The International Exascale Software Project RoadMap

Draft 11/15/09 11:55 AM

1. Introduction

The technology roadmap presented here is the result of nearly a year of coordinated effort within global software community for high end scientific computing. It is the product of a set of first steps taken to address an critical challenge that now confronts modern science and which is produced by a convergence of three separate 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 begin to meet this critical 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 ultrahigh 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 peta/exascale systems and for 2) catalyzing, coordinating, and sustaining the effort of the international open source software community to create that environment as quickly as possible.*

There are 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 are daunting, to say the least. 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.

Finally, 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 upon 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 all on its own.

The IESP was formed to help achieve this goal. Beginning in the spring of 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 first relatively complete version of that roadmap.
3. *Provide a framework for organizing the software research community*: With a reasonably stable version of the roadmap is 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, feasible schemes for selecting valuable software research and incentivizing its translation into usable, production-quality software for application developers, etc. 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 vendor community in crosscutting 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 stake holders 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 and 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 essentially addresses 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. The additional ideas and information we gather as the roadmap is disseminated are likely 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. The 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 the 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 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 an* ***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, etc.), the design of the X-stack should strive for modularity that makes it possible for many groups to contribute and accommodate more than one alternative 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 upon which subsequent sections this roadmap document builds.

3.1 Technology trends:

In developing a roadmap for X-stack software infrastructure, the IESP has been able to draw upon several thoughtful and extensive studies of impacts of the current revolution in computer architecture [ref. Kogge, etc.]. 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 can not be cost-effectively addressed with pairing or TMR, and will 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 10MW. Exascale systems could consume over 100MW, 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, and the X-stack and the applications it hosts must be conscious of this and action to minimize it.

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

More importantly, 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 also be perceived through analysis. The DARPA Exascale Technology Study

[Koegge, et al] exposes power as the pace setting 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 5X 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 grain 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 only become clear 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 Pico-joules 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 (this is approximate with a possible error of a factor of 2)
- Multi-threaded fine grain concurrency of 10 to 100 way concurrency per core
- 100's 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 Petabytes 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 inter-socket
- Optical bandwidth of 1 Terabit per second (+/- 50%)
- System-wide latencies on the order of 10's 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

3.2 Science trends:

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 only be studied through computational approaches; well-known examples include simulating complex processes in climate and astrophysics. Increasingly, experiments and observational systems are finding that the data they generate are not only exceeding petabytes and rapidly heading towards exabytes, but the computational power needed to process the data are 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. Here we only summarize some of the significant findings on the scientific necessity exascale computing, and focus primarily on the need for the software environments needed to support the science activities. The US Department of Energy 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." The four fields were astrophysics, atmospheric sciences, evolutionary biology, and chemical separations.

Likewise, the US National Science Foundation 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 online in 2011. It has commissioned a series of task forces to help it 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: there are significant scientific and engineering challenges in both simulation and data analysis that are already exceeding petaflops and are rapidly approaching exaflops class computing needs. In Europe the Partnership for Advanced Computing in Europe (PRACE) involves twenty partner countries and supports access to world-class computers and has activities aimed at supporting multi-petaflops and eventually exaflops-scale systems for science. The European Union is also planning to launch projects aimed at petascale and exascale computing and simulation. Japan has a project to build a 10 petaflops 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 out joint activities that will be needed to support these emerging science trends. (comment: should say something more specific about prace, japan, etc).

To give a specific and very 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:

- ¨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, that determine regional climate change with global warming?

[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

- ¨What are the future sea level and ocean circulation changes?

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

- 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 the obvious 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

- Verification of "carbon treaty" compliance
- Improving the safety, security & economics of nuclear fission
- Improve the efficacy of carbon-based electricity production & transportation
- Improve reliability and security in (electric) grid
- Nuclear fusion as 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, geothermal, etc., in many cases requiring 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 Relevant Politico-economic trends

The HPC market is growing at approximately 11% 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 very 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 no longer the case. 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 will not change, and that there will be little demand for those features of the X-stack unique to exascale computing from commercial HPC users.

3.4 Key requirements that these trends impose on the X-stack

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

- **Concurrency**: A 1000x increase in concurrency for a single job will be necessary achieve exascale throughput. New programming models will be needed to enable application groups to address concurrency in a more natural way. This will likely have to include "strong scaling" as growth in the volume of main memory won't match that of the processors. This in turn will require minimizing any X-stack overheads that might otherwise become a critical Amdahl fraction.
- **Energy**: As 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. This 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 a reliable as those used today. All software, and therefore every application, will have to address resiliency in a thoroughgoing way if it is to be expected to run at scale. This means that the X-stack will have to recognize and adapt to errors continuously, and 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 (i.e., STI Cell) while still providing a general purpose platform. An example of such 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 become more heterogeneous, address problems multi-scale 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.

