture of application problems. Consistent standards in the design of these interfaces will be needed to make debuggers and tools more portable as well as easer to develop and maintain.
- Layered contexts or modes of debugging – Instead of a one-size-fits-all approach, developers will need to be able to select custom levels of debug in order to connect the dots between potential bugs and their causes. "All the data all the time" will not be an option for full-scale exascale debugging. Intelligent selection from a menu of reliable data sources will have to be able to target the specifics of a potential bug.
- Automatically triggered debugging – Instead of debugging test cases in a separate session, some exascale debugging must be delivered as problems unfold. Users will have to be able to advise the application about objectives from which deviation is considered a bug. A debugging framework with these capabilities would enable applications to advise the user about problem indicators, for example, expanding memory footprint, incorrectness, and sudden changes in performance.

By focusing on the ability of debugging frameworks to scale and communicate well, this agenda will lower the barriers to debugging, lower the human and machine costs of debugging, and enhance the trust in the reliability of scientific output from exascale systems.

Timeframe	Targets and Milestones – Debugging Tools
2010-11	Planning and workshops Lightweight debugging at 1e5 cores
2012-13	Support for heterogeneity in nodes
2014-15	Simulation at 10^6 cores
2016-17	Software development to support 1e6 core production debugging
2018-19	Near-production exascale

4.3 Applications

While IESP may not focus on developing applications per se, they are nevertheless the reason for the existence of such systems. It may be that exascale systems are specialized machines, co-designed with specific families of applications in mind. Therefore, IESP needs to invest in the technology that makes these applications feasible.

4.3.1 Application Element: Algorithms

4.3.1.1 Technology and Science Drivers for Algorithms

Algorithms must be developed to deal with the architectural realities in an exascale system. In addition, algorithmic innovation can provide efficient alternatives to computer hardware, addressing issues such as reliability and power.

Scalability is perhaps the most obvious driver for algorithms. Contributing to scalability are problems in concurrency, latency, and load balancing. Because an exascale system will have 10^8 to 10^9 threads, simply creating enough concurrency from an application can become a challenge (a 1000^3 mesh has one point per thread on such a system; the low computation/communication ratio of such a problem is typically inefficient). Even current systems have a 10^3–10^4cycle hardware latency in accessing remote memory. Hiding this latency requires algorithms that achieve a computation/communication overlap of at least 10^4 cycles; exascale systems are likely to require a similar degree of latency hiding (because the ratio of processor and memory speeds are expected to remain about the same). Many current algorithms have synchronization points (such as dot products/allreduce) that limit opportunities for latency hiding (e.g., Krylov methods for solving sparse linear systems). These synchronization points must be eliminated. Moreover, static load balancing rarely provides an exact load balance; experience with current terascale and near-petascale systems suggests that this is already a major scalability problem for many algorithms.

Fault tolerance and fault resilience are also drivers for algorithms. While hardware and system software solutions to managing faults are possible, it may be more efficient for the algorithm to contribute to solving the fault resilience problem. Experience shows applications may not detect faults (which may also be missed by the hardware); we need to evaluate the role of algorithms in detecting faults. Detecting faults in hardware requires additional power and memory. Regardless of which component detects a fault, must be repaired. The current general-purpose solutions (e.g., checkpoint/restart) are already demanding on high-end platforms (e.g., requiring significant I/O bandwidth). We need to evaluate the role of algorithms in repairing faults, particularly transient (e.g., memory upset) faults. In addition, one can imagine a new class of algorithms that are inherently fault-tolerant, such as those that converge stochastically. The advantage of robustness on exascale platforms will eventually override concerns over computational efficiency.

Because of the likely complexity of an exascale system, algorithms must be developed that are a good match to the available hardware. One of the most challenging demands is power; algorithms that minimize power use need to be developed. This will require performance models that include energy. Note that this may be combined with other constraints, since data motion consumes energy. As many proposals for exascale systems (and power-efficient petascale systems) exploit heterogeneous processors, algorithms will need to be developed that can make use of these processor structures. The current experience with GPGPU systems, while promising for some algorithms, has not shown benefits with other algorithms. Heterogeneous systems also require different strategies for use of memory and functional units. For example, on some hardware it may be advantageous for algorithms to exploit multiple levels of precision. Exascale systems are likely to have orders of magnitude less memory per core than current systems (though still large amounts of memory). Power constraints may reduce the amount of fast memory available, adding to the need for latency hiding. Thus we need algorithms that use memory more efficiently, for example, more accuracy per byte, fewer data moves per result. The choice of algorithm for a particular application may depend sensitively on details of the memory hierarchy and implementation; portability between diverse architectures will require algorithms that can automatically adjust to local hardware constraints.

The final driver is this need to re-examine the classes of applications that are suitable for exascale computing. Because exascale systems are likely to be different from simple extrapolations of petascale systems, some application areas may become suitable again; others (because of the extreme scale and degree of concurrency) may become possible for the first time.

A major concern is that an exascale system may be very different from current systems and will require new approaches.

4.3.1.2 Alternative R&D Strategies for Algorithms

All strategies for developing algorithms for exascale systems must start with several “strawman exascale architectures” that are described in enough detail to permit the evaluation of the suitability of current algorithms on potential exascale systems. There are then two basic strategies: (1) refine existing algorithms to expose more concurrency, adapt to heterogeneous architectures, and manage faults and (2) develop new algorithms.

In refining algorithms, a number of strategies may be applied. Developing new algorithms requires rethinking the entire application approach, starting with the choice of mathematical model and approximation methods used. It is also important to re-evaluate existing methods, such as the use of Monte Carlo; reconsider tradeoffs between implicit and explicit methods; and replace FFT with other approaches that can avoid the all-to-all communication. In creating algorithms that are fault tolerant, a key approach is to use or create redundant information in the algorithm or mathematical model. To make effective use of likely exascale hardware, methods that make more efficient use of memory, such as higher-order methods, as well as the development of more predictive analytic performance models, will be key.

4.3.1.3 Recommended Research Agenda for Algorithms

A research agenda is shown in the table below, along with comments providing more detail about each in the enumerated list below. Not captured in this table is the need to follow two broad strategies: an evolutionary one that updates current algorithms for exascale (following the approaches that have successfully been followed to take us to petascale) and one that invests in higher risk but higher payoff development of new algorithms. In either case, it is important to develop performance models (and thus strawman exascale architecture designs) against which algorithm developments can be evaluated. In addition, it is all too easy for applications to define algorithm “requirements” that overly constrain the possible solutions. It is important to re-evaluate application needs, for example, evaluating changes to the model or approximation to allow the use of exascale-appropriate algorithms.

Against this background, the critical research challenges that need to be addressed for application algorithms that build on the X-stack are as follows:

- Gap analysis – need to perform a detailed analysis of the applications, particularly with respect to quantitative models of performance and scalability
- Scalability, particularly relaxing synchronization constraints
- Fault tolerance and resilience, including fault detection and recovery
- Heterogeneous systems – algorithms that are suitable for systems made of functional units with very different abilities

Timeframe	Targets and Milestones – Algorithms
2010-11	Gap analysis. Needs to be completed early to guide the rest of the effort. Evaluation of algorithms needed for applications. Needs to be initiated early and completed early to guide allocation of effort and to identify areas where apps need to rethink the approach (cross-cutting issue). Needs to develop and use more realistic models of computation (quantify need).
2012-13	Algorithms for intranode scaling Algorithms for internode scaling Evaluation on petascale systems Better scaling in node count and within nodes can be performed using petascale systems in this Timeframe (so it makes sense to deliver a first pass in this Timeframe).
2014-15	Prototype algorithms for heterogeneous systems Heterogeneous systems are available now but require both programming model and algorithmic innovation; while some work has already been done, others may require more time. This can be viewed as "a significant fraction of algorithms required for applications expected to run at exascale have effective algorithms for heterogeneous processor systems."
2016-17	Fault resilience Fault resilience is a hard problem; this assumes that work starts now but will take this long to meet the same definition as for heterogeneous systems – "a significant fraction of algorithms have fault resilience."
2018-19	Efficient realizations of algorithms on exascale architectures Efficient implementation includes the realization in exascale programming models and tuning for real systems, which may involve algorithm modifications (since the real architecture will most likely be different from the models used in earlier developments). In addition, the choice of data structures may also change, depending on the abilities of compilers and runtimes to provide efficient execution of the algorithms.

4.3.1.4 Cross-Cutting Considerations

The ability to design and implement efficient and novel algorithms for exascale architectures will be closely tied to improvements in many cross-cutting areas. Examples include the following:

The development of libraries that recognize and exploit the presence of mixed precision mathematics will spur the creation of algorithms that effectively utilize heterogeneous hardware. Ideally, the user could specify the required precision for the result, and the algorithm would choose the best combination of precision on the local hardware in order to achieve it. The actual mechanics would be hidden from the user.

The creation of debugging tools that expose cache use, load imbalance, or local power utilization will be critical for the implementation of self-optimizing algorithms in each of these areas. Currently available methods of debugging large-scale codes to catch, for example, load-balancing issues are manpower intensive and represent a significant barrier to the development of efficient algorithms.

Runtime systems that make available to the running code information about MTBF on the hardware can allow for auto-adjustment of defensive restart strategies. The I/O strategy for even a petascale simulation must be carefully optimized to avoid wasting both compute and storage resources. The situation will only be more critical at exascale.

Tuning of algorithms for performance optimization will benefit from compilers and programming languages that can recognize and utilize multiple levels of parallelism present in the hardware. Current strategies for optimization on HPC architectures result in either one-off, hand-tuned codes or portable and inefficient codes, since it is difficult to express multiple possible levels of parallelism into the structure of the code. The increased portability allowed by some measure of autotuning will maximize the ROI on code development and thus lower the effective cost of entry into HPC.

4.3.2 Application Support: Data Analysis and Visualization

4.3.2.1 Technology and Science Drivers for Data Analysis and Visualization

Modern scientific instruments—for example, in synchrotron science, high energy physics, astronomy, and biotechnology—are all experiencing exponential growth in data generation rates through a combination of improved sensors, increases in scale, widespread availability, and rapid advances in the supporting information technology. Model simulations—for example, in climate, CFD, materials science, and biological science—are also producing vast amounts of data as they scale with the exponential growth in HPC performance. Experimental science, modeling, and simulation are routinely generating petabyte-scale data sets. Exabyte-scale data sets are now part of the planning process for major scientific projects.

The increasing scale and complexity of simulations and the data they produce will be a key driver of the research agenda in the area of data analysis and visualization. These will force new approaches to coupling analysis and visualization computations to the larger datasets. Considerations of dataset size will also drive innovations in analysis techniques, allowing for the advancement of current technology and requiring the research and development of new solutions. Analysis and visualization will be limiting factors in gaining insight from exascale data.

Interactive data exploration will also become increasingly important as dataset scale and complexity continue to grow; however, it will become increasingly difficult to work interactively with these datasets, thus requiring new methods and technologies. These solutions will need to supply the scientist with salient reductions of the raw data and new methods for information and process tracking.

4.3.2.2 Alternative R&D Strategies for Data Analysis and Visualization

Several strategies for enabling data analysis and visualization at exascale are available to us. One strategy would be to continue to incrementally improve and adapt existing technologies (visualization and analysis algorithms, data management schemes, end-to-end resource allocation). This adiabatic expansion of current efforts is well traveled and has a lower barrier to entry than others, but it may not provide adequate solutions in the long run.

Inevitably, some combination of existing technologies and the integration of the four approaches described next will serve important roles in the necessary R&D enterprise.

- **New algorithms** – It would make sense to pursue development of entirely new algorithms that fit well with new large and complex architectures. This approach will be increasingly difficult, owing to the need to explicitly account for larger pools of heterogeneous resources.
- **New data analysis approaches** – New mathematical and statistical approaches must be identified for analysis of exabyte data sets.
- **Integrated adaptive techniques** – Development of integrated adaptive techniques will enable on the fly and learned pattern performance optimization from fine to coarse grain. This strategy would provide a range of means to extract meaningful performance improvements implicitly, rather than by explicit modeling of increasingly complex systems.

- **Pro-active software methods** – Another strategy is to expand the role of supporting visualization environments to include more pro-active software: model- and goal-aware agents, estimated and fuzzy results, and advanced feature identification. This strategy will require abdicating some responsibility to autonomous system software in order to more rapidly sift through large amounts of data in search of hidden elements of discovery and understanding.
- **Metatools** – With a focus on mitigating the increasing burden of high-level organization of the exploration and discovery process, it would be advantageous to invest in methods and tools for keeping track of the processes and products of exploration and discovery. These will include aids to process navigation, hypothesis tracking, workflows, provenance tracking, and advanced collaboration and sharing tools.
- **Collaboration** – Deployment of a global system of large-scale, high-resolution (100 Mpixel) visualization and data analysis systems based on open-source architecture will link universities and research laboratories and facilitate collaborations.

4.3.2.3 Recommended Research Agenda for Data Analysis and Visualization

Many of the innovations required to cope with exascale data analysis and visualization tasks will require considerable development and integration in order to become useful. At the same time, most would be of considerable utility at the petascale. Consequently, it is not only required but could provide up-front benefits to aggressively develop the proposed methods so that they can be deployed early, at least in prototype form, for extensive use in research situations and rigorously evaluated by the application community.

Among the research topics that will prove critical in achieving this goal are the following:

- Identification of features of interest in exabytes of data
- Visualization of streams of exabytes of data from scientific instruments
- Integrating simulation, analysis, and visualization at exascale

Ongoing activities supporting adiabatic expansion of existing techniques onto new hardware architectures and R&D of new algorithms will continue throughout the time span. The major milestones and timetable reflected in the following table would be supported by development of many of the ideas at smaller scale, beginning as soon as possible.

Timeframe	Targets and Milestones – Data Analysis and Visualization
2010-11	Planning and workshops • Assess current tools and technologies • Perform needs and priority analysis across multiple disciplines • Identify common components • Identify new mathematical and statistical research needed for analysis of exabyte data sets • Integrate analysis and visualization into scientific workflows • Develop exascale data analysis and visualization architecture document • Commence initial set of projects for common components and domain specific data analysis and visualization libraries • Plan deployment of global system of large-scale, high-resolution (100 Mpixel) visualization and data analysis systems to link universities and research laboratories
2012-13	Develop 1.0 common component data analysis and visualization libraries Develop 1.0 priority domain-specific data analysis and visualisation libraries Begin deployment of global system of large-scale, high-resolution (100 Mpixel) visualization and data analysis systems Achieve data analysis and visualization at 10^5 cores with petabyte data sets Provide support for heterogeneity in nodes
2014-15	Integrate data analysis and visualization tools in domain-specific workflows Achieve data analysis & visualization at 10^6 cores with 10–100 petabyte data sets
2016-17	Complete 2.0 domain specific data analysis and visualization libraries and workflows Complete 2.0 common component data analysis and visualisation libraries Achieve data analysis and visualization at 10^6 cores with near-exascale data sets
2018-19	Roll out data analysis and visualization at exascale

4.3.2.4 Cross-Cutting Considerations

Architecture at coarse and fine grain. Analysis and visualization can use any or all of the computational, storage, and network resources in a computational environment. Methods developed to address the driving technology and science issues are likely to intersect with design and implementation of future architectures at all granularities, from wide-area considerations to heterogeneity of available processing elements. Also, compiler and debugging tools appropriate for software development on exascale systems will need to be developed to meet the needs of the timetable outlined above.