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 three, into concrete recommendations for research and development agendas for each of the software areas and necessary

components of the X-stack. Accordingly, the discussion of each component or area is divided in the following parts:

- **Technology and science drivers**: The implications 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 each 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, in so far as we can see them today, need to be described and explored.
- **Research and development agenda recommendations**: Finally, the 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.
- **Crosscutting Considerations**: In many of these different parts of the X-stack, there will be interdependencies and crosscutting 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 be desirable to break out a summary of these considerations as a separate section in order to highlight gaps or to insure that activities are suitably coordinated. This version of the Roadmap focuses on four such crosscutting areas: resiliency, power/Total-Cost-of-Ownership, performance and programmability.

4.1 Systems 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, run time 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

Contributors: Barney Maccabe (ORNL), Pete Beckman (ANL), Fred Johnson (DOE)

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, to attain the benefits offered by an exascale system, there will be an increasing importance in the effective management of these resources.

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, it is all but certain that the cores 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.

In addition to the increasing changes in the resources provided by an exascale system, the programming models will undergo an equivalent evolution. In particular, non-MPI programming models will undoubtedly have increasing presence in exascale systems. The only trends that are clear at the present

time is there will be an increasing emphasis on data-centric 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 shmem. 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 shmem).

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, it is essential that we define a common operating system for the Exascale community. This will provide developer with a common set of APIs for the basis of their runtime systems. Moreover, it will provide the developers of Exascale systems with a common framework for exposing unique resources.

The ultimate goal of this Exascale community operating system is to provide a common API that could 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). This requires that the APIs supported by the operating system expose low-level resource APIs and that the runtime is aware of the context (within the application) of a specific computation.

4.1.1.2 Alternative R&D strategies for Operating Systems

There are several approaches that could be adopted in the development of a community OS for Exascale systems. One approach is to evolve an existing OS, e.g., Linux, Plan 9, IBM's Compute Node Kernel. An alternate 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 there are many runtime implementations that 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 provide 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.

It is essential that the operating system maintain a high degree of flexibility. This can only be accomplished 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, i.e., to complete the initial designs that will provide the foundation for the community OS. This should be undertaken by a small collection of researchers in the HPC-OS community who have significant experience with the implementation of 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. There are a number of efforts currently underway that address 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 intra-node communication, inter- and intra-node 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 ground breaking, 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

Time Frame	Targets and Milestones
2010-11	Community defined framework for HPC operating systems. This framework will define a set of core components and course grained APIs for accessing the resources provided by an HPC system.
2012-13	Scalable, full-system simulation environment. A full system simulation environment that can be used to evaluate resource management mechanisms at scale.
2014-15	APIs for fine-grained management of inter-node 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

Contributors: Jesus Labarta (BSC, ES), Rajeev Thakur (ANL), Shinji Sumimoto (Fujitsu)

4.1.1.4 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 otherwise force them to think about the computer rather than the science that they are trying to do. As description of the technology trends and science requirements above should suggest, it will be extremely challenging to create runtime systems that can continue to fulfill this. 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 (sec. 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 affectively in the limited timeframe that is typically available at 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, i.e. between alternatives that follow a flat runtime model (i.e., message passing), and those that follow 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 as the hybrid approach can certainly benefit from the flat approach pushing its capabilities to the limits. Another set of alternatives to explore general-purpose runtime systems, on the one hand, and application type/area specific (or customizable) runtime systems, capable of more effectively exploiting platform resources relative to special sets of needs, the other.

4.1.2.3 Recommended Research Agenda for Runtime Systems

Different topics are identified as challenging research topics, including 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 in 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 the exploit the level of granularity supported by the underlying hardware.
    - Anticipated research directions: 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 dimension broaden 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: general purpose self tuned runtimes that detect imbalance and reallocate resources (e.g. cores, storage, DVFS, bandwidth) within/across processes and other entities at the different level(s); 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, partial job preemptions, etc.

- Impact: research in this direction 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 to the programmers to achieve efficient resource utilization and make sure that the resources that cannot be profitably used are returned to the system to be reallocated.

- **Flat runtimes:**
  - A key challenge is to try and 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 presently 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.
  - Foreseen research directions are: optimization of resources/infrastructure needed for the implementation of 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); provide hierarchical implementations of flat models (e.g., thread based MPI, optimization of collective operations) ; adapt communication subsystems to application characteristics (routing, mapping, RDMA, etc.)
  - Impact: research in this direction 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:**
  - 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 an efficient utilization of the resources for any target machine. Constraining the size of the name/address spaces (i.e. ranks, amount of shared state) while still providing a fair level of concurrency/flexibility within each level is one of the challenges that motivates the hierarchical approach.
  - Foreseen research directions are: 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 direction will result in effectively matching the execution to the available resources. It will enable smooth migration paths from today's flat codes

4.1.2.4 Recommended Research Agenda

Time Frame	Targets and Milestones
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 "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, minimize memory consumption at large scale, maximization of performance to power ratio, malleability, tolerance to performance noise/interference, and 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 run time: tolerating injections 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 deliver useful results.
2018-19	Fully decoupling run time: dynamically handling all types of resources such as cores, bandwidth, logical and physical (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 that will have to be matched in a very dynamic environment.