Opportunistic methods. Many emerging approaches to analysis and visualization leverage opportunities that arise from data locality (e.g., in situ methods), synergies of happenstance (as in analysis embedded in I/O libraries and data movers), and unused capacity (e.g., background analysis embedded in I/O servers). These will each require coordination with fine-grained execution of the numerical algorithms used in the simulation, ongoing read/write operations, and system-level resource scheduling. Resaerchers should consider using exascale performance to rapidly perform model simulations, with data analysis and visualization integrated into the simulation to avoid storing vast amounts of data for later analysis and visualization. This strategy would affect the development of domain-specific simulation codes.

End-to-end or global optimizations. Improvements in understanding algorithms for large-scale heterogeneous architectures and the related advances in runtime and compiler technologies are likely to afford new opportunities for performance optimization of the combined simulation and analysis computations. These and other benefits may accrue from taking a more holistic view of the end-to-end scientific discovery pipeline. Integrating data analysis and visualization into domain-specific exascale scientific workflows will be essential to maximizing the productivity of researchers working on exascale systems.

4.3.3 Application Support: Scientific Data Management

4.3.3.1 Technology and Science Drivers for Scientific Data Management

Management, analysis, and mining of large data sets already present challenging problems, but these activities are critical in petascale systems and will be even more so for exascale systems. Most science applications at this scale will be extremely data intensive: individual simulations are expected to produce petabytes of data; and when combined with multiple executions, the data could approach exabyte scales. Thus, managing scientific data has been identified by the scientific community as one of the most important emerging needs because of the sheer volume and increasing complexity of data. The potential impact of exascale computing will be measured not just in the power it can provide for simulations but also in the capabilities it provides for managing and making sense of the data produced. Clearly needed is an end-to-end approach that encompasses all stages, from the initial data acquisition to the final analysis of the data. Many common questions arise across various application disciplines. Are data management tools available that can manage data at this scale? Although scalable file systems are important as underlying technologies, they are not suitable as a user-level mechanism for scientific data management. What are the scalable algorithm techniques for statistical analysis and mining of data at this scale? Are there mathematical models? Does the "store now and analyze later" model work at this scale? What are the models and tools for indexing, querying, and searching these massive datasets and for knowledge discovery? What are the tools for workflow management? An emerging model relies ever more on teams working together to organize new data, develop derived data, and produce analyses based on the data, all of which can be shared, searched, and queried. What are the models for such sharing, and what are designs for such databases or data warehouses? Data provenance is another critical issue at this scale. What are scalable data formats, and what are the formats for metadata?

4.3.3.2 Alternative R&D Strategies for Scientific Data Management

Scientific data management covers many subfields, from data formats, workflow tools, and query to data mining and knowledge discovery. For most of the subfields, R&D strategies must simultaneously consider the scalable I/O and storage devices for the required scaling for exascale systems.

Data Analysis and Mining Software and Tools: Knowledge discovery from massive datasets produced or collected will require sophisticated, easy-to-use, yet scalable tools for statistical analysis, data processing, and data mining. Scalable algorithms and software must be developed that can handle multivariate, multidimensional (and large number of dimensions), hierarchical, and multiscale data at massive scales. Scalable tools based on these algorithms must be developed with a capability to incorporate other algorithms. Traditionally, analytics and mining specification languages have been sequential and are unable to scale to massive datasets. Parallel languages for analysis and mining that can scale to massive data sets will be important. Data mining and analysis scalability can also be addressed via the use of accelerators such as GPGPUs and FPGAs; and the development of scalable algorithms, libraries, and tools that can exploit these accelerators will be important. Techniques for on-line analytics, active storage models, and co-processing models should be developed that can run concurrently (potentially on a subsystem) with the simulations and can exploit the multicore nature of the systems. Also, maximizing the use of data while it is available should be investigated.

Scientific Workflow Tools: Scientific workflow is defined as a series of structured activities, computation, data analysis, and knowledge discovery that arise in scientific problem-solving. That is, it is a set of tools and software that allow a scientist to specify end-to-end control and data flow as well as coordination and scheduling of various activities. Designing scalable workflow tools with easy-to-use interfaces will be important for exascale systems, both for performance and for scientific productivity as well as for effective use of these systems. Scaling of workflow tools will entail enhancements of current designs and/or developing new approaches that can effectively use scalable analytics and I/O capabilities and that can incorporate query processing. New design mechanisms, including templates, semantic types, and user histories will simplify workflow design and increase dependability. As a part of workflow tools, the creation, management, querying, and use of data provenance must be investigated.

Extensions of Databases Systems: Commercial database systems such as those based on relational or object models (or derivation thereof) have proved unsuitable for organizing, storing, or querying scientific data at any reasonable scale. Although it is an alternative for pursuing data management solutions, it is not likely to be successful.

Design of New Database Systems: A potential approach to database systems for scientific computing is to investigate completely new approaches that scale in performance, usability, query, data modeling, and an ability to incorporate complex data types in scientific applications and that eliminate the overconstraining usage models, which are impediments to scalability in traditional databases. Scalable file systems will be critical as an underlying software layer, but not as a user-level interface for data management purposes. It is critical to move to dataset-oriented paradigms for data management, in which the file systems serve the data management layer and need to be optimized for limited functionality needed by the data management layer, which in turn presents an intuitive, easy-to-use interface to the user for managing, querying and analyzing data with a capability for the users to embed their functions within the data management systems.

Scalable Data Format and High-Level Libraries: Scientists use different data formats, mainly driven by their ability to specify the multidimensional, multiscale, often sparse, semi-structured, unstructured, and adaptive data. Examples of these formats and corresponding libraries include netCDF and HDF and their corresponding parallel (PnetCDF and PHDF) versions. Changes in these have been driven mainly by backward compatibility. Approaches to adapt and enhance these formats and scale the data access libraries must be investigated. Furthermore, new storage formats that emphasize scalability and the use of effective parallel I/O, along with the capabilities to incorporate analytics and workflow mechanisms, are important areas for research and development. Although the use of new storage devices such as SCM has been discussed in the context of I/O systems, their use in redesigning or optimizing storage of data and metadata for performance and effective querying high-level data formats and libraries should be pursued, especially given that accessing metadata is a major bottleneck.

Search and Query Tools: Effective searching and querying of scientific data are critical. New technology is needed for efficient and scalable searching and filtering of large-scale, scientific multivariate datasets with hundreds of searchable attributes to deliver the most relevant data and results. Users may be interested in querying specific events or the presence or absence of certain data subsets. Furthermore, filtering of data based on certain query specifications is important, including capabilities to combine multiple data sets and query across them.

Wide-Area Data Access, Movement, and Query Tools: Wide-area data access is becoming an increasingly important part of many scientific workflows. In order to most seamlessly interact with wide-area storage systems, tools must be developed that can span various data management techniques across a wide area, integrated with scalable I/O, workflow tools, and query and search techniques.

4.3.3.3 Recommended Research Agenda for Scientific Data Management

The recommended research agenda for scientific data management systems is all items above except item 3.

Timeframe	Targets and Milestones – Scientific Data Management
2010-11	• Extensions and redesign of scalable data formats • Extend capabilities of workflow tools to incorporate analytics • Design of data mining and statistical algorithms for multiscale data
2012-13	• Design and definition of scientific database systems • Workflow tools with fault-resiliency specification capabilities • Integration of scalable I/O techniques with wide-area SDM technologies
2014-15	• Analytics and mining for active storage Systems, including functionality for users to embed their functions. • Scalable implementations of high-level libraries for various high-level data formats • Scalable query and search capabilities in scientific database systems
2016-17	• Comprehensive parallel data mining and analytics suites for scalable clusters with GPGPU and other accelerators • Extensive capabilities for managing data provenance within the workflow and other SDM tools • On-line analytics capability and its integration with workflow tools
2018-19	• Real-time knowledge discovery and insights • Comprehensive scientific data management tools

4.3.3.4 Cross-Cutting Considerations

Scientific data management clearly has cross-cutting considerations with scalable storage and I/O, visualization techniques and tools, operating systems, fault-resiliency mechanisms, the communication layer, and, to some extent, programming models.

4.4 Cross-Cutting Dimensions

4.4.1 Resilience

Since exascale systems are expected to have millions of processors and hundreds of millions of cores, resilience will be necessary for the exascale applications. If the relevant components of the X-stack are not fault tolerant, then even relatively short-lived applications are unlikely to finish, or worse, may terminate with an incorrect result. In other words, insufficient resilience of the software infrastructure would likely render extreme scale systems effectively unusable. The amount of data needing to be checkpointed and the expected rate of faults for petascale and larger systems are already exposing the inadequacies of traditional checkpoint/restart techniques. The trends predict that, for exascale systems, faults will be continuous and across all parts the hardware and software layers, which will require new programming paradigms. Because there is no compromise for resilience, the challenges it presents need to be addressed now for solutions to be ready when exascale systems arrive.

4.4.1.1 Technology Drivers for Resilience

Five technology drivers have been identified.

- Exponential increases in the number of sockets, cores, threads, disks, and memory size are expected.

- Because of the size and complexity, there will be more faults and a large variety of errors (soft errors, silent soft errors, transient and permanent software and hardware errors) everywhere in the system. Some projections consider than the full-system mean time to failure will be in the range of one minute.
- Silent soft errors will become significant and raise the issues of result and end-to-end data correctness.
- New technologies such as Flash memory (SSD), phase-change RAM, and accelerators will raise both new opportunities (stable local storage, faster checkpointing, faster checkpoint compression, etc.) and new problems (capturing the state of accelerators)).
- Intel has estimated that additional correctness checks on chip will increase power consumption 15–20%. The need to significantly reduce the overall power used by exascale systems is likely to reduce the reliability of components and reduce the mean time before failure of the overall system.

4.4.1.2 Gap Analysis

This section briefly identifies the technology gaps that must be overcome in moving from current high-performance computing to the exascale.

- Existing fault tolerance techniques (global checkpoint/global restart) will be impractical at the exascale. New techniques for saving and restoring state need to be developed into practical solutions.
- The most common programming model, MPI, does not offer a paradigm for resilient programming. A failure of a single task often leads to killing the entire application.
- Present applications and system software are neither fault tolerant nor fault aware and are not designed to confine errors/faults, to avoid or limit their propagation, and to recover from them when possible.
- There is no communication or coordination between the layers of the software stack in error/fault detection and management, nor coordination for preventive or corrective actions.
- Errors, fault root causes, and propagation are not well understood.
- There is almost never verification of the results from large, long-running simulations.
- There are no standard metrics, no standardized experimental methodology, and no standard experimental environment to stress resilience solutions and compare them fairly.

4.4.1.3 Alternative R&D Strategies

Resilience can be attacked from different angles:

- Global recovery versus fault confinement and local recovery
- Fault recovery versus fault avoidance (fault prediction + migration)
- Transparent (system managed) versus application directed
- Recovery by rollback versus replication

Since rollback recovery, as we know it today, will be not applicable by 2014–2016, research needs to progress on all techniques that help to avoid global coordination and global rollback.

4.4.1.4 Recommended Research Agenda for Resilience

The recommended research agenda follows two main tracks:

1. Extend the applicability of rollback toward more local recovery – scalable, low overhead, fault tolerant protocols, integration of SSD and PRAM for checkpointing, reducing checkpoint size (new execution state management), error and fault confinement and local recovery, consistent fault management across layers (including application and system software interactions), language support and paradigm for resilience, and dynamic error handling by applications

2. Fault-avoidance and fault-oblivious software to limit the recovery from rollback – improve RAS collection and analysis (root cause); improve understanding of error/fault and their propagation across layers; develop situational awareness, system-level fault prediction for time optimal checkpointing and migration, fault-oblivious system software, and fault-oblivious applications

Timeframe	Targets and Milestones -- Resilience
2010-12	Target 1: Extension of the applicability of rollback recovery Milestones: Design of scalable, low-overhead, fault-tolerant protocols Milestones: Integration of checkpoint size reducing techniques (compiler, incremental, compression, etc.) Milestone: Demonstration of partial-local replication as complement to rollback
2013-15	Target 1: Extension of the applicability of rollback recovery Milestone: Integration of Phase-change RAM technologies Milestone: Implementation of error and fault confinement, local recovery, TMR (cores) Milestone: Development of fault-aware system software Milestone: Provision of language support and paradigm for resilience Milestone: Developoment of application and system software interactions (standard API) Milestone: Consistency across layers (CIFTS or CIFTS like mechanisms) Target 2: Fault-avoidance and fault-oblivious software Milestone: RAS collection and analysis (root cause), situational awareness Milestone: Hardware and software integration
2016-19	Target 2: Fault-avoidance and fault-oblivious software Milestone: System-level fault prediction for time-optimal checkpointing and migration Milestone: Fault-oblivious system software Milestone: Fault-oblivious applications

4.4.2 Power Management

4.4.2.1 Technology Drivers for Power Management

Power has become the leading design constraint for future HPC system designs. In thermally limited designs, power also forces design compromises that lead to highly imbalanced computing systems (such as reduced global system bandwidth). The design compromises required for power-limited logic will reduce system bandwidth and consequently reduce delivered application performance and greatly limit the scope and effectiveness of such systems. From a system management perspective, effective power management systems can substantially reduce overall system power without reducing application performance, and therefore make fielding such systems more practical and cost-effective. The existing power management infrastructure has been derived from consumer electronic devices and fundamentally never had large-scale systems in mind. Without comprehensive cross-cutting technology development for a scalable active power management infrastructure, power consumption will force design compromises that will reduce the scope and feasibility of exascale HPC systems.

From an applications perspective, active power management techniques improve application performance on systems with a limited power budget by dynamically directing power usage only to the portions of the system that require it. For example, a system without power management would melt if it operated memory interfaces at full performance while also operating the floating-point unit at full performance—forcing design compromises that limit the memory bandwidth to 0.01 bytes/flop according to the DARPA projections. In this thermally limited case, however, one can deliver higher memory bandwidth to the application for short periods of time by shifting power away from other components. Whereas the projected bandwidth ratio for a machine would be limited to 0.01 bytes/flop without power management, the delivered bandwidth could be increased to 1 byte/flop for the period of time where the application is bandwidth limited, by shifting the power away from floating point (or other components that are under-utilized in the bandwidth-limited phase of an algorithm). Therefore, power management is an important part of enabling better delivered application performance through dynamic adjustment of system balance to fit within a fixed power budget.

From a system management perspective, power is a leading component of system total-cost-of-ownership. Every megawatt of reduced power consumption translates to a savings of $1M/year on even the least expensive energy contracts. For systems that are projected to consume hundreds of megawatts, power reduction makes fielding of such systems more practical. HPC-focused power management technology can have a much broader impact across the large-scale computing market. High-end servers, which are the building blocks of many HPC systems, were estimated to consume 2% of North American power generation capacity as of 2006, and this factor is growing. By 2013, IDC estimates that HPC systems will be the largest fraction of the high-end server market. Hence, the direct impact of improved power management technology is to reduce the operating cost for exascale HPC systems, but the broader impact is to reduce power consumption of the largest and fastest growing sector of the computing technology market (HPC systems) and reduce carbon emissions for all server technology.