4.1.2.5 Crosscutting considerations

The runtime functionality interacts with all cross cutting areas.

- Power management: The runtime will be responsible for measuring the application performance and decide 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 (ie. job scheduler) or to the user. The runtime will also need monitoring information about the performance of the computational activity of the application to take decisions of most appropriate resource for them or to select the appropriate power mode.

- Resilience: The runtime will be responsible for implementing some fine grain mechanisms (ie. reissue failed tasks, control speculative state) as well as to decide when to fire coarse grain 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.

4.1.2 I/O systems

Contributors: Alok Choudhary (Northwestern U.), Yutaka Ishikawa (U. of Tokyo, JP)

4.1.2.1 Technology and Science drivers for I/O Systems

There are many technology and science drivers for I/O systems ranging from 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, its future usage and availability also have influence on 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 shown not to be scalable as needed. That is, traditionally, I/O has been considered as a separate activity which 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 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, that 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, coherence; mostly ignoring 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 (SSDs) or Storage Class Memories (SCM) have the potential to significantly alter the I/O architectures, systems, performance and the software system to exploit them. 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.

4.1.2.2 Alternative R&D strategies for I/O Systems

There are many R&D strategies at different levels of the architecture and software stack (see below) that can potentially address the above technology drivers and for exascale systems. The metrics of I/O systems are performance, scalability, adaptability of applications, programmability, and fault resiliency.

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's where the most amount of semantic, data distribution, data usage and access pattern information is available. The middleware is not only for the single user space, but also cooperating with other user file I/O activities on the machine so that the system-wide optimization could 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 user space to perform I/O relation functions and optimizations on behalf of the applications. Using the application requirements, it can perform intelligent and proactive caching, data reorganization, optimizations, smoothening of I/O accesses from bursty to smooth patterns. This approach can provide services to the applications 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 its preprocessing before it is consumed. The availability of multicore nodes enable the opportunity to use one or more cores on each node to perform I/O services to using an exclusive set of select nodes, providing a range of customization options including locality enhancements.

2. Active Storage and Online Analysis: The concept of active storage is based on a 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, redistribution etc. This concept has a significant potential to help improve the 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 significant more memory and GPGPUs and 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 time axis. Optimizations in the two activities may require different approaches by the software layers. The software layers from file systems, middleware and high-level should be modified with incorporation of these capabilities by exploiting the purpose of I/O.
4. Software Systems for Integration of Emerging Storage Devices: There is a significant potential of emerging storage devices such as Solid-State Devices (SSD) and Storage Class Memories (SCM) to improve performance, reduce power consumption, improve caching; and can potentially reduce/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 system and middleware would be very important for the exploitation of these devices. Various approaches must be investigated along with the various options of using these devices within the exascale architecture (e.g., an SCM devices being part of each node's memory hierarchy to them being part of a separate section (subset) of the architecture that have these devices). These alternatives will 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. Extend Current File Systems: In this approach, 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. Develop New Approach to Scalable Parallel File Systems: Newer models, interfaces and approaches, which are not limited by sequential semantics and consistency models, that incorporate newer and highly scalable metadata techniques, that can exploit information available from user and higher levels and that can incorporate newer storage devices and hierarchies would be important.

7. Incorporate I/O into Programming Models and Languages: 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, which can be further be used by compilers to optimize I/O, pipeline I/O, and intelligently schedule I/O to maximize overlap with other computations; and in which multicore architectures can be exploited to utilize cores for enhancing I/O performance; specify online analysis functions on delegate systems of active storage are important research areas.
8. Wide-Area I/O and integration of external Storage Systems: This topic has components within the Scientific data management section. Scalable techniques are needed in which parallelism in accessing storage devices is integrated with parallelism with network streaming. Also, integrating parallel streaming of data over the network, using similar principles as those in parallel I/O would be important.

4.1.2.3 Recommended Research Agenda I/O Systems

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

Time Frame	Targets and Milestones
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 system 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

4.1.2.4 Crosscutting considerations

The architecture of the systems in general and in particular for storage and I/O systems, and their use of emerging devices will influence the I/O system software. Architectures should consider the issues outlined above in designing the I/O systems. I/O related communication and storage device usage would

significantly influence power optimizations. The I/O system software clearly has implications on resiliency, the schedulers, the operating systems and programming models and languages.

4.1.3 External Environments

4.1.3.1 Technology and Science drivers for External Environments

Lots of words

4.1.3.2 Alternative R&D strategies for External Environments

Lots of words

4.1.3.3 Recommended Research Agenda External Environments

Lots of words

Crosscutting considerations

Lots of words

4.1.4 Systems Management

Contributors: Robert Wisniewski (IBM) and Bill Kramer (NCSA)

Systems management comprises a broad range of technical areas. We divided the topics into five areas to be able to more tightly describe the challenges, research directions, and impact of each. The first area was "Resource control and scheduling". This is an area includes configuring, start-up and reconfiguring the machine, defining limits for resource capacity and quality, provisioning the resources, and workflow management. The second category is "Security" and includes authentication and authorization, integrity of the system, data integrity, and detecting anomalous behavior and inappropriate use. The third category is "Integration and test". It involves managing and maintaining the health of the system and performing continuous diagnostics. A related fourth category is "Logging, reporting, and analyzing information". 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). The final category ,"External coordination of resources", is how the machine coordinates with external components, for example how the HPC machine fits in a cloud. It comprises a common communication infrastructure, reporting errors in a standardized way, and integrating with in a distributed computing environment.

4.1.4.1 Recommended Research Agenda for Systems Management

We group the five above described topics in three areas for defining needed deliverables by the community. "Resource control and scheduling" and "External coordination of resources" is category 1, "Security" is category 2, and "Integration and test" and "Logging, reporting, and analyzing information", is category 3.

Time Frame	Targets and Milestones
2010-11	Category 1) Create and validate an analytic model and simulation capability for Exascale resource management that spans different implementation of job and resource management systems. This will enable experimentation of alternative designs that will accelerate implementation in the later time frames. 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 on-the-fly provide applications with more nodes and memory.

	Category 3) Unified framework for event collection: providing a community-agreed-upon standard format for events across machines and sub-systems within a machine.
2014-15	Category 1) Expand the 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 non-traditional resources, such as bandwidth, power: by using the above models and filters, and the dynamic provisioning of resources, provide the ability to manage new important resource such power and data motion. Category 3) Continual monitoring and test: by building on the unified framework for collecting data and filters, provide real-time monitoring and testing of the machine.
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 non-traditional resources, 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: need "defense in depth" security so that security does not solely rely on access control to the machine, develop end-to-end methodologies including integrated hardware to protect all components of the machine. Category 3) Proactive failure detection: For an exascale machine with the high component count and failure rate, it will be important to proactively predict failures. By building on the above continual monitoring and analysis tools, provide the ability to predict failures.

4.1.4.2 Technology and Science Drivers for System Management

In addition to the fundamental drivers mentioned above (scale, component count, scale, 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, albeit the time periods range 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. In the exascale time the tasks also have to be automatous 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 giga to peta scale – is processors and computational operations – are no longer the most constrained resource at exascale. DARPA studies listed in this report document data movement, rather than computational processing, will be the constrained resource at exascale. This is especially true when power and energy is taken into account as limiting design and total cost of ownership criteria. Hence, resource control and management – and the utilization logs for resources – has 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. (Note 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, data integrity) will have to be built into the many components of the system. The "defense in depth" concepts that are successful for facility wide security will have to be extended throughout the exascale system without impinging on performance or function.

Finally, system complexity is another driver at the exascale. HPC systems are exceedingly complex and susceptible to small perturbations having extraordinary impact on performance, consistency and usability. Taking the measure of number of transistors multiplied by the number of lines of code simultaneously in use as a measure of complexity, exascale systems will be 4 orders[2] of magnitude more complex than their petascale predecessors. The system manager's job is to manage this complexity to provide consistency high performance and quality of service. Without the re-invention of many of the tools used today, and the invention of new tools, system managers will not be able to meet those expectations.

4.1.4.3 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 system, 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 very likely exascale systems will have a PERC metric within an order of magnitude of exascale because of much less efficient resource management, much less consistency and much less reliability.

Another approach could be 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 it real time systems, which uses control theory, statistical learning techniques and other methods to management in a proactive manner, limited resources. 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.

4.1.4.4 Recommended Research Agenda System Management

Below we list and comment on 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"

- Need to better characterize and manage non-traditional resources such as power and I/O bandwidth
- Determine how to manage and control communication resources – provision and control, different for HPC than 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

[2] Estimates of today's vendor supplied system software contains 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 OS functions use $1/30^{th}$ 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 – $6*10^{14.}$ At Exscale, there may be 10,000,000 nodes. If the code complexity only doubles for Exascale, the complexity is $1.2*10^{14.}$ 4 orders of magnitude more complex in the simplest case.