The current state-of-the-art power management systems are based on features developed for the consumer-electronics and laptop markets, which make local control decisions to reduce power. Unfortunately, the technology to collect information across large-scale systems and make control decisions that coordinate power management decisions across the system is not well developed, nor are reduced models of code performance for optimal control. Furthermore, the interfaces for representing sensor data for the control system, describing policies to the control system, and distributing control decisions are not available at scale. Effective systemwide power management will require development of interface standards to enable both vertical (e.g., between local components and integrated system) and horizontal (e.g., between numerical libraries) integration of components. Standardization is also a minimum requirement for broad international collaboration on development of software components. The research and development effort required to bring these technologies into existence will touch on nearly every element of a large-scale computing system design—from library and algorithm design to system management software.

4.4.2.2 Alternative R&D Strategies for Power Management

Fundamentally, power management technology attempts to actively direct power towards useful work. The goal is to reduce system power consumption without a corresponding impact on delivered performance. This is accomplished primarily through two approaches

1. *Power down components when they are underutilized:* Examples include Dynamic Voltage and Frequency Scaling (DVFS), which reduces the clock rate and operating voltage of components when the OS directs it to. Memory systems also support many low-power modes when operating at low loads. Massive Arrays of Redundant Disks (MAID) allow disk arrays to be powered down incrementally (subsets of disks) to reduce power. In the software space, operating systems or libraries use information about the algorithm resource utilization to set power management policy to reduce power.
2. *Explicitly manage data movement:* Both algorithms and hardware subsystems are used to manage data movement to make the most effective use of available bandwidth (and hence power). Examples from the hardware space include solid state disk caches to lower I/O power for frequently accessed data, offloading of work to accelerators, and software-managed memory hierarchies (local stores). Examples from the software space include communication avoiding algorithms, programming models that abstract use of local stores, and libraries that can adapt to current power management states or power management policy.

Current power management features are derived primarily from consumer technology, where the power savings decisions are all made locally. For a large parallel system, locally optimal solutions can be tremendously nonoptimal at the system scale. When nodes go into low-power modes, opportunistically based on local decisions, they create a jitter that can substantially reduce systemscale performance. Therefore, localized automatic power management features are often turned off on production HPC systems. Moreover, the decision to change system balance dynamically to conserve power requires advance notice because there is latency for changing between different power modes. Hence, the control loop for such a capability requires a predictive capability to make optimal control decisions. Therefore, new mechanisms that can coordinate these power savings technologies at system scale will be required to realize an energy-efficiency benefit without a corresponding loss in delivered performance.

A completely adaptive control system requires a method for sensing current resource requirements, making a control decision based on an accurate model for how the system will respond to the control decision, and then distributing that control decision in a coordinated fashion. Currently the control loop for accomplishing this kind of optimal control for power management is fundamentally broken. Predictive models for response to control decisions are generally hand-crafted (a time-consuming process) for the few examples that exist. There is no common expression of policy or objective. There is no comprehensive monitoring or data aggregation. More important, there is almost no tool support for integration of power management into libraries and application codes. Without substantial investments to create systemwide control systems for power management, standards to enable vertical and horizontal integration of these capabilities, and the tools to facilitate easier integration of power management features into application codes, there is little chance that effective power management technologies will emerge. The consequence will be systems that must compromise system balance (and hence delivered application performance) to fit within fixed power constraints, or systems that have impractical power requirements.

4.4.2.3 Recommended Research Agenda for Power Management

The R&D required for the X-stack to enable comprehensive, systemwide power management is pervasive and will touch on a broad variety of system components. The cross-cutting research agenda includes the following elements.

Operating System/Node Scale Resource Management: Operating systems must support quality-of-service management for node-level access to very limited/shared resources. For example, the OS must enable coordinated/fair sharing of the memory interface and network adaptor by hundreds or even thousands of processors on the same node. Support for local and global control decisions requires standardized monitoring interfaces for energy and resource utilization (PAPI for energy counters). Standard control and monitoring interfaces enable adaptable software to handle diversity of hardware features/designs. Future operating systems must also manage new power-efficient architectures; heterogeneous computing resources, including devices such as GPUs, embedded CPUs, and nonvolatile low-power memory and storage; and data movement and locality in memory hierarchy.

Systemscale Resource Management: Power performance monitoring and aggregation are needed that scale to a 1 billion-core system. System management services require standard interfaces to enable coordination across subsystems and international collaboration on component development. Many power management decisions must be executed too rapidly for a software implementation and hence must be expressed as a declarative policy rather than a procedural description of actions. Therefore, policy descriptions must be standardized to do fine-grained management on chip. In particular, standards are required for specifying reduced models of hardware power impact and algorithm performance to make logistical decisions about when and where to move computation as well as the response to adaptations. These include analytical power models of system response and empirical models based on advanced learning theory. Also needed are scalable control algorithms to bridge the gap between global and local models. Systems to aggregate sensor data from across the system (scalable data assimilation and reduction) must make control decisions and distribute those control decisions in a coordinated fashion across large-scale systems hardware. Both online and offline tuning options based on advanced search pruning heuristics should be considered.

Algorithms: We must investigate energy-aware algorithms that base order of complexity on energy cost of operations rather than FLOPs. A good example of this approach is communication-avoiding algorithms, which trade off FLOPS for communication to save energy. Since the optimal trade-off is context specific, however, we must enable libraries to be annotated for a parameterized model of energy to articulate a policy to manage those trade-offs on different system architectures. Standardizing the approach to specifying lightweight models to predict response to resource adjustment will be important to this effort.

Libraries: To create cross-architecture compatible, energy-aware libraries, library designers need to use their domain-specific knowledge of the algorithm to provide power management and policy hints to the power management infrastructure. This research agenda requires that performance/energy efficiency models and power management interfaces in software libraries be standardized. Such standardization will ensure compatibility of the management interfaces and policy coordination across different libraries (horizontal integration) as well as support portability across different machines (vertical integration).

Compilers: Compilers and code generators must be able to automatically instrument code for power management sensors and control interfaces to improve the programmability of such systems. Compiler technology can be augmented to automatically expose "knobs for control" and "sensors" for monitoring of nonlibrary code. A more advanced research topic is to find ways to automatically generate reduced performance and energy consumption models to predict response to resource adaptation.

Applications: Applications require more effective declarative annotations for policy objectives and interfaces to coordinate with advanced power-aware libraries and power management subsystems.

The proposed research agenda targets the following key metrics for improving overall effectiveness of exascale systems.

- **Performance:** Scalable, lightweight, and cross-software hierarchy performance models (analytic models and empirical models) need to be consructed that enable predictive control of application execution, so that we can find ways of reducing power without having a deleterious impact on performance.

- **Programmability**: The applications developers cannot be expected to manage power explicitly due to the overwhelming complexity of the hardware mechanisms. Making power management accessible to application and library architects requires coordinated support from compiler, libraries, and system services.
- **Composability**: There must be standards to enable system components and libraries that are developed by different research groups to work in coordinated fashion with underlying power systems. Standardization of monitoring and control interfaces minimizes the number of incompatible ad-hoc approaches and enables an organized international effort.
- **Scalability**: We must be able to integrate information from the OS, the system-level resource manager, and applications and libraries for a unified strategy to meet objectives.

Timeframe	Targets and Milestones – Power Management
2010-11	Energy onitoring Interface Standards Energy-aware/communication avoiding algorithms ▪ System management ▪ Algorithms ▪ Libraries ▪ Compilers and frameworks ▪ Applications
2012-13	Local OS-managed, Node-Level, Energy Efficiency Adaptation System-level standard interfaces for data collection and dissemination of control requests
2014-15	Compatible Energy-Aware Libraries Using Standardized Interfaces Ability to annotate libraries for parameterized model of energy to articulate a policy to manage trade-offs (different architectures) Standardized approach to expressing lightweight performance models for predictive control (analytic models and empirical models) Scalable algorithms for adaptive control
2016-17	Automated Code Instrumentation (Compilers, Code Generators, Frameworks) Standardized models of hardware power impact and algorithm performance to make logistical decisions (when/where to move computation + response to adaptations)
2018-19	Automated System-Level Adaptation for Energy Efficiency Scale up systems to 1 billion-way parallel adaptive control decision capability

4.4.3 Performance Optimization

4.4.3.1 Technology and Science Drivers for Performance Optimization

Exascale systems will consist of increasingly complex architectures with massive numbers of potentially heterogeneous components and deeper memory hierarchies. Meanwhile, hierarchies of large, multifaceted software components will be required to build next-generation applications. Taken together, this architectural and application complexity is compounded by the fact that future systems will be more dynamic in order to respond to external constraints such as power and failures. As reduced time-to-solution is still the major reason to use supercomputers, powerful integrated performance modeling, prediction, measurement, analysis, and optimization capabilities will be required to efficiently operate an exascale system.

4.4.3.2 Alternative R&D Strategies for Performance Optimization

In the exascale regime the challenges of performance instrumentation, analysis, modeling, and engineering will be commensurate with the complexity of the architectures and applications. An instrumented application is nothing but an application with modified demands on the system executing it. This makes current approaches for performance analysis still feasible in the future as long as all involved software components are concurrent and scalable. In addition to increased scalability of current tools and the use of inherently more scalable methods such as statistical profiling, techniques such as automatic or automated analysis, advanced filtering, on-line monitoring, clustering, and analysis as well as data mining will be of increased importance. A combination of various techniques will have to be applied.

Another alternative is a more performance-aware and model-based design and implementation of hardware and software components from the beginning, instead of trying to increase the performance of a functionally correct but poorly performing application after the fact.

In addition to user-controlled analysis and tuning, especially on higher-level (internode) components of the X-stack, self-monitoring, self-tuning frameworks, middleware, and runtime schedulers—especially at node levels—are necessary. Autotuning facilities will be of great importance here.

Worse, all of these approaches might not work for machine architectures that are radical departures from today's machines. This situation likely will need fundamentally different approaches to performance optimization.

In the performance modeling area, new methodologies will be needed that go beyond a static description of the performance of applications running on the system, to capture the dynamic performance behavior under power and reliability constraints. Performance modeling will also be a main tool for the co-design of architectures and applications.

4.4.3.3 Recommended Research Agenda for Performance Optimization

The following considerations are key for a successful approach to performance at exascale:

- Continued development of scalable performance measurement, collection, analysis (online reduction and filtering, clustering), and visualization (hierarchical) facilities. Here, performance analysis needs to incorporate techniques from the areas of feature detection, signal processing, and data mining.
- Support for modeling, measurement, and analysis of heterogeneous hardware systems.
- Support for modeling, measurement and analysis of hybrid programming models (mixing MPI, PGAS, OpenMP, and other threading models, accelerator interfaces).
- Automated/automatic diagnosis and autotuning.
- Reliable and accurate performance analysis in the presence of noise, system adaptation, and faults. This work will require inclusion of appropriate statistical descriptions.
- Performance optimization for metrics other than time (e.g., power).
- Performance observability and control by hardware and software components through appropriate interfaces and mechanisms (e.g., counters). The aim is to provide sufficient performance details for analysis if a performance problem unexpectedly escalates to higher levels. Vertical integration

across software layers (OS, compilers, runtime systems, middleware, and application) will be needed for this task.

- Design of programming models with performance analysis in mind. Software and runtime systems must expose their model of execution and adaptation as well as their corresponding performance through a (standardized) control mechanism in the runtime system.

Timeframe	Targets and Milestones – Performance Optimization
2012-13	• Support for hybrid programming models (mixing MPI, PGAS, OpenMP, and other threading models, accelerator interfaces) • Support for modeling, measurement, and analysis, and autotuning on/for heterogeneous hardware platforms
2014-15	• Handling of observation of million-way concurrency • Predictive exascale system design
2016-17	• Handling of observation of hundreds of million-way concurrency • Characterize performance of exascale hardware and software for application enablement
2018-19	• Handle observation of billion-way concurrency

4.4.3.4 Cross-Cutting Considerations

In order to ensure performance analysis and optimization at exascale, the various components and layers of the X-stack must be transparent with respect to performance. This *performance intransparency* will result in escalation of unforeseen problems to higher layers, including the application. It is not a new problem, but certain properties of an exascale system significantly increase its severity and significance.

- At this scale, there always will be failing components in the system with a large impact on performance. A real-world application will never run on the exact same configuration twice.
- Load-balancing issues limit the success even on moderately concurrent systems, and the challenge of locality will become another severe issue that has to be addressed by appropriate mechanisms and tools.
- Dynamic power management (e.g., at the hardware level inside a CPU) will result in performance variability between cores and across different runs. The alternative of running at lower speed without dynamic power adjustments may not be an option in the future.
- The unknown expectation of the application performance at exascale will make it difficult to detect a performance problem if it is escalated undetected to the application level.
- The ever-growing higher integration of components into a single chip and the use of more and more hardware accelerators make it more difficult to monitor application performance and move performance data out of the system unless special hardware support will be integrated into future systems.
- Performance comes from all layers of the X-stack, so an increased integration with the different layers, especially the operating systems, compilers, and runtime systems will be essential.

An integrated and collaborative approach clearly is needed to handle performance issues and correctly detect and analyze performance problems.

4.4.4 Programmability

Programmability is the cross-cutting property that reflects the ease by which application programs may be constructed. Although quantitative metrics are uncertain (e.g., SLOC) in their effectiveness, a qualitative level of effort in programmer time may reflect relative degree, noting that there is no "bell jar" programmer by which to make absolute comparisons. Programmability itself involves three stages of application development: (1) program algorithm capture and representation, (2) program correctness debugging, and (3) program performance optimization. All levels of the system, including the programming environment, the system software, and the system hardware architecture, affect programmability. The challenges to achieving programmability are myriad, related both to the representation of the user application algorithm and to underlying resource usage.

- Parallelism – sufficient parallelism must be exposed to maintain exascale operation and hide latencies. It is anticipated that 10 billion-way operation concurrency will be required.
- Distributed Resource Allocation and Locality Management – to make such systems programmable, the tension must be balanced between spreading the work among enough execution resources for parallel execution and colocating tasks and data to minimize latency.
- Latency Hiding – intrinsic methods for overlapping communication with computation must be incorporated to avoid blocking of tasks and low utilization of computing resources.
- Hardware Idiosyncrasies – properties peculiar to specific computing resources such as memory hierarchies, instruction sets, and accelerators must be managed in a way that circumvents their negative impact while exploiting their potential opportunities without demanding explicit user control.
- Portability – application programs must be portable across machine types, machine scales, and machine generations. Performance sensitivity to small code perturbations should be minimized.
- Synchronization Bottlenecks – barriers and other overconstraining control methods must be replaced by lightweight synchronization overlapping phases of computation.

4.4.4.1 Technology and Science Drivers for Programmability

As a cross-cutting property of exascale systems, programmability is directly affected by all layers of the system stack. The programming model and language provide the application programming interface to the user, determine the semantics of parallel computing, and deliver the degree of control and abstraction of the underlying parallel execution system. The compiler assists in extracting program parallelism, establishing granularity of computing tasks, and contributing to task scheduling and allocation. The runtime system is critical to exploiting runtime information and determines the level of dynamic adaptive optimization that can be exploited. The operating system supports the runtime system by providing hardware resources on demand and providing robust operation. And, while not part of the software system, the architecture directly affects programmability by fixing the overhead costs, latency times, power requirements, memory hierarchy structures, heterogeneous cores, and other machine elements that determine many of the challenges to programming and execution.