- Coordinated resource discovery and scheduling aligned with Exascale resource management

There are five areas of research we identified for category 1. The first, is obtaining a better characterization of non-traditional resources such as power and I/O data motion. Related, is research into the ability of how to control that data motion. As part of that study, the community needs to identify whether additional hardware enhancements should be designed, for example, 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 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. In the event that a failure is not pre-detected, the system management infrastructure must be able to detect, isolate, and recover from the failure, by allocating additional equivalent resources. While there is effort underway in the application space to handle failures, system management research should target presenting application with machines where failures are corrected transparently by reallocating working resources to replace the failed ones. Finally, to integrate the HPC machine into a larger infrastructure, research should be undertaken to provide standardized reporting of machine definitions and capabilities and exist in a globally scheduled environment.

Category 2) "Security"

- Fine grained authentication and authorization by function/resources
- Security Verification for SW 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 just rely on traditional security at the boarders (login nodes). Essentially, fine grained authentication and authorization by function and for each resource is needed through all software and hardware components of the system. This has to be light weight so as not to restrict or slow down authorized use or limit scalability, while at the same time comprehensive 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 also 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 in an efficient manner. This will entire how 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 is to considered is to make 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. Finally, 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 improves 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, a survey and 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 developing 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 loosing critical events. For filtering, providing stateful techniques, where the dynamic state of the machine determines what events the filter provides, is critical. Techniques must be researched to allow this monitoring, filtering, and analysis to occur in real-time without effecting application behavior running on the system. Finally, the above defined research initiatives need to feed research of proactively determining where failures will occur by monitoring and analyzing filtered data.

4.1.4.5 Crosscutting considerations

System Management functionality crosses all aspects of the vertical integration – performance, usability/programmability, resilience, and power. System management is 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. Multi-level analysis of system usage, subsystem activities and component and subsystem health are needed to provide dynamically resource provision and to facilitate consistent and correct execution of application tasks.

4.2 Development Environments

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

4.2.1 Programming Models

Contributors: Barbara Chapman (U. of Houston), Mitsuhisa Sato, (U. of Tsukuba, JP), Taisuke Boku (U.of Tsukuba, JP), Koh Hotta, (Fujitsu), Matthias Mueller (TU Dresden, DE), Xuebin Chi (Chinese Academy of Sciences)

4.2.1.1 Technology and Science drivers for Programming Models

- 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 model 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 model and Languages should alleviate the programming difficulties arising from such heterogeneity.

- At the same time, 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.
- Modern high performance systems have complex memory hierarchies. Memory bandwidth is critically important even 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 increasing complexity of applications and the need to have an increasing level of detail represented in the simulated models require high programmer productivity.
- The use of deep, large software stacks require the capability to detect and isolate errors at various stages (code development, production, compile time, run time) and report them at an appropriate level of abstraction.

4.2.1.2 Alternative R&D strategies for Programming Models

The following alternative strategies are proposed:

- Hybrid vs. uniform: A hybrid programming model is a practical way to program exascale systems which 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, yet their provision is a challenge.
- Evolutionary and revolutionary approaches: Specification of incremental improvements to the existing models is a safe approach. Revolutionary approaches may be ambitious and 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.
- Widely embraced standards vs. single implementations: while the latter has the advantage of rapid development and implementation the former is based on the experience of a wider community and often required by application groups.

4.2.1.3 Recommended Research Agenda Programming Models

Research is needed into a variety of promising programming models for exascale computing, including system-wide 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. Both 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.

Time Frame	Targets and Milestones
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 available Standard programming model for heterogeneous nodes System-wide 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 Crosscutting considerations Programming Models

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 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, execute correctness checks that have been deferred to run time, collect 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 to enable the orchestration of the actions of different threads across the system. It 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 that are 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. They will need to be aware of the specific factors that influence program performance under a given programming model. In addition to the programming model, additional user annotations may need to be defined to support the actions of the supporting compilers and tools.

4.2.2 Frameworks

Contributors: Michael Heroux and Robert Harrison

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 would face a daunting collection of infrastructure requirements, independent of the science requirements. Frameworks (when properly developed) have been very successful at providing 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 retaining, or even improving, performance.

It is compelling to have a focused effort on frameworks for exascale systems for the following reasons:

1. We have a large body of existing scalable applications that we want to migrate toward exascale.
2. There are likely many yet-to-be-developed exascale-class applications.
3. The natural cost and feature benefits of frameworks provide the best cost and time approach to application development.
4. Exascale computing provides a new opportunity for multi-scale, multi-physics and multi-disciplinary applications.

4.2.2.2 Alternative R&D strategies for Frameworks

No frameworks: Most successful frameworks are developed in response to substantial experience developing individual components, where these components have substantial common requirements, have 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 may appear to be an attractive approach because it provides the shortest path to single application completion. However, as more applications are developed, this approach produces a lot of 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 re-factored. In this case, the framework will be best constructed in to solve a minimally interesting problem. Then existing applications will be "mined" for their useful software fragments. This approach was required for many applications when making the transition from vector multi-processors 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:

- Identify and develop 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 manycore libraries are common needs.
- Refactoring for manycore: In anticipation of manycore programming model decision, we must still make progress in preparing for exascale systems by understanding the common requirements of manycore programming that will be true regardless of the final choice in programming models.

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 are multi-institutional but most are primarily 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 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 capabilities and gaps analyses. Subsequent workshops will focus on specific R&D issues necessary for success.

Breadth-first frameworks: The next major effort will be the development of 2-3 frameworks, one for libraries and 1 or 2 specific application domains. Although programming models, libraries and fault-resilient capabilities will probably not be mature, this initial breadth-first approach will facility 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, and 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 start, utilizing the design decisions, and tool-chain established by the first frameworks.

Deployment: Finally, 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.

Time Frame	Targets and Milestones
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. • Develop library data model semantics. – Workshop 3: • App-driven resilience models.
2012-13	Develop first 2 app and first library frameworks, 2013. – Mine components from existing capabilities. – Implement common tool chain, programming model, first resilience harness, library interfaces. – Breadth first approach.
2014-15	Fully develop exascale-specific framework features: – Mature framework-library data layout semantics.

	– Fully capable fault resilience capabilities. – Fully-defined common toolchain.
2016-17	Develop 2-3 additional app frameworks, 2017. – Leverage infrastructure/design knowledge from first efforts. – Develop inter-component coupling capabilities (e.g., data sharing).
2018-19	Demonstrate full-scale application capabilities across all frameworks on Exascale system, 2019.

4.2.2.4 Crosscutting considerations

Framework efforts will be greatly impacted by evolving programming models, libraries and new algorithm development, as well as fault-resilient requirements and capabilities. Although it appears that MPI will likely be part of the picture, with a node programming model underneath, it is possible that a radical new programming and execution model will be needed. In all cases, a framework will be very 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.

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

4.2.3 Compilers

Contributors: Barbara Chapman (U. of Houston), Mitsuhisa Sato, (U. of Tsukuba, JP), Taisuke Boku (U.of Tsukuba, JP), Koh Hotta, (Fujitsu), Matthias Mueller (TU Dresden), Xuebin Chi (Chinese Academy of Sciences)

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 support the balancing of 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, and 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 Compilers

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. Advances in compiler technology are key to the provision of programming models that offer both performance and productivity characteristics. We can no longer afford to do without the benefit of compilers at run time. The following topics should be pursued:

- Techniques for the translation of new exascale programming models and languages supporting high productivity and performance, support for hybrid programming models and for 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.
- Exascale compilers could benefit from recent experiences with just-in-time compilation and perform online feedback-based optimizations, try out different optimizations, generate multiple code versions or perform more aggressive speculative optimizations. They will need to incorporate a variety of light-weight strategies for modifying code on the fly.
- Compilers will need to play a role in supporting strategies for enabling fault tolerance. For example, they may be able to help reduce the amount of data involved in checkpointing.
- Interactions between the compiler and the development and execution environment should be enabled using standard interfaces. 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 may be developed e.g. to support the application development process, to help interpret the impact of the compiler's translation on the application's runtime behavior, and to explain how the application developer might be able to improve the results of this translation.
- Compilers may be able to benefit from auto-tuning approaches, may incorporate techniques for learning from prior experiences, exploit knowledge on suitable optimization strategies that is gained from the development and execution environments, and apply novel techniques that complement traditional translation strategies.

Time Frame	Targets and Milestones
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 System-wide 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	Refine architecture awareness Compilers that interact smoothly with performance and runtime tools

4.2.3.4 Crosscutting 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 run time 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

Contributors: Jack Dongarra (U. of Tennessee), Bill Gropp (UIUC), Mike Heroux (SNL), Anne Trefethen (Oxford U., UK) Aad van der Steen (NCF, NL)

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 are dependent upon the formation of various standards that will be needed to insure the wide spread deployment of the software components. The libraries will be equally dependent upon the operating system as well as the computer architecture features and how they communicated to the library level.

4.2.4.2 Alternative R&D strategies for Libraries

In effect the alternate research and develop strategies for libraries will be driven by the operating system and software environment provided on given architectures. We can assume that we see models such as message passing libraries, global address space languages, and message driven work queues. As we can assume that all three models will occur at some level in future systems, this means that matching

implementations need to be developed concurrently. In particular the three programming model should be interoperable to permit the widest deployment.