4.4.4.2 Alternative R&D Strategies for Programmability

The two general strategies for programmability are *evolutionary,* based on incremental extensions to conventional programming models, and *revolutionary,* based on a new model of computation that directly addresses the challenges to achieving exascale computing. The evolutionary strategy is expected to be pursued as part of community efforts to extend common practices as far into the trans-petaflops performance regime as possible. The MPI-3 Forum, the HPCS program, and the roadmaps for Cray and IBM indicate possible trajectories of such incremental approaches. Hybrid programming models derived from the integration of MPI and OCL or UPC have been suggested to achieve higher levels of scalability through hierarchical parallelism while retaining compatibility with existing legacy codes, libraries, software environments, and skill sets. However, it is uncertain as to how far the evolutionary approach can be extended to meet the escalating challenges of scalability, reliability, and power. The evolutionary strategy also assumes incremental extensions to current operating systems, primarily Unix derivatives (e.g., Linux), that can improve efficiency of synchronization and scheduling while retaining the basic process, Pthreads, and file model.

The revolutionary path follows historical patterns of devising new paradigms to address the opportunities and challenges of emergent enabling technologies and the architectures devised to exploit them. Revolutionary programming models and contributions at other system layers can be created to minimize the programming burden of the programmer by employing methods that eschew the constraints of earlier techniques while reinforcing the potential of future system classes.

4.4.4.3 Recommended Research Agenda for Programmability

Unlike programming models and languages, programmability spans all components of the system stack, both system software and hardware architecture, that in any way influence the usability of the system to craft real-world applications and have them perform correctly and with optimal performance through minimum programmer time and effort. Thus, while research in programmability must include factors of programming models, languages, and tools, it will also consider compilers, runtime systems, operating systems, and hardware architecture structures and semantics.

New Model of Computation: In synthesizing the effects of potentially all system layers on programmability, a single unifying conceptual framework is required to provide the governing principles establishing the functionality and interoperability of the system components to operate in synergy and realize critical performance properties. CSP, the common scalable execution model for STEM application targeted systems, is already unduly stressed in support of present multicore/many-core heterogeneous systems and cannot, in its current form, be expected to achieve the required functionality for scalability, efficiency, and dynamic scheduling. Therefore, research must be conducted to devise a new, overarching execution model either as a dramatic extension of current practices or an entirely new (likely based in part on experimental prior art) model of computation explicitly derived to address the unique challenges of exascale computing.

New Programming Models and Methods: Research into new programming models and ultimately APIs, tools, and methods will be required in order to provide the user interface to construct new application (and system software) programs and to determine which responsibilities of control of exascale systems will devolve directly to the user and which will be assigned to lower levels of the system, thus relieving the user of these burdens (but possible inhibiting needed control as well). An important property of any new programming model is a clear separation of logical functionality from performance attributes distinguishing those aspects of code specification that convey across multiple platforms unchanged (portability) from those that must be adjusted on a per platform basis for performance optimization (tuning). Preferably, all machine-specific program optimizations will be accomplished by lower system layers. New programming models will have to greatly expand the diversity of parallelism forms and sizes over conventional control semantics to dramatically increase by many orders of magnitude exploitable concurrency. Additionally, whether entirely new or an extended derivative, the next-generation exascale programming models will have to interoperate with legacy codes, both application (e.g., numerical libraries) and systems software (e.g., parallel file systems), for ease of transition of community mission critical workloads to the new classes of exascale systems architecture. Future models need to include semantic constructs in support of the broad range of dynamic graph-based algorithms whose access, search, and manipulation can be very different from more prosaic vectors and matrices for which current systems have been optimized. Emergent programming methods will require new tools and environments to make the best use of them from a programmer perspective.

New Runtime Systems: Research into advanced runtime systems will be an important means of dramatically improving programmability supporting dynamic software behavior, such as load balancing, thread scheduling, processing and memory resource allocation, power management, and recovery from failures. Only runtime systems (and operating systems to some degree) can take advantage of on-the-fly system status and intermediate application software state that cannot be predicted at compile time alone.

This situation will be particularly true for systems of up to a billion cores and constantly changing system configurations. In particular, new runtime software will move most programming practices from a static methodology to dynamic adaptive techniques exploiting runtime information for improved performance optimization. Examples include the lightweight thread scheduling, context switching, and suspension management, as well as interthread synchronization, management of deep memory hierarchies, and namespace management. For dynamic graph-based problems, data-directed execution using the graph structure to efficiently define the parallel program execution will require runtime support.

New Compiler Support: While much of the responsibility of future compilers will reflect prior techniques for back-end support, many new responsibilities will accrue as well to drive the exascale systems of the future. Advanced compiler techniques and software will be required for automatic runtime tuning to match hardware architecture specific properties (e.g., cache sizes), for heterogeneous architectures, to interface with and support advanced runtime systems, to detect alternative forms of parallelism, for employing advanced synchronization semantics and primitives, to take advantage of more sophisticated messaging methods (e.g., message-driven mechanisms), and to involve new forms of active global address space and its management.

X-Gen Architectures: Although the actual development of exascale architectures is beyond the scope of the IESP program agenda, research in critical system software and programming methods will be sensitive to and have to respond to the emergence of new architectures. Of particular concern are methods to reduce the temporal and power overheads of parallel control mechanisms, optimize the exploitation of heterogeneous core architectures, support fail-safe reconfigurable system structure techniques for fault tolerance, engage in active power management, and support self-aware resource management.

New Operating System: While the execution model is the machine, as seen from the semantic perspective, the operating system is the machine from the usage viewpoint. The OS owns the system, manages its resources, and makes them available to the program layer as well as provides many services to that layer. A new operating system will be essential for the X-gen architectures and its supporting programming environments, including APIs, compilers, and greatly expanded runtime software. One of the most important attributes of a new OS is its order-constant scaling property such that it can operate at speed, independent of the scale of number of processor cores or memory banks. A second critical property is the management of an advanced class of global address space that can support multiple applications sharing all resources in the presence of the need for dynamic allocation and data migration even as it provides interjob protection. The new OS must support the greatly expanded role of the runtime system even as it takes on the added complexity of dealing with heterogeneous cores and deeper memory hierarchies. The old view of conventional processes and parallel OS threads will have to be revised, supporting much more lightweight mechanisms offered by the underlying architectures while yielding many responsibilities to the runtime software driven by application requirements and new programming models. The operating system will have to provide much more information about system operational state so that self-aware resource management techniques can be more effectively developed and applied for fail-safe, power-efficient scalable operation.

4.4.4.4 Cross-Cutting Considerations

Programmability is a cross-cutting factor affected by all layers of the system stack including software and hardware. It also is interrelated with other cross-cutting factors such as performance and potentially resilience. Whether there exists a relationship between programmability and power management is uncertain; however, when writing system software, one clearly needs to develop power management software for the operating system and possibly the runtime system.

Programmability and performance are tightly coupled. For high-performance computing, a major factor affecting programmability has been performance optimization. This relates to the exposure of application parallelism, locality management and load balancing, and memory hierarchy management. These components are expected to be even more important for exascale systems. The complexity at that extreme scale will require that the responsibility for all but parallelism (and even not all of that) be removed from the programmer and handled by a combination of compiler and runtime in cooperation with the operating system and system architecture.

With respect to reliability, it may be valuable for the programmer to have the option of dictating the required recourse in the presence of faults, such as recovery or prioritized actions (in the case of urgent/ real-time computing). However, default options should be prevalent and used most of the time, in order to minimize programmer intervention and therefore improve programmability.

4.5 Summary of X-Stack Priorities

Below we present a prioritized list of research and development items for each software component area in the X-stack. To assure that software efforts receive appropriate attention, we use two attributes for each effort:

- Uniqueness to exascale: Some efforts are concerned with exascale systems and have little relevance for less capable systems. Other efforts are relevant to exascale but will likely impact lesser systems (i.e., petascale and upper-end terascale); we refer to this as "spanning." And some efforts are important to all future scales of computing.
- Criticality for exascale: During early classification discussions, we determined that uniqueness to exascale was insufficient for prioritizing activities. In particular, although there are efforts that are not unique to exascale, some of these are essential for successful exascale computing. We classify an items criticality as either critical, unknown/indeterminate, or non-critical.

The following are examples:

- **Application-managed resilience - uniquely exascale and critical:** Resilience is an issue for many efforts. Historically, resilience has not required applications to do anything but checkpoint/restart. At present, there is general agreement that the entire software stack, including user and library code, will need to explicitly address resilience beyond the classic checkpoint/restart approach.
- **Many-core mathematical libraries –n not uniquely exascale but critical:** Many-core configuration is an essential element of all exascale plans, but libraries for many-core configurations are also critical for all levels of computing. Although exascale requirements may exceed those of scales, we should recognize and leverage other funding sources for this kind of work, clearly identifying and funding the uniquely exascale aspects of this work.

The table below lists each of the X-stack components along with their needed capabilities. Each component capability is followed, to the right, by its uniqueness and criticality at exascale level. The following scale is used:

<u>Uniqueness</u>	<u>Criticality</u>
Unique = 3	Critical = C
Spanning = 2	Unknown = U
Nonunique = 1	Noncritical= N

X-Stack Components	*Needed Capabilities*	*Exascale Uniqueness*	*Exascale Criticality*
Frameworks	Resilience API and utilities	3	C
	Multi-institutional/multiproject collaboration plan	2	U
	Tool chain development/selection	2	U
	Programming model evaluation/adoption	2	C

	Data placement	2	C
	Multicomponent simulation utilities	2	U
	Access to third-party libraries	1	C
Numerical Libraries	Fault-oblivious, error-tolerant software	3	C
	Asynchronous methods	2	C
	Overlap data and computation	3	U
	Self-adapting hybrid and hierarchical based algorithms	1	C
	Hybrid and hierarchical-based algorithms (e.g., linear algebra split across multicore and GPU)	1	U
	Algorithms that minimize communications	3	C
	Architecture-aware algorithms/libraries	3	C
	Autotuning-based software	1	U
	Standardization activities	1	U
	Energy-efficient algorithms	2	U
	Mixed arithmetic	1	U
Algorithms	Scalability	2	N
	Fault tolerance/resilience	1	N
	Conforming to architectural requirements	3	N
	New areas/uses of algorithms	1	U
Debugging	Concurrency and architecture driven high frequency of errors/failures	3	C
	Scalability of debugger methodologies (data volumes and interfaces)	3	C
	Focus on multilevel debugging, communicating details of faults between software layers	3	U
	Synthesis of fault information into understanding in the context of apps and architecture	3	C
	Specialized lightweight operating systems	2	N
	Automatic triggers, need compile time bridge to debugging that removes need to rerun	2	N
	Scalable clustering of apps process states and contexts, filter/search within debugger	2	N
	Vertical integration of debug and per information across software layers	2	N
	Excision of buggy code snippets to run at lower concurrencies	1	N
	Heterogeneity	1	N
I/O	Customization with I/O, purpose-driven I/O	3	C
	New I/O models, SW, runtime systems and libs	3	C
	Intelligent/proactive caching mechanisms for I/O	3	N
	Fault-tolerant mechanisms	3	C
	I/O into programming models and languages	3	N
	Balanced architectures with newer devices	2	N
	File systems or alternative mechanisms	2	N
	Active storage	2	N

	Wide-area I/O and integration of external storage systems	2	N
	Special-purpose network protocols for parallelism	2	N
	Balanced architectures with newer devices embedded with the node	1	N
Scientific data management	Scalable data analysis and mining SW and tools	3	C
	Scalable data format and high-level libraries	3	C
	Scientific workflows tools	2	C
	Search and query tools	2	N
	Wide-area data access movement and query tools	2	N
	Scientific databases	2	N
Programming models	Exascale programming model	3	C
	Scalable, fault-tolerant MPI	3	C
	Applications development tools	3	N
	Heterogeneous node programming model	2	C
	Domain-specific programming models	2	N
	Language features for massively parallel I/O	2	U
	Language support for adaptive computation	2	N
	Interoperability between models	1	2N
Compilers	Implement exascale languages	3	C
	Support for resilience	3	C
	Implement heterogeneous programming models	2	C
	Support for massive I/O	2	C
	New optimization frameworks	2	N
	Interactions between compilers and tools, runtime	2	C
	Dynamic compilation, feedback optimization	2	N
	Autotuning-based software	2	N
	Enhancements to existing languages/APIs	1	N
	Automatic parallelization	1	N
Operating Systems	Define the base OS (Standard API)	3	C
	APIs for resilience (access to RAS, etc)	3	C
	Collective OS operations	3	N
	Scalable system simulation environment	2	C
	Improved APIs for scalable performance monitoring and debugging	2	C
	New APIs for energy management	2	U
	Improved APIs for explicit memory management	1	C
	Improved APIS for threading	1	U
Performance	Extremely scalable performance methods and tools	3	C
	Performance measurement and modeling in presence of noise/faults	3	C
	Automated/automatic diagnosis and autotuning	2	N
	Predictive future large-scale system design	2	C
	Vertical integration across SW layers	2	N
	Performance-aware design and implementation	2	U

	Performance optimization for other metrics than time	2	U
	Support for heterogeneous hardware and hybrid programming models	1	C
Power	Power performance monitoring and aggregation that scales to 1 billion core system	3	C
	Power control system	3	C
	Scalable control algorithms to bridge gap between global and local power models	2	C
	Power-aware and scalable resource control and scheduling	2	C
	Optimally tuned system power based on control loop	1	N
	Power instrumentation and control standardization	1	N
Programmability	New models of computation	3	C
	New runtime/OS interface for environment aware programming	2	C
	Programmability to decouple exascale system issues from applications programming	1	C
Resilience	Performance measurement and modeling in presence faults	3	C
	Better fault tolerance protocols	2	C
	Fault isolation/confinement	2	C
	NV-RAM for local state, cache of file system	2	C
	Replication (TMR, backup core)	2	U
	Proactive actions (migration)	2	U
	Domain-specific API and utilities for frameworks	2	C
	Applications-guided fault management	2	C
	Language/compiler/runtime support for resilience (fault-aware programming, API from OS, RAS)	2	C
	Fault-tolerant MPI	2	C
	Fault-oblivious, error-tolerant numerical libraries	2	C
	Resilient applications and algorithms	1	N
	Fault-oblivious system software	2	C
	Fault-aware system software and API for resilience	2	C
	Prediction for time optimal checkpoint/migration	2	U
	Fault models, event log standardization root cause analysis	2	C
	Resilient I/O, storage, and file systems	2	C
	Situational awareness	2	C
	Experimental environment	2	C
	Fault isolation/confinement + local management	2	C
Runtime Systems	Load balance	3	C
	Asynchrony, overlap	2	C
	Hierarchical execution models and scheduling	3	N

	Scaling/optimization of communications	3	C
	Memory management and locality scheduling	2	C
	Heterogeneity: scheduling	2	U
	Fine-grained mechanisms at node level	1	U

5. Application Perspectives and Co-Design Vehicles

Standing at the beginning of the road to exascale, application communities that are highly motivated to take that road are well aware of the challenges confronting them. Many of the applications for which exascale systems will be built exist today in high-performance implementations. But all of them will have to be rewritten substantially, in terms of data structures, algorithms, and possibly even mathematical formulations; any new applications under development should be formulated from the start with exascale in mind. As applications custodians and exascale customers, we respond by considering how particular applications—so-called co-design vehicles, or CDVs, after the principal new programming paradigm in the exascale regime—will migrate to the exascale. Here, we summarize several factors that we believe are key to exascale success for application communities. We then present the concept of CDVs; describe some of their issues, limitations, and requirements; and give the first examples of what we hope will be a diverse portfolio of CDVs that can help drive the X-stack development process and start producing exascale science at the earliest possible date.