4.2.4.3 Recommended Research Agenda Libraries

The existing numerical libraries will need to be rewritten and extended in the 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. Due to 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:

1. Hybrid and hierarchical based software: efficient implementations need to be aware of the underlying platform and memory hierarchy for optimal deployment.
2. Auto tuning: 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 w.r.t. the increased rate of faults in the data being processed.
4. Mixed arithmetic for performance and energy saving: 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 provided 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 are a requirement as communications plays such an important role in performance and scalability.
8. Algorithms for shared memory architectures have always been around but 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. Libraries often introduce artificial separations into the code, based on the function of each routine. Techniques that permit the fusion of library routine implementations (e.g., fusion of the loops in two consecutive library calls) will be needed.

Time Frame	Targets and Milestones
2010-12	Standards for hybrid (heterogeneous) computing are needed immediately. 2011: Milestone: Heterogeneous software libraries 2012: Milestone: Language issues
2012-14	Standards required Architectural characteristics agreed. 2013: Milestone: Architectural transparency
2014-16	2015: Milestone: Self adapting for performance Standards required for energy aware

2016-17	2016: Milestone: Energy aware Standard for fault tolerance required
2018-19	2018 Milestone: Fault tolerance 2019: Milestone: Scaling to billion way

4.2.4.4 Crosscutting considerations

Libraries will require agreed standards to build on. These will include standards for power management, architectural characteristics, programming for heterogeneous environments and fault tolerance. This presupposes that the information regarding the underlying architecture, energy usage, etc., 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 while the calling sequences for their routines should fit in with the various programming models that are available for exascale environments.

4.2.5 Debugging tools

Contributors: David Skinner (LBL), Wolfgang Nagel (Dresden, DE),

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 tera and peta scale has lead to profound challenges in the ability of a software debugger to encompass the entire parallel application consisting of thousands of processes. Additionally it has brought 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.

Looking towards 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 itself. As a concrete example, 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 aspects of exascale technology that will drive decisions in debugging are

- 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 these drivers will make debugging at exascale an increasingly costly endeavor both in terms of human effort applied to debugging as well as

diminishing the investment in HPC resources by requiring more machine hours to be devoted to costly debug sessions. We therefore propose a research strategy for exascale debugging which aims 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 system. 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 the research and development of improving the information sources from which a variety of debugging frameworks can benefit. This 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 which deliver that information in a portable manner. For example, PAPI provides a common interface for performance information upon which performance tools may be built.

In order 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.

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 is in contrast with the idea of using a debugger to step through executing code on an instruction or subroutines basis and moves in the direction of the debugging framework becoming advisory and participatory in the production execution of codes.

Secondly, 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 the end-to-end solution within each debugger we imagine a lowered barrier to entry to the design of special purpose custom fitted debuggers which 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 incorporate the handling of faults and problem scenarios internally. Currently there is a large mismatch 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 Debugging

The general thrust of this analysis is that debugging technology needs to grow away from monotlithic applications towards runtime libraries and layers that detect problems and aggregate highly concurrent debugging information into a categorical rather than task based context. Pursuing this path, however, raises a variety research challenges the solution to which 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, and 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 fullscale exascale debugging. Intelligent selection from a menu of reliable data sources will have to be able 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 just-in-time as problems unfold. Users will have to be able advise the application about objectives from which deviation is considered a bug. A debug framework wit these capabilities would enable the application to advise the user about situations indicative of problems, such as expanding memory footprint, incorrectness, sudden changes in performance.

By focusing on the ability of debugging frameworks to scale well 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.

4.2.5.4 Roadmap for Exascale Debugging

Time Frame	Targets and Milestones
2010-11	Planning &Workshops Lightweight debugging @ 1e5 cores
2012-13	Support for heterogenity in nodes
2014-15	Simulation @ 10^6 cores
2016-17	Software development to support 1e6 core production debug
2018-19	Near-production exascale

4.3 Applications

While IESP may not focus on developing applications per se, nevertheless they are the very 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 IESP Application Co-Design Vehicles

Contributors: Richard Kenway (University of Edinburgh, UK) and William Tang (Princeton U/PPPL)

Co-Design Vehicles (CDVs) are applications that provide targets for, and feedback to, the hardware and software 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 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 terascale application today with a demonstrated need for exascale performance;
2. In progressing to exascale, at least one milestone will be achieved that has significant scientific impact 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;
3. A realistic productive pathway to exascale can be mapped out over 10 years; and
4. The community developing the CDV application is experienced in algorithm, software and/or hardware developments and willing to engage in the exascale co-design process.

The IESP will identify a manageable number 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 applications 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. Finally, 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 "living 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 vs software components (Ref. -- Section being developed by Bill Kramer).