5.1 From Here to Exascale: An Application Community View

The application leaders who have been informing the development of the IESP roadmap over the past year recognize a certain disconnect between the planning effort the IESP has initiated and the current state of major science applications. Specifically, although the shared goal is to enable exascale science on exascale systems by the end of the decade, the reality is that today only a scant few applications can successfully exploit the power of current and emerging petascale systems. The difficulties involved in finding the support and recruiting the interdisciplinary teams needed to create such leading-edge applications is, no doubt, part of the explanation for this disconnect. But these same difficulties, perhaps in even higher degree, will confront the communities aiming toward exascale.

At the same time, participating application representatives have expressed a clear desire for exascale computational power in order to make fundamental progress in their respective areas. The sources of this desire are largely *intrinsic* to the process of scientific exploration: scientists want to resolve their models at their full, natural range of length or time; to accommodate physical effects with greater fidelity; to create models with degrees of freedom in all relevant dimensions; to better isolate artificial boundary conditions or better approach realistic levels of dilution; combine multiple complex models; to solve inverse problems, or perform data assimilation; to perform optimization or control; tand o quantify uncertainty and make statistical estimates with orders of magnitude more accuracy.

The computational obstacles to achieving these goals are easy to quantify for some applications, such as QCD, cosmology, and seismic inversion, which are already scaling extremely well and experiencing processing bottlenecks. The situation is harder to quantify but equally important for less uniform applications (e.g., reservoir monitoring) with complex geometry, adaptivity, and multiple phases with different physics. Such differences between application groups make it clear that the former group will not be able to adequately proxy for the latter in terms of defining X-stack requirements.

But some common obstacles, which are bound to become more prominent on the road to exascale, are already appearing in the experience of many groups. At the level of hardware architecture, for example, the most commonly envisioned path to exascale is thousandfold many-core at 1 GHz each, within a tightly coupled network of about 1 million such nodes. However, memory bandwidth is already limiting today's low core count nodes to less than 10% of peak on most applications, whose kernels offer little cache reuse (e.g., stencil operations or sparse matvecs). Processors are cheap, small in chip area (compared to memory), and relatively low in power, so there is no harm in having them in excess most of the time; but the opportunities for exploiting the main new source for performance are undemonstrated for most applications. At the much higher and more abstract level of interdisciplinary research, while there are opportunities for combining today's individually high-capability simulations into more complex simulations, there is no silver bullet for merging the data structures of the separate applications. Moreover, given the current state of software infrastructure, the data copying inherent in the code coupling will likely prevent exploitation of the apparent concurrency opportunities

Surveying such experiences in the light of projections by the IESP community about the probable path to exascale, we have identified the following items as keys to success for many application communities:

- *Programming model:* Prior to possessing exascale hardware, appllications groups can prepare themselves by exploring new programming models on many-core and heterogeneous nodes. Attention to locality and reuse is valuable at all scales and will produce performance paybacks today and in the future. New algorithms and data structures can be explored under the assumption that FLOPs are cheap and moving data is expensive. Considering mixed-precision algorithms and using lower precision wherever possible can also reduce bandwidth pressure.
- *Data I/O*: Many communities are already struggle to cope with a growing deluge of data, and this data flood presents both tremendous opportunities and challenges. In simple terms, an exascale machine, once the data is loaded up, is a 32-petabyte fast store, with lots of processors to graze over it. We expect that there will be many new and exciting applications to take advantage of such storage, for example, idata mining in climate modeling and astrophysics. Such applications can begin to be explored today in miniature on petascale computers with 300 terabytes. But it is widely agreed that the I/O—reading data in and writing data out for analysis, checkpointing, visualization, etc.—is already a bottleneck for some applications and is likely to become one for many fields as data quantities escalate.
- *Fault tolerance*: Applications people reluctantly recognize that fault tolerance is a shared responsibility. It is too wasteful of I/O and processing cycles to handle faults purely automatically through checkpoint/restart. Different types of faults may be handled different ways, depending on the consequences evaluated by scientific impact. For example, application developers and users can orchestrate strategic, minimal working set checkpoints.
- *Reproducibility*: Applications people realize that bit-level reproducibility is unnecessarily expensive most of the time. Although scientific outcomes must be runtime independent and machine independent, we have no illusions about bit-level reproducibility for individual pairs of executions with the same inputs. Since operands may be accessed in different orders, even floating-point addition is not commutative in parallel and on inhomogeneous hardware platforms. A new feature in the context of co-design, with an emphasis on low power (low-voltage switching), is that lack of reproducibility may emerge for many other (hardware-based) reasons. If applications developers are tolerant of irreproducibility for their own reasons (e.g., for validation and verification through ensembles), then this has implications for considering less expensive, less reliable hardware.

5.2 IESP Application Co-Design Vehicles

Co-design vehicles are applications that provide targets for, and feedback to, the software research, design, and development efforts in the IESP. These are required because there are several possible paths to exascale, with many associated design choices along the way. The earliest realizations will include some of today's terascale or petascale applications that have a clear need for exascale performance and are sufficiently well understood that the steps required to achieve it can be mapped out. CDVs are accordingly a key part of the exascale design and development process. However, the specific domain applications themselves are not necessarily the scientific or societal drivers for developing exascale capabilities.

A CDV must satisfy the following criteria:

1. It is a petascale or near-petascale application today with a demonstrated need for exascale performance.
2. In progressing to exascale, it should achieve significant scientific goals in an area that is expected to be a scientific or societal driver for exascale computing, such as basic physics, environment, engineering, life sciences, or materials. Ideally, the results of the application should be amenable to experimental validation. This criterion is designed to help ensure that the effort elicits the necessary support from at least one agency.
3. It should offer realistic and a definable set of steps to exascale that can be mapped out over 10 years or less.
4. The community developing and supporting the CDV application should be experienced in algorithm, software, and/or hardware developments and be willing to engage in the exascale co-design process. In other words, there must be at least one organized research group, considered to be among the leaders in the field, that is interested in and willing to work with the IESP.
5. The CDV should be modular and open enough to stimulate the development of additional modules addressing related questions in the area.
6. Since the X-stack will need to be stressed along a number of different dimensions, the CDV should fill a slot in the portfolio of extreme-scale application needed to test all these dimensions.

The IESP will identify a manageable portfolio of CDVs (e.g., 4 or 5) that span the full range of anticipated software challenges. A short list of the most important science drivers in a specific application's domain will be articulated, and then a description provided of what the barriers and gaps might be in these priority research directions (PRDs). The primary task for each candidate CDV is to demonstrate the need for exascale and what will be done to address the PRDs. A major component of this activity is to identify what new software capabilities will be targeted and to what purpose. Further, it is necessary to describe how the associated software R&D can be expected to help the targeted application benefit from exascale systems, in terms of accelerating progress on the PRDs. With regard to developing an appropriate roadmap for this activity, it will be important to identify the timescale on which involvement in the path to exascale R&D can produce significant exascale-enabled impact. The choice of CDVs will be informed by the matrix of HPC applications versus software components (Section 5.3).

Different categories of CDVs include (1) societally relevant simulations (e.g., climate, patient-specific medicine); (2) more likely readily scaled simulations (e.g., QCD, cosmology); (3) data-processing problems (e.g., Square Kilometer Array in Australia, which generates 1 EB/s of data and needs FFTs per image while data is streaming); and (4) surprise outsiders, not currently practical at the terascale or petascale.

5.3 Initial Considerations for CDV Analysis

The application participants in the IESP have begun to develop an analysis of the issues, limitations, and needs to be addressed to make good use of CDVs in the X-stack research and development process.

Issues for Scaling Up CDVs: The big question in terms of CDV scalability concerns whether the software for co-design factors or whether all the inefficiency, over time, involves data copies at interfaces between the components. In selecting CDV applications to move toward exascale, in a staged co-design process, types that need to be examined include the following:

- Weak-scaling applications, up to distributed-memory limits and/or proportional to the number of nodes

- Strong-scaling applications, beyond distributed-memory limits and/or proportional to cores per node/memory unit
- Applications whose workflow scales, proportional to the number of instances (ensembles) and/or in integrated end-to-end simulation

Limitations to Be Explored by CDVs:

- Strong scaling algorithms may be limited in terms of sufficient coarse-grained parallelism and may encounter problems with load imbalance due to irregular task/data size; bulk synchronous algorithms on 1 million nodes are not currently tolerant to load imbalance worse than one part per million for a synchronous task.
- For acceptable single-node performance, compiler-generated code for hybrid/multicore may be limited. Linear algebra kernels typically come with autotuning; but for nonstandard linear algebra kernels, we will need the autotuning tools, not just their output.

Needs to Be Addressed by CDVs:

- CDV developers need tools to generate domain-specific languages and to provide for powerful source-to-source transformations; to enhance composability in order to enable new science and expand developer and user communities (which implies *decreasing* complexity as we go to exascale); to write performance-portable code (retargetable) that can extend the effective lifetime of code over generations of hardware; and to implement domain-specific frameworks that both provide solutions of significant HPC problems and are interoperable, so as facilitate collaboration in an increasingly multidisciplinary future.
- Expanded or new programming models are neded that move more of the burden of managing the scheduling of computation and placement of data to runtime; expand intrinsically fault tolerant programming models to be relevant to a broader class of algorithms; increase the interoperability of programming models (GAS, MPI, Cilk, HPCS, etc.) that we already have.
- CDV developers must understand the design space trade-offs associated with options for power consumption and resilience, taking into account the nature of expected faults, including common signaled faults and especially silent faults.

5.4 Representative CDVs

To provide specific examples of CDVs that conform to the selection criteria, we focus here on the high energy physics/QCD and the plasma physics/fusion energy sciences areas. It should not be inferred that these are the highest priority applications in the path-to-exascale portfolio. The IESP is considering a range of applications as CDVs, including simulations with special relevance to vitally important problems (e.g., climate change, patient-specific medicine), and applications that involve extremely data-intensive analysis (e.g., the Square Kilometer Array in Australia). We expect to recruit more CDVs as IESP partners in 2010 in order to stress all critical aspects of the X-stack.

5.4.1 High Energy Physics/QCD

Simulations of QCD, the theory of the strong interaction between quarks and gluons that are the basic building blocks of hadrons, have played a pioneering role in the development of parallel and high-performance computing since the early 1980s. Today, lattice QCD codes are among the fastest-performing and most scalable applications on petascale systems. Through 30 years of efforts to control all sources of numerical uncertainty and systematic errors, the current state of the art is that fully realistic simulations are possible and starting to provide results for a range of quantities needed by the experimental program, relating to the masses and decays of hadrons, with uncertainties at the few-percent level. Expected discoveries at the LHC will drive the need to extend these simulations to other quantum field theories that might describe new physics underlying electroweak symmetry breaking.

Lattice QCD already has a long track record of acting as a CDV. Specifically, it meets all of the above criteria for exascale co-design:

- Lattice QCD codes sustain multi-teraflops performance today and appear capable of scaling linearly through the petascale range. They are compute-limited, specifically demanding a balance between compute and on-/off-node memory access speeds, so that scientific progress requires the highest possible sustained performance. In order to deliver realistic and sufficiently precise results for the range of quantities needed by today's experiments, lattice sizes must at least double, increasing the computational cost by a factor of more than 1000. Even larger lattices will open up more hadronic quantities to first-principles computation and require performances well into the exascale range.
- As lattice QCD codes sustain multi-petaflops, the original goal of the field—to solve QCD at the few-percent level for many of the simplest properties of hadrons—will be achieved. Not only will this be a major milestone for theory, but it will also enable experiments to identify possible discrepancies with the Standard Model and, hence, clues to new physics. In approaching sustained exaflops, sufficiently large lattices will be employed to extend these computations to multi-hadron systems, permitting nuclear physics to be computed also from first principles. Depending on what is discovered at the LHC, petascale/exascale simulations may help explain electroweak symmetry breaking.
- The pathway to early exascale performance for QCD requires developing multilayered algorithms and implementations to exploit fully (heterogeneous) on-chip capabilities, fast memory, and massive parallelism. Optimized single-core and single-chip complex linear algebra routines, usually via automated assembler code generation, and the use of mixed-precision arithmetic for fast memory access and off-chip communications, will be required to maintain balanced compute/memory access speeds while delivering maximum performance. Tolerance to and recovery from system faults at all levels will be essential because of the long runtimes. In particular, use of accelerators and/or GPGPUs will demand algorithms that tolerate hardware without error detection or correction. The international nature of the science will demand further development of global data management tools and standards for shared data.
- The lattice QCD community has a successful track record in co-design, extending over 20 years and three continents: for example, the QCDSP and QCDOC projects in the United States, the series of APE machines in Europe, and CP-PACS in Japan. Notably, design features of QCDOC influenced IBM's Blue Gene. In all cases, QCD physicists were involved in developing both the hardware and system software. Typically, these projects resulted in systems that achieved performances for QCD comparable to the best that could be achieved at the time from commercial systems. The community has also agreed on an international metadata standard, QCDML.

As a CDV, lattice QCD has already been adopted by IBM for stress testing and verification of new hardware and system software. Other cross-cutting outputs from a QCD CDV are likely to include performance analysis tools, optimizing compilers for heterogeneous microprocessors, mechanisms for automatic recovery from hardware/system errors, parallel high-performance I/O, robust global file systems and data sharing tools, and new stochastic and linear solver algorithms.

5.4.2 Plasma Physics/Fusion Energy Sciences

Major progress in magnetic fusion research has led to ITER—a multi-billion-dollar burning plasma experiment supported by seven governments (EU, Japan, US, China, Korea, Russia, and India) representing over half of the world's population. Currently under construction in Cadarache, France, it is designed to produce 500 million watts of heat from fusion reactions for over 400 seconds with gain exceeding 10, thereby demonstrating the scientific and technical feasibility of magnetic fusion energy.

Strong research and development programs are needed to harvest the scientific information from ITER to help design a future demonstration power plant with a gain of 25. Advanced computations at the petascale and beyond, in tandem with experiment and theory, are essential for acquiring the scientific understanding needed to develop whole device integrated predictive models with high physics fidelity.

As a representative CDV, the fusion energy sciences (FES) area meets the criteria for exascale co-design:

- FES applications currently utilize the leadership computing facilities at ORNL and Argonne as well as advanced computing platforms at LBNL, demonstrating scalability of key physics with increased computing capability. Two high-performance computing FES topics with significant scientific impact were identified at the major DOE Workshop on Grand Challenges in FES & Computing at the Extreme Scale (April 2009): high physics fidelity integration of multi-physics, multiscale FES dynamics and burning plasmas/ITER physics simulation capability.
- A productive FES pathway of over 10 years can be readily developed for exploitation of exascale. This includes carrying out experimentally validated confinement simulations (including turbulence-driven transport) and demonstrates the ability to include higher physics fidelity components with increased computational capability. This is needed for both of the areas identified as PRDs, with the following associated barriers and gaps:
    - While FES applications for macroscopic stability, turbulent transport, edge physics (where atomic processes are important), and others have demonstrated, at various levels of efficiency, the capability of using existing LCFs, a major challenge is to integrate/couple improved versions of large-scale HPC codes to produce an experimentally validated, integrated simulation capability for the scenario modeling of a whole burning plasma device such as ITER.
    - New simulations of unprecedented aggregate floating-point operations will be needed for addressing the larger spatial and longer energy-confinement time scales as FES enters the era of burning plasma experiments on the reactor scale. Demands include dealing with spatial scales spanning the small gyroradius of the ions to the radial dimension of the plasmas (i.e., an order of magnitude greater resolution is needed to account for the larger plasmas of interest such as ITER) and with temporal scales associated with the major increase in plasma energy confinement time (~1 second in the ITER device) together with the longer pulse of the discharges in these superconducting systems.
- With regard to potential impact on new software development, each science driver for FES and each exascale-appropriate application approach currently involves the application and further development of current codes with respect to mathematical formulations, data structures, current scalability of algorithms and solvers (e.g., Poisson solvers) with associated identification of bottlenecks to scaling, limitations of current libraries used, and "complexity" with respect to memory, flops, and communication. In addition key areas being targeted for significant improvement over current capabilities include workflows, frameworks, verification and validation methodologies including uncertainty quantification, and the management of large data sets from experiments and simulations. As part of the aforementioned ongoing FES collaborations with the LCFs, assessments are moving forward on expected software developmental tasks for the path to exascale with the increasingly difficult challenges associated with concurrency and memory access (data movement approaches) for new heterogeneous architectures involving accelerators. Overall, new methods and exascale-relevant tools can be expected to emerge from the FES application domain. With respect to potential impact on the user community (usability, capability, etc.), the two FES PRDs noted earlier will potentially be able to demonstrate how the application of exascale computing capability can enable the accelerated delivery of much needed modeling tools. The timescale in which such impact may be felt can be briefly summarized as follows for the FES application: 10 to 20 petaflops (2012) for

integrated plasma core-edge coupled simulations and 1 exaflop (2018) for whole-system burning plasma simulations applicable to ITER.

5.4.3 Strategic Development of IESP CDVs

The technology drivers for CDV applications are, for the most part, connected to advanced architectures with greater capability but with formidable software development challenges. The need to address concurrency issues and to deal with complex memory access/data movement challenges for emerging heterogeneous architectures with accelerators is expected to drive new approaches for scalable algorithms and solvers. For risk mitigation, alternative R&D strategies need to be developed for choosing architectural platforms capable of effectively addressing the PRDs in the various domain applications while exploiting the advances on the path to the exascale. Beneficial approaches include the following:

- Developing effective collaborative alliances involving computer science and applied mathematics (e.g., following the SciDAC model)
- Addressing cross-cutting challenges shared by CDV applications areas through identification of possible common areas of software development, appropriate methodologies for verification and validation and uncertainty quantification, and the common need for collaborative interdisciplinary training programs to deal with the critical task of attracting, training, and assimilating young talent.

5.5 Matrix of Applications and Software Components Needs

The matrix below was created to stimulate and inform thinking about CDVs. Clearly all science areas and engineering areas that contain potential CDVs need *something* in all the software areas, but for the purposes of this exercise we tried to sort out areas of emphasis for each application domain, i.e., where we expect the major challenges will be for that domain. For example, all areas need some I/O, but the ones checked were deemed to need considerable I/O, based on the problems that exist today. Likewise, the areas that have less software maturity (e.g., health and energy) have more Xs in the programming, languages, and debugging columns.

Science and Engineering Disciplines	**Subareas**	**New Programming Models and New ways to specify**	**Programmability - Improved Code Development and Application**	**Resource management, power management**	**Dynamic Data Storage and Management**	**Libraries that exploit advanced HW and SW features**	**Resiliency and Fault Management**	**Debugging and Performance Tuning at Scale**	**System management and Security**	**Scalable Operating Systems**	**Support for Application Modeling**
Materials Science	• Nano-science • Structural Analysis • Electronic Structures		X	X		X	X	X		X	X
Energy Sciences	• Alternative Fuels • Nuclear Fission • Combustion • Nuclear Fusion • Solar • Energy Efficiency	X	X			X	X	X	X		X
Chemistry	• Molecular Dynamics		X			X				X	X

Science and Engineering Disciplines	Subareas	New Programming Models and New ways to specify	Programmability - Improved Code Development and Application	Resource management, power management	Dynamic Data Storage and Mangement	Libraries that exploit advanced HW and SW features	Resiliency and Fault Mangement	Debugging and Performance Tuning at Scale	System management and Security	Scalable Operating Systems	Support for Application Modeling
Earth Systems	• Climate • Weather • Earthquake/ Seismic • Subsurface Transport • Water Resources	X	X	X	X	X	X	X		X	X
Astrophysics Astronomy	• Dark Energy • Galaxy Formation/Interaction • Cosmic Microwave Background Radiation • Supernova • Sky Surveys		X	X	X	X	X			X	X
Biology/Life Systems	• Genomics • Protein Folding • Evolution • Ecology • Organism Engineering	X		X		X	X	X	X		X
Health Sciences	• Drug Design • Contagious Disease • Radiation-related Health • Medical Records • Comparative Genomics	X	X		X	X		X	X		X
Nuclear and High Energy Physics	• QCD • Neutrinos • Accelerator Design	X		X	X	X			X		X
Fluid Dynamics	• Internal • External	X	X	X		X				X	X

6. Perspectives on Cooperation between IESP and HPC Vendor Communities

In order to meet the many challenges involved in programming exascale machines, the components of the X-stack that the IESP community aims to produce must entrain a whole software ecosystem. As the size of the ecosystem grows, vendors will be increasingly motivated to leverage and contribute to the community's efforts to satisfy that ecosystem's requirements. In order to achieve this goal, however, several challenges must be overcome, including (1) finding a suitable structure to agree on common APIs; (2) producing a coordinated, interlocked effort between vendor partners, the IESP and scientific communities, and HPC facilities, with meaningful deliverables and time tables; (3) balancing the time needed for research and exploration to overcome the exascale hurdles with the need to produce timely, concrete implementations that can be integrated by the vendor partners and used by the IESP and scientific communities to run on the exascale systems; and (4) finding appropriate development, support, intellectual property, and funding models that allow vendor partners to incorporate software produced by the community, that can be supported by the community and funded by the interested government agencies.

Recent discussions among vendors as part of the IESP process have produced a number of considerations that need to be taken into account. We first expand on the likely challenges that need to be overcome for vendor partners to utilize the research and development efforts of the IESP community. We then present a taxonomy that describes the different models of development and support for software that might structure cooperation within the X-stack ecosystem. Next we describe the requirements and methods of such software. We conclude with a set of recommendations to help guide both the IESP community and vendors to effectively collaborate to produce the kind of ecosystem this collective effort needs.

6.1 Challenging Issues for Vendor/Community Cooperation

Common APIs: It is critical to agree on common and open APIs. The development and evolution of APIs must occur in a way that produces the kind of stability that IESP vendor partners need, but must also be flexible enough to incorporate early research and exploration of alternatives. Waiting to achieve agreement through slow-moving, formal standards processes may not be timely enough to meet the expected needs of X-stack software. There are components of the system software that need to take into account hardware-specific characteristics or that can be better tuned by exploiting hardware-specific features. Because multiple vendor partners will be working on such low-level aspects, it becomes even more important to the community to find a methodology to agree on common APIs, at least for the exascale effort.

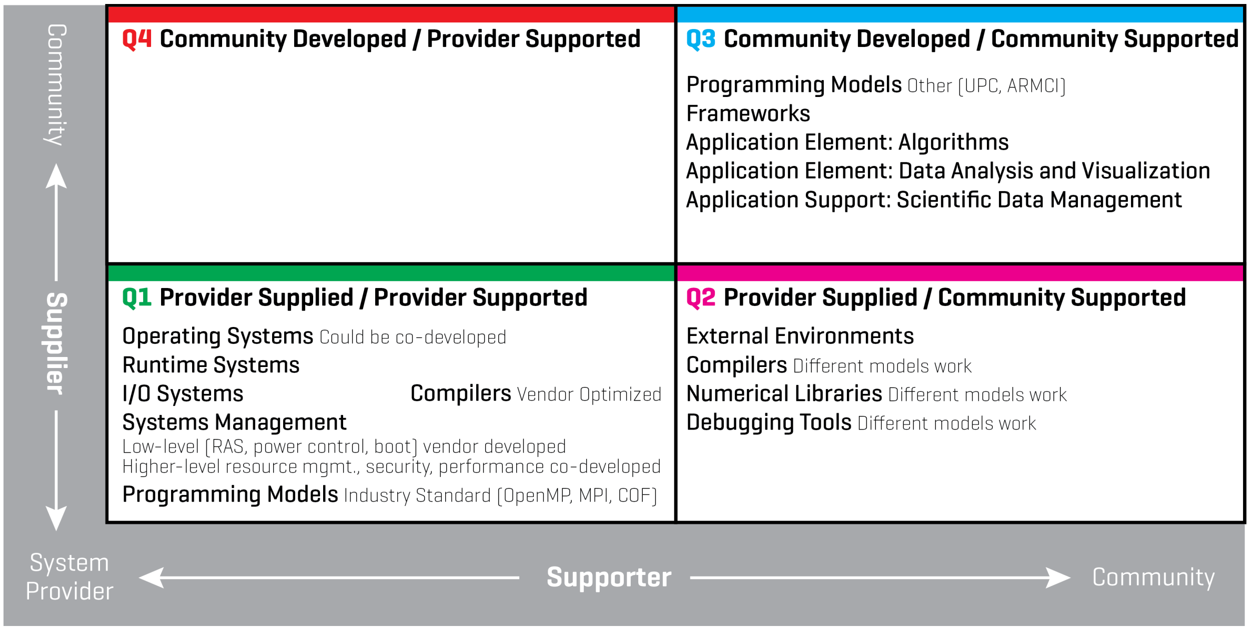
X-Stack Co-Development: The IESP community, vendor partners, and HPC facilities must work together to produce the software stack. The IESP community's message about the importance of vendor participation should be communicated clearly and repeatedly. If it appears that the community is going to fund all or most of the components of the X-stack, vendor partners will find it challenging to achieve the levels of software testing expertise and resources required to work with their results.

Research Time vs. Development Time: Research and early investigation are necessary in addressing exascale software challenges. It is also crucial that when the hardware becomes available, the software is sufficiently mature. For the interim system, targeted for 2015, time is short for making decisions on high-level issues (e.g., is programming model X the correct one for exascale?). It is important that funding agencies realize the urgency in producing solicitations and making funding available for the early investigations.

Support: Providing sufficient, on-going support for the components may be the largest nontechnical challenge facing the HPC community. Software researchers have typically not provided the level of support provided by vendor partners, and few research groups provide the level of support needed for HPC facilities to meet their traditional quality of service requirements. Further, to date there has not been a strong track record for the community coordinating with vendor partners tightly enough so the vendor partners could include software components in their product plan. In order to produce the rich software ecosystem the X-stack needs, a novel structure needs to be put in place to address these support issues.

6.2 Taxonomy of Development/Support Models

Thee vendor partners, funding agencies, and research and development community must have each software component in the X-stack categorized in terms of two key characteristics: (1) who is expected to develop/supply the component, and (2) who is expected to maintain and support the component. The figure below shows the four quadrants defined by these characteristics and how some of the component areas of the X-stack sort into them.



Q4 Community Developed / Provider Supported

Q3 Community Developed / Community Supported

Programming Models Other (UPC, ARMCI)
Frameworks
Application Element: Algorithms
Application Element: Data Analysis and Visualization
Application Support: Scientific Data Management

Q1 Provider Supplied / Provider Supported

Operating Systems Could be co-developed
Runtime Systems
I/O Systems
Compilers Vendor Optimized
Systems Management
Low-level (RAS, power control, boot) vendor developed
Higher-level resource mgmt., security, performance co-developed
Programming Models Industry Standard (OpenMP, MPI, COF)

Q2 Provider Supplied / Community Supported

External Environments
Compilers Different models work
Numerical Libraries Different models work
Debugging Tools Different models work

Elements in X-stack software roadmap categorized relative to supplier/support criteria from the vendor perspective. Cross-cutting areas—resilience, power management, performance optimization, and programmability—are not shown since they affect components at all layers and which may fall in different quadrants. As components are designed, the project owners should clearly identify the appropriate category for the component.

In Q1, the system provider both supplies and supports the component. This is the typical model of system providers who supply a proprietary software stack. However, the software components in this quadrant may also be open source, community-developed, co-developed, and/or third-party software components for which the system provider also provides support. In this context, then, "supplies" basically means "tests and packages for the system." Linux and MPI are often in this category for vendor partners.

In Q2, the system provider supplies a community-developed component, and the community provides the support. In this case, the system provider builds the component and supplies it to customers for each installation. Although the system provider does not maintain or support the component, it may be one of the contributors for that component in the community. PeTSC, ScaLAPACK, and *gcc* are examples from this quadrant.

In Q3, the component is developed/supplied and supported by the community. The facility and/or end-user obtains, builds, and installs the software on the system and works with the community for maintenance and support. For example, NWCHEM and *gnu* software are in this quadrant.

In Q4, the component is developed by the community, but the system provider is expected to be responsible to fully maintain and support the component. Examples in this quadrant are typically unique to specific customers. From the perspective of the vendor partners, this quadrant is an undesirable quadrant because, while they are expected to take responsibility for maintenance and support, they do not have enough control to sufficiently influence the component development/support community or control the destiny of the component. Consequently, facilities have difficulty obtaining the quality of support they are interested in.

From the system provider's perspective, components in Q1 and Q2 are appropriate as RFP requirements. However, only components in Q1 are appropriate as strong acceptance criteria. The Q3 is extremely difficult for the system providers and should be avoided whenever possible. There are no restrictions on

Q4 from the vendor partners, but there may be issues regarding the expectations of facilities managers and scientific users, and some of these issues may require alternative resource and/or funding streams.

While the system providers may participate in developing software in any of the quadrants, it is likely that system providers will be more active in the development of components in Q1 and Q2.

6.3 Requirements and Methods

The goals of the IESP effort challenge not only the technical capability of the HPC community but also the social and economic models that the community uses to create, integrate, test, and support software for emerging extreme-scale systems. Policies surrounding open source software offer one illustration of the challenge. On one hand, many government funding organizations require that software developed with public funds be available as open source. However, the absolute requirement for all software thus created to be open source makes it difficult for the providers of systems and the facilities deploying and supporting them as scientific tools to meet the quality of service objectives that the user community has come to expect. Pulling in the other direction, however, is the recognition that the HPC community is relatively small, while many hands are needed to craft viable solutions in the time available. This recognition is one of the primary reasons for trying to harness the entire international community to the effort. To engage everyone, there needs to be a shared and open way to work together. By its very nature, proprietary code tends to thwart goal and reduce the number of hands that can contribute.

To describe this tension and evaluate the tradeoffs, we define the requirements that science users have for the large X-stack software development effort, many of which we believe can be met by open software. The goals and expectations of computing center management, the software research community, and the scientific application users include the following:

- The community *does not want to be limited to proprietary solutions* over which they have little or no control. The features and improvements that have to wait for commercial providers to supply them can be problematic. Often these providers have priorities not always aligned with the HPC/exascale community, making improvements and/or corrections less timely and/or less functional than needed.
- Many aspects of exascale have a degree of uncertainty (risk) that strongly suggests having *alternatives for risk mitigation* and being prepared to replace components of the software stack in a timely manner.
- Software developers, ranging from application developers to system tool and feature developers, need *well-defined and consistent* APIs to which they can write code.
- Government organizations need to be able to *leverage their investments of public funds* in software development, so that results in one project or area can be reused for the multiple exascale hardware targets and for other non-exascale projects or areas as well.
- Government organizations need to be able to *protect their investments of public funds* in software development from being lost. In the past, significant publicly funded software (and hardware) investments have been lost when companies go out of business or change to other business models.
- Applications teams will be working to create highly scalable applications that run effectively on multiple system targets. These application teams want to have a cross-platform, or easily portable, *programming and development environment* to increase productivity.
- Exascale systems will be advanced scientific instruments. As part of the scientific process, scientists need to know how the devices work for *scientific reproducibility and accuracy*. Treating the system software as a black box run by code that cannot be examined or verified does not accomplish this goal.

System providers have their own requirements, some of which were expressed in the above provisioning and support graphic. The primary requirement is that system providers not be held responsible or liable for the correctness or performance of software over which they do not have control. Providers want the freedom, based on their business models, to use open source and other software components to meet requirements at their own risk; for example, they may decide to offer an open source component but budget the effort to provide the necessary support themselves. On the other hand, providers should not be held accountable for software they do not control. Sound business practices also dictate that providers be able to protect their proprietary information (e.g., low-level system hardware design), as has historically been the case.

The facilities that will deploy the exascale systems and help scientists make efficient use of the systems have traditionally made both explicit and implicit quality of service commitments to users and have accepted quality of service expectations/metrics from the funding agencies. Just like vendor partners, facilities are hesitant to rely on casual support agreements (e.g., open source) to resolve problems and make improvements in software that are critical to their success, particularly if they do not have the resources to provide their own full support for the component. Facilities, as surrogates for government stakeholders, also have to ensure the systems they deploy are the *best value* possible.

While there are overlaps, the methods below capture, to first order, the primary methods for developing and supporting software.

- *Open source* is defined, in the current context, as when all software is provided as buildable source code, with licenses that allow full rights for others to change and use the software without infringement to anyone's intellectual property. Support for the software may be casual to nonexistent. An example is the Perl scripting language.

- *Open source with formal support* is an enhancement of the open source in which all software is buildable source, as above, but in which there also exists a formal, or in some cases paid, arrangement for support of the software. An example is Lustre.

- *Open software* should be differentiated from open source. "Open software" refers to software where all APIs are published and supported and are not changed arbitrarily or unduly, but the buildable source code is not released with rights to use or modify. Open software allows software developers to create software that interfaces with other component (including application codes) and allows components to be replaced as long as the component has the same API.

- *Collaborative development* is a method that extends to both joint development and joint ownership of the software IP with a formal agreement defining roles, responsibilities, and rights. These agreements typically define a way to provide on-going support as well as original development. An example is the HPSS Collaboration.

- *Co-development* is a method that captures more ad hoc arrangements for joint development and support efforts. Co-development may co-exist with open software and/or open source. Examples in this category include MPICH and the ACTS toolkit.

- *Proprietary development* is the funded or unfunded development and support by an organization that retains the IP rights. For example, DARPA HPCS efforts fund vendor partners to create software that in some cases remain proprietary.

- *Proprietary development with code escrow* is the funded or unfunded development and support where the provider retains IP but formally agrees to release all software without restriction if certain conditions occur, such as the provider leaving the business or not longer wanting to support the software.

Requirement	Open Source	Open Source with Formal Support	Open Software	Collaborative Development	Co - development	Proprietary Development	Proprietary Development with Escrow
Community							
Does not want to be limited to a fully proprietary solution	X	X	X	X	?		
Flexibility to replace components of the stack	X	X	X	X	?		
Open API	X	X	X	X	X		
Leverage Gov't investment	X	X		X	X	X	
Protect Gov't investment	X	X		X	X	X	X
Applications have compatible environment	X	X	X	X	X	?	?
Scientists need to how their devices work for reproducibility	X	X		X	?	?	?
Provider							
Not held responsible for components that they do not have control over		X	X	X		X	X
Protect other provider proprietary information				X		X	X
Facility							
Level of Quality		X		X	X	X	X
Best Value		X		X	X	X	X

Table 1 – Matrix mapping the requirements for exascale software to methods of software development and support.

Table 1 characterizes which software development and support methods address which requirements of computing center managers, software research and development groups, and scientific application groups. An X means the method substantially addresses the requirements. A question mark means it may, with some restrictions, address the requirements. A blank space means the method does not support the requirement.

This table shows that the Collaborative Development approach addresses all the requirements, because there is shared responsibility and defined roles. More important, there is shared ownership of the software, so if one partner drops out of the relationships, other partners can continue. Open Source with Formal Support addresses all the requirements except for protecting the system provider from proprietary details if the software components have to interface to the hardware system at the low level (e.g., low-level interconnect features); in this case, releasing the code may implicitly release the proprietary hardware details.

6.4 Software Testing

So far, for the sake of simplicity, we have focused on software component development and support. In any large software development project, however, integration and testing (I&T) must be an integral and well-planned effort to ensure success, often taking at least as much effort and time as the actual code development. For the X-stack project, the situation is complicated by the fact that machines at this scale are unique resources, so they are the only place where testing can be done. As a consequence, all exascale and pre-exascale systems must, as part of their design, support the community I&T. Vendor partners are expected to take the responsibility for I&T in Quadrants 1 and 2 and are concerned that there are either explicit or implicit unfunded requirements for I&T in Quadrants 3 and 4. On the other hand, with a few exceptions, funders and facilities do provide sufficient resources to do the appropriate level of I&T without a vendor or facility incurring penalties.

In the case of X-stack, with the limited number of systems planned, the aggressive increase in scale, and the potential radical departures in hardware and software, the IESP roadmap must have a credible plan with clear responsibilities for integration and testing at expanding scales.

6.5 Recommendations

Discussion between the vendor partners, funding agencies, facilities, and IESP and the scientific community has yielded the following recommendations.

1. The IESP community should produce a methodology for categorizing software components into the development and support model they will fit. This should be broken down by each planned component, for example, OS, runtime, programming models. It is also possible that different instantiations within a component may be categorized differently; for example, within programming models, MPI and OpenMP may be treated different than UPC. Therefore, this process may need to iterate to gain a meaningful understanding of the X-stack creation and support plan. The result should be a “living document” and be refined as more information is learned about each of the components.

2. Interlocking (between vendor partner, community, facility) milestones should be clearly defined. In order to work effectively together and provide a mechanism for vendor partners to have confidence including “not invented here” components into their product plan, these milestones will allow the vendor partner, as the product roadmap progresses, to ensure the requirements are on track to meet the required schedule. As illustrated in the requirement versus method grid (Sec. 6.3), the co-development model, with joint ownership and responsibility with a formal agreement, meets the requirements.

3. The community should produce a model that allows for components to become mature before inclusion into the product stack. Linux, for example, was not supported by vendor partners until it had been existence for at least ten years. While this amount of lead time may not be needed for all components, a mechanism for allowing components to mature before inclusion is important.

4. As part of meeting co-development goals, the roadmap committee should interact with the application groups to identify key application characteristics, early enough to enable the characteristics to influence the hardware and software design tradeoffs. These characteristics can then be used as input into the

overall software architecture, requirements, and design. They can also be used by hardware architecture teams.

5. Funding agencies should apply resources to integration, testing, maintenance, and support as well as development of X-stack software. Enabling the community to effectively deploy and utilize the X-stack components requires a non-trivial investment of resources. Funding agencies, aware of this fact, need to be prepared to help underwrite that investment. Furthermore, there must be a model in place that allows the community to support that software. A good rule of thumb is that for every dollar dedicated to researching and developing a component, there should be a dollar dedicated for testing, maintenance, and support. Insuring the success of the IESP effort will require a well planned program of resource integration and testing.

6. Open source licenses from non-profits and publicly funded efforts should be vendor friendly. The pedigree of the code should track with contributor agreements, clearly indicating that the code is free of IP entanglements from the start. The license should be "nonviral" in order to allow the software to be included into vendor commercial products. In fact, this model should be encouraged, since it facilitates a more sustainable software base, not just for exascale, but for other efforts as well.

7. The community should start working early on to draft IP agreements with the goal of producing the bulk of the IP agreement that can be agreed to across countries, agencies, vendor partners, regions, components, and so forth. This is likely to need an even longer lead-time than the technology, so starting as soon as possible is highly recommended, since it will resolve many important questions and issues earlier rather than later.

7. IESP Organization and Governance

Initial discussions of a long-term organization and governance model for the IESP took place at the April 2010 meeting in Oxford. A relatively large group of representatives from participating governmental agencies, including representatives from the US (DOE, NSF, DARPA), European Commission, and Japan (MEXT, RIKEN), as well as national funding agencies from the UK (EPSRC, BBSRC, STFC), France (ANR, GENCI), Germany, and the Netherlands (NOW), considered potential governance models in various aspects. Below we present some of the main considerations on which the views of the participants converged.

7.1 Importance of a Business Case

Taking seriously the possibility of formally organizing the IESP and providing it with ongoing support means, first and foremost, acknowledging the validity of basic questions about the need for such an organization: Is the research and development of software for exascale systems really something new, especially as compared to the road to petaflop/s computing? Why is a separate project or program needed? What would happen if the funding agencies were to say, "Why bother: this regulates itself?" Deliberations about IESP governance began with such questions, which were pursued in something of a "devil's advocate" spirit. Although we concluded that there is, indeed, something new and uniquely challenging about the expected path to exascale software infrastructure, so that the IESP will require more formal organization and ongoing funding, it was also clear that *documenting a business case* for this will be essential in order to involve the funding agencies and provide them with the policy resources necessary to enable them to raise the funding. The costs and benefits for doing a *common* (i.e., international) project will have to be made clear.

Contents of a business case typically contain budget estimates, timelines, expected actors, roadmaps, risks, and contingency plans. It is believed that each funding agency will need a general business case, but should also have room for aspects in the business case that are of local importance to the country of the funding agency. This approach will ensure compatibility of business cases between the funding agencies.

Another important aspect is the scope of IESP. There is some question, for example, as to whether the IESP will end with the delivery of the first exascale system or whether it represents a distinctly new phase, which happened to begin just last year, of a continuous movement that will extend into the future.

A third important aspect that should be addressed by a business case is what can be called a tree or pyramid effect: Show that parts that are developed in IESP could and would be leveraged by a much broader user community several years after deployment. Such effects make funding agency and vendor interest stronger.

7.2 Application of Current Funding Mechanisms

One aspect to be addressed is the need for coordination of funding between the funding agencies (both within and among nations), once the business case has been validated. Currently, some types of funding calls can be identified, ranging from loose to much more regulated (loosely coupled, coordinated, joint, or in a well-specified legal framework). Either coordinated or joint funding models are considered the best options for the IESP. For example, a coordinated call might have characteristics such as the following: issued at the same time, having the same text proposal, and including several subjects within one call. Based on experience, it certainly seems feasible to have a few funding agencies working together to issue a coordinated call; but the larger the set of funding agencies participating, the better the coordination between the efforts will be. In this regard, an important aspect is the alignment of the subjects of the calls to the priorities of the funding agencies. Coordinated or joint call models should enable such appropriate alignments.

7.3 Governance Model

One of the key items of a working governance model for the IESP is the fact that the agencies funding the effort will need to remain in control of what they fund, why, and when. We believe that IESP should deliver to the funding agencies the analysis and planning resources that they will require to make such coordinated solicitations regarding exascale software infrastructure possible. One approach might be to have two separate tasks (and the entity to perform these tasks): one *defining* and one *monitoring*. The defining task would constitute the software roadmap and the breakdown of this roadmap into components, including timelines, procurable elements, and deliverables. This roadmap would need to take the business case as input and could be viewed as a practical plan of execution for IESP. The monitoring task would monitor progress on the roadmap, but across disciplines, borders, and agency domains, and would report and advise the funding agencies. The funding agencies could then decide on continuation of funding based on progress. Periodic updates and contingency plans will be needed. We view an approach based on such defining and monitoring tasks as a plausible and realistic way to move forward.

7.4 Vendor Interaction

An important aspect of sustainable relationships between vendors and funding agencies is the classification of software components with respect to ownership and ongoing or long-term support. Vendor perspectives on these issues are discussed in detail in Section 6. From an agency perspective, in the ideal situation, each software component of the X-stack would be open source. This approach was strongly advocated, if not firmly posed, as a requirement by the funding agencies represented in our initial discussions. But common sense dictates that some relaxation of this requirement will probably be appropriate if the software comes closer to the individual hardware components (e.g., firmware) because these components are likely to involve elements proprietary to the vendor. We also remark that this issue is not directly relevant if a vendor is not funded for the development of that component. The open source discussion has at least two facets. First, if X-stack research and development are to be funded by the government with public funds, funding agencies take the view that the results of such publicly funded research results should be open (source) to the people who paid for it. Second, the view that scientific experiments must be described in all detail and be reproducible is now being expressed by the community with increasing strength; to achieve this goal in research that uses exascale systems, all details of the software will have to be known. This requirement is independent of the IP rights discussion. It is more a matter of principle with respect to what constitutes valid scientific research. Licensing and IP issues are obviously related to practical questions about how valid scientific methods can be implemented and pursued in the coming era of exascale science. Although all details on these matters are not available yet, it clearly makes sense to try to anticipate the consequences of different rule sets and to plan accordingly, at an early stage of the IESP project. We plan to work with the results from the discussions of IESP vendor partners (Sec. 6) to begin fashioning such a plan.

7.5 Timeline

The timeline for the process will depend on the end point(s), the funding models and the levels of national and international cooperation and organization within the IESP. The end point(s) will be a function of the long-term requirements and goals of the different funding agencies involved in the process. At this time the first planned deployments are anticipated to be the U.S. Department of Energy. This first deployment sets the initial timeline for the overall software process.

In addition, there is clearly a need for a test and integration process and an intermediate scale facility to prepare for the initial deployment, which is likely to occur in 2015. Given these two points in the process and the current status, we can construct an initial timeline for the overall process. The early part of the process and the final state can be reasonably defined. The intermediate stages are still subject to considerable uncertainty.

The timeline below does not address other important issues about which discussions have already begun: security (rely on community-developed software components), testing and integration facilities, practical aspects of co-design, and funding of multiple approaches for similar software components. These items are slated for further development and will be included in future timelines.

Timeframe	Targets and Milestones – Performance Optimization
2010	• Initial mission-based software solicitations by DOE NNSA and Office of Science in the fall, with an expected emphasis on conservative technology choices • Creation of software roadmap, including requirements based prioritization, critical paths, funding and software clearinghouse, support models developed among the group of international agencies involved
2011	• Initial solicitations for software development programs based on the software roadmap for international partners
2012-13	• Initial software deliveries and evaluations
2014-15	• Delivery of final components of software stack, integration and testing in process on non-exascale platforms • Early technology delivery of a mini-exascale system of ~200 PF with a minimal but functional software stack
2016-17	• Ability to handle observation of hundreds of million-way concurrency • Characterization of performance of exascale hardware and software for application enablement
2018-20	• Initial delivery of full system with a full, integrated software stack • Ability to handle observation of billion-way concurrency • At-scale testing, debugging, and early scientific runs
2020	• Exascale systems in production

8. Contributing Authors to the IESP Roadmap

Author		Affiliation
Aerts	Patrick	Netherlands Organization for Scientific Research (Netherlands)
Aloiso	Giovanni	Euro-Mediterranean Center for Climate Change, University of Salento (Italy)
Andre	Jean-Claude	CERFACS (France)
Beckman	Pete	Argonne National Laboratory (US)
Berthou	Jean-Yves	EDF (France)
Boku	Taisuke	University of Tsukuba (Japan)
Braunschweig	Bertrand	Agence Nationale de la Recherche (France)
Cappello	Franck	INRIA (France)
Chapman	Barbara	University of Houston (US)
Chi	Xuebin	Chinese Academy of Sciences (China)
Choudhary	Alok	Northwestern University (US)
Dongarra	Jack	University of Tennessee (US)
Dosanjh	Sudip	Sandia National Laboratory (US)
Dunning	Thom	National Center for Supercomputer Applications (US)
Fiore	Sandro	Euro-Mediterranean Center for Climate Change, University of Salento (Italy)
Geist	Al	Oak Ridge National Laboratory (US)
Gropp	Bill	University of Illinois, Urbana-Champaign (US)
Harrison	Robert	Oak Ridge National Laboratory (US)
Hereld	Mark	Argonne National Laboratory (US)
Heroux	Mike	Sandia National Laboratory (US)
Hoisie	Adolfy	Los Alamos National Laboratory (US)
Hotta	Koh	Fujitsu (Japan)
Ishikawa	Yutaka	University of Tokyo (Japan)
Johnson	Fred	US Department of Energy (US)
Kale	Sanjay	University of Illinois, Urbana-Champaign (US)
Kenway	Richard	EPCC, University of Edinburgh (UK)
Keyes	David	Columbia University (US)
Kramer	Bill	National Center for Supercomputing Applications (US)
Labarta	Jesus	Barcelona Supercomputing Center (Spain)
Alain	Lichnewsky	Grand Equipement National de Calcul Intensif (France)
Lippert	Thomas	Jülich Supercomputing Centre (Germany)
Lucas	Bob	USC Information Science Institute (US)
Maccabe	Barney	Oak Ridge National Laboratory (US)
Matsuoka	Satoshi	Tokyo Institute of Technology (Japan)
Messina	Paul	Argonne National Laboratory (US)
Michielse	Peter	Netherlands Organization for Scientific Research (Netherlands)

Author		Affiliation
Mohr	Bernd	Jülich Supercomputing Centre (Germany)
Moore	Terry	University of Tennessee (US)
Müller	Matthias	Technische Universtät Dresden (Germany)
Nagel	Wolfgang	Technische Universtät Dresden (Germany)
Nakashima	Hiroshi	Kyoto University (Japan)
Papka	Michael	Argonne National Laboratory (US)
Reed	Dan	Microsoft Research (US)
Sato	Mitsuhisa	University of Tsukuba (Japan)
Seidel	Ed	National Science Foundation
Shalf	John	Lawrence Berkeley National Laboratory (US)
Skinner	David	Lawrence Berkeley National Laboratory (US)
Snir	Marc	University of Illinois, Urbana-Champaign (US)
Sterling	Thomas	Louisiana State University (US)
Stevens	Rick	Argonne National Laboratory (US)
Streitz	Fred	Lawrence Livermore National Laboratory (US)
Sugar	Bob	University of California at Santa Barbara (US)
Sumimoto	Shinji	Fujitsu (Japan)
Tang	William	Princeton Plasma Physics Laboratory (US)
Taylor	John	Commonwealth Scientific and Industrial Research Organization (Australia)
Thakur	Rajeev	Argonne National Laboratory (US)
Trefethen	Anne	Oxford University (UK)
Valero	Mateo	Barcelona Supercomputing Center (Spain)
van der Steen	Aad	National Computing Facilities (Netherlands)
Vetter	Jeffrey	Oak Ridge National Laboratory (US)
Williams	Peg	Cray (US)
Wisniewski	Robert	IBM (US)
Yelick	Kathy	Lawrence Berkeley National Laboratory (US)

Bibliography

[1] "Challenges In Climate Change Science and The Role of Computing at the Extreme Scale - Workshop Summary Report," Department of Energy, Washington, DC, pp. November 6-7, 2008.

[2] "Forefront Questions in Nuclear Science and the Role of High Performance Computing Summary Report - Summary Report," Department of Energy, Washington DC, pp. January 26-28, 2009. http://extremecomputing.labworks.org/nuclearphysics/PNNL_18739_onlineversion_opt.pdf.

[3] "Scientific Challenges for Understanding the Quantum Universe and the Role of Computing at Extreme Scale -- Summary Report," Department of Energy, Menlo Park California, Scientific Grand Challenges Workshop Series, pp. 129, December 9-11, 2008. http://extremecomputing.labworks.org/highenergyphysics/reports/HEPreport101609_final.pdf.

[4] M. L. Garcia and O. H. Bray, "Fundamentals of Technology Roadmapping " Sandia National Laboratory, pp. 34, 1997.
http://www.sandia.gov/PHMCOE/pdf/Sandia'sFundamentalsofTech.pdf.

[5] P. M. Kogge and et al, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," DARPA Information Processing Techniques Office, Washington, DC, pp. 278, September 28, 2008.
http://users.ece.gatech.edu/~mrichard/ExascaleComputingStudyReports/exascale_final_report_100208.pdf.

[6] National Research Council Committee on the Potential Impact of High-End Computing on Illustrative Fields of Science and Engineering, "The Potential Impact of High-End Capability Computing on Four Illustrative Fields of Science and Engineering," Washington, DC, pp. 142, 2008.

[7] V. Sarkar, et al., "ExaScale Software Study: Software Challenges in Extreme Scale Systems," DARPA Information Processing Techniques Office, Washington, DC., pp. 159, September 14, 2009.
http://users.ece.gatech.edu/~mrichard/ExascaleComputingStudyReports/ECSS%20report%20101909.pdf.

[8] V. Sarkar, W. Harrod, and A. E. Snavely, "Software challenges in extreme scale systems," *Journal of Physics: Conference Series* pp. 012045, 2009. http://stacks.iop.org/1742-6596/180/012045.

[9] R. Stevens, T. Zacharia, and H. Simon, "Modeling and Simulation at exascale for Energy and the Environment Town Hall Meetings Report," Department of Energy Office of Advance Scientific Computing Reserach, Washington, DC, pp. 174, 2008.
http://www.sc.doe.gov/ascr/ProgramDocuments/Docs/TownHall.pdf.

Appendix: IESP Meeting Attendees

(Attendees who were official observers are marked with "•")

		Affiliation	Country	SC08	Sante Fe	Paris	Japan	Oxford
Patrick	Aerts	NWO	NL		x	x	x	x
Giovanni	Aloisio	Euro-Mediterranean Centre for Climate Change	Italy			x	x	x
Dong	Ahn	LLNL	US	x				
Yutaka	Akiyama	Tokyo Tech	Japan				x	
Jean-Claude	Andre	CERFACS	France			x	x	x

		Affiliation	Country	SC08	Sante Fe	Paris	Japan	Oxford
Phil	Andrews	UT	US	x				
Mutsumi	Aoyagi	U Kyushu	Japan			x	x	
Mike	Ashworth	Daresbury	UK			x	x	x
Franck	Barbier	ANR	France			x		
Venkat	Balaji	Princeton	US					X
David	Barkai	Intel	US		x	x	x	x
Sanzio	Bassini	CINECA	Italy			x		
Kyriakos	Baxevanidis ●	EC	EU			x		x
Pete	Beckman	ANL	US	x	x	x	x	x
Jean-Yves	Berthou	EDF	France	x	x	x	x	x
Richard	Blake	Daresbury	UK		x			x
Jay	Boisseau	TACC	US	x				
Taisuke	Boku	U of Tsukuba	Japan		x	x	x	x
Bertrand	Braunschweig	ANR	France		x	x	x	
Bill	Camp	Intel	US		x			
Franck	Cappello	INRIA	France	x	x	x	x	x
Charlie	Catlett	ANL	US					x
Ruay-Shiung	Chang	National Dong Hwa University	Taiwan					x
Barbara	Chapman	U of Houston	US		x	x	x	x
Xuebin	Chi	CAS	China				x	x
Alok	Choudhary	NWU	US		x	x	x	x
Iris	Christadler	LRZ	Germany			x		
Almadena	Chtchelkanova	NSF	US		x			
Guillaume	Colin de Verdière	CEA	France			x		
Frederica	Darema	NSF	US		x			
Bronis	de Supinski	LLNL	US	x				
Peter	Coveney	U College London	UK					x
David	Dean ●	ORNL/DOE	* US				x	x
Mik	Dewer	NAG	UK					x
Jack	Dongarra	U of Tennessee	US	x	x	x	x	x
Sudip	Dosanjh	SNL	US		x	x	x	x
Thom	Dunning	NCSA	US	x				
Iain	Duff	Rutherford Lab	UK					x
Hugo	Falter	ParTec	Germany		x	x	x	
Teresa	Finchum	U of Tennessee	US				x	x
Leonardo	Flores ●	EC	EU					x
Fabrizio	Gagliardi	Microsoft	US			x		
Alan	Gara	IBM	US		x	x		
Al	Geist	ORNL	US		x			
Luc	Giraud	CERFACS	France		x	x		
Kostas	Glinos ●	EC	EU			x		x
Jean	Gonnord	CEA	France	x		x		x
Robert	Graybill	ISI	US		x			
Bill	Gropp	UIUC	US	x		x	x	x
Jim	Hack	ORNL	US	x				
Jean-Francois	Hamelin	EDF	France		x		x	
Robert	Harrison	ORNL	US				x	x
Bill	Harrod	DAPRA	US	x				

			Affiliation	Country	SC08	Sante Fe	Paris	Japan	Oxford
Stefan	Heinzel		Max Planck DEISA	Germany		x	x	x	x
Barb	Helland	●	OS	US	x				x
Mike	Heroux		Sandia	US	x	x	x	x	x
Ryutaro	Himeno		RIKEN	Japan		x		x	x
Kimihiko	Hirao		Riken	Japan				x	
Dan	Hitchcock		OS	US	x				
Thuc	Hoang		NNSA	US		x			
Adolfy	Hoisie		LANL	US				x	
Charlie	Holland		DARPA	US	x				
Koh	Hotta		Fujitsu	Japan				x	x
Herbert	Huber		LRZ	Germany					x
Yuichi	Inoue		MEXT	Japan				x	
Yutaka	Ishikawa		U of Tokyo	Japan		x	x	x	x
Satoshi	Itoh		MEXT	Japan				x	
William	Jalby		U of Versailles	France		x			
Jean-Pascal	Jégu		Teratec	France			x		
Zhong	Jin		CAS	China				x	x
Fred	Johnson		DOE	US	x		x	x	x
Andrew	Jones		NAG	UK			x	x	
Emma	Jones		EPSRC	UK					x
Laxmilkant	Kale		UIUC	US			x		
Richard	Kenway		EPCC	UK		x		x	x
David	Keyes		Columbia U.	US		x	x		x
Moe	Khaleel		PPNL	US				x	
Kimmo	Koski		CSC	Finland			x		
Bill	Kramer		NCSA	US		x	x	x	x
Dimitri	Kusnezov		NNSA	US	x				
Jesus	Labarta		BSC	Spain		x	x	x	x
Jean-Francois	Lavignon		Bull	France		x	x	x	x
Alain	Lichnewsky		Genci	France		x	x		x
Volker	Lindenstruth		Heidelberg U	Germany			x		
Thomas	Lippert		Juelich	Germany	x		x	x	x
Bob	Lucas		ISI	US		x	x		x
Barney	Maccabe		ORNL	US		x	x	x	x
Satoshi	Matsuoka		TiTech	Japan	x	x	x	x	x
Simon	McIntosh-Smith		U. Bristol	UK					x
Bob	Meisner		NNAS	US	x				
Paul	Messina		ANL	US	x		x	x	
Peter	Michielse		NWO	NL		x	x		x
Kazunori	Mikami		Cray	Japan				x	
Leighanne	Mills		U of Tennessee	US				x	
Bernd	Mohr		Juelich	Germany		x	x	x	x
Terry	Moore		U of Tennessee	US	x	x	x	x	x
Hervé	Mouren		Teratec	France			x		
Jean-Michel	Muller		CNRS	France			x		
Matthias	Müller		Dresden	Germany				x	
Wolfgang	Nagel		Dresden	Germany		x	x	x	x
Kengo	Nakajima		U of Tokyo	Japan				x	x

		Affiliation	Country	SC08	Sante Fe	Paris	Japan	Oxford
Hiroshi	Nakashima	Kyoto U.	Japan			x	x	x
Mamoru	Nakono	Cray	Japan				X	
Jeff	Nichols	ORNL	US		x		x	x
Jane	Nicholson	EPSRC	UK				x	x
Jean-Philippe	Nominé	CEA	France			x		x
Nick	Nystrom	PSC	US		x			
Per	Oster	CSC	Finland	x	x			
Mike	Papka	ANL	US					x
Abani	Patra	NSF	US		x	x	x	
Rob	Pennington ●	NSF	US		x			x
Serge	Petiton	CNRS	France			x		x
Claude	Puech	INRIA	France		x	x		
Tracy	Rafferty	U of Tennessee	US	x	x	x	x	x
Dan	Reed	Microsoft	US		x	x		
Michael	Resch	HLRS Stuttgart	Germany		x			x
Catherine	Rivière	GENCI	France		x	x		
Ralph	Roskies	PSC	US	x				
Faith	Ruppert	ANL	US		x			
Christian	Saguez	Teratec	France			x		
Vivek	Sarkar	Rice	US		x			
Stef	Salvini	Oxford	UK					x
Mitsuhisa	Sato	U of Tsukuba	Japan	x	x	x	x	x
Stephen	Scott	ORNL	US		x			
Mark	Seager	LLNL	US		x			
Ed	Seidel	NSF	US	x		x	x	x
Akiyuki	Seki	MEXT	Japan				x	
Satoshi	Sekiguchi	AIST/METI	Japan				x	
Hideo	Sekino	Toyohash Inst Tech	Japan				x	
John	Shalf	LBNL	US			x	x	
Horst	Simon	LBNL	US	x	x			
David	Skinner	LBNL	US		x	x	x	x
Marc	Snir	UIUC	US	x				
Mary	Spada	ANL	US	x				
Thomas	Sterling	LSU	US		x	x	x	
James	Southern	Fujitsu	UK					x
Rick	Stevens	ANL	US	x	x		x	x
Michael	Strayer ●	DOE OS	US	x	x			
Fred	Streitz	LLNL	US				x	x
Bob	Sugar	UCSB	US				x	
Shinji	Sumimoto	Fujitsu	Japan				x	
Makoto	Taiji	Riken	Japan			x	x	x
Toshikazu	Takada	Riken	Japan				x	x
Hiroshi	Takemiya	JAEA	Japan					x
Bill	Tang	PPPL	US				x	x
John	Taylor	CSIRO	AU				x	x
Rajeev	Thakur	ANL	US				x	x
Anne	Trefethen	Oxford	UK		x	x	x	x
Akira	Ukawa	U of Tsukuba	Japan				x	

		Affiliation	Country	SC08	Sante Fe	Paris	Japan	Oxford
Mateo	Valero	BSC	Spain	x		x		x
Aad	van der Steen	NCF	NL				x	
Jeffrey	Vetter	ORNL	US		x	x	x	x
Vladimir	Voevodin	Moscow State U	Russia			x		
Andy	White	LANL	US	x	x			
Peg	Williams	Cray	US		x	x	x	x
Robert	Wisniewski	IBM	US				x	x
Felix	Wolf	Juelich	Germany					x
Kathy	Yelick	LBNL	US	x	x	x		
Akinori	Yonezawa	U Tokyo	Japan				x	
Thomas	Zacharia	ORNL	US	x				x

10^{18}

IESP

www.exascale.org

University of Tennessee EECS Technical Report
UT-CS-10-662
October 18, 2010