4.3.1.1 Representative CDVs

In order to provide some illustrative 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.

I. High Energy Physics/QCD

Simulations of QCD, the theory of the strong interaction between quarks and gluons which are the basic building blocks of hadrons, have played a pioneering role in the development of parallel and, latterly, high-performance computing since the early 1980's. Today, lattice QCD codes are amongst 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 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; i.e.,

1. 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.

2. 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, it will also enable experiment 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, peta/exascale simulations may help explain electroweak symmetry breaking.
3. The pathway to early exascale performance for QCD requires developing multi-layered 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 due to 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.
4. 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 US, the series of APE machines in Europe, and CP-PACS in Japan. Notably, design features of QCDOC influenced IBM's BlueGene. 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 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.

II. Plasma Physics/Fusion Energy Sciences (FES)

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 FES area meets the criteria for exascale co-design in that:

- FES applications currently utilize the LCF's at ORNL and ANL as well as advanced computing platforms at LBNL – demonstrating scalability of key physics with increased computing capability; 2. HPC FES topics with significant scientific impact were clearly identified at the

major DOE workshop on Grand Challenges in FES & Computing at the Extreme Scale (April, 2009); i.e., (a) high physics fidelity integration of multi-physics, multi-scale FES dynamics; and (b) burning plasmas/ITER physics simulation capability; and

- A productive FES pathway (over 10 years) can be readily developed for exploitation of exascale. This includes carrying out experimentally-validated confinement simulations (including turbulence-driven transport) and serves to demonstrate 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 & Gaps:
    - While FES applications for macroscopic stability, turbulent transport, edge physics (where atomic processes important), etc. have demonstrated at various levels of efficiency the capability of using existing LCF's, 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 solves) 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 (V&V) methodologies including uncertainty quantification (UQ), and the management of large data sets from experiments & simulations. As part of the aforementioned ongoing FES collaborations with LCF's, 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 PF (2012) integrated plasma core-edge coupled simulations
- 1 EF (2018) whole-system burning plasma simulations applicable to ITER

III. Other possible CDV's go here

…

The technology drivers for CDV applications are for the most part connected to advanced architectures with greater capability but with formidable software development challenges. It is expected that the need

to address concurrency issues and to deal with complex memory access/data movement challenges for emerging heterogeneous architectures with accelerators will drive new approaches for scalable algorithms and solvers. For risk mitigation purposes, alternative R&D strategies need to be developed for choosing architectural platform(s) capable of effectively addressing the PRDs in the various domain applications while exploiting the advances on the path to the exascale. Beneficial approaches include:

1. Developing effective collaborative alliances involving CS and Applied Math (e.g., following the SciDAC model);
2. Addressing crosscutting challenges shared by CDV applications areas via identification of possible common areas of software development, appropriate methodologies for V&V and UQ, and the common need for collaborative interdisciplinary training programs to deal with the critical task of attracting, training, and assimilating young talent.

In summary, the current applications identification exercise is intended to complement and provide input into the building of the IESP Roadmap -- a planning instrument designed to enable the international HPC community to improve, coordinate, and leverage their collective investments and development efforts.

4.3.2 Application Element: Algorithms

Contributors: Bill Gropp (UIUC), Fred Streitz (LLNL), Mike Heroux (SNL), Anne Trefethen (Oxford U., UK)

4.3.2.1 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 currency, latency, and load balancing. Because an Exascale system will have 108 to 109 threads, simply creating enough concurrency from an application can become a challenge (a 10003 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 a103-104 cycle hardware latency in accessing remote memory. Hiding this latency requires algorithms that achieve a computation/communication overlap of at least 104 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 (this includes Krylov methods for solving sparse linear systems). These synchronization points must be eliminated. Finally, 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 is another driver 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 role of algorithms in detecting faults. Note that detecting faults in hardware requires additional power, memory, etc. Regardless of who detects a fault, it 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 role of algorithms in repairing faults, particularly transient (e.g., memory upset) faults. In addition, on 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 are minimize power use need to be developed. Naturally, 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. Finally, 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 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 than 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.

4.3.2.2 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) the development of new algorithms.

In refining algorithms, there are a number of strategies that may be applied. For sc

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.

4.3.2.3 Recommended Research Agenda 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 list or 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 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 an 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

Time Frame	Targets and Milestones
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 approach (cross-cutting issue). Needs to develop and use more realistic models of computation (quantify need).
2012-13	Algorithms for intra-node scaling Algorithms for inter-node scaling Evaluation on petascale systems Better scaling in node count and within nodes can be performed using petascale systems in this time frame (so it makes sense to deliver a first pass in this time frame).
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. At the time of this bullet, view this 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 very 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.

4.3.2.4 Crosscutting considerations

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

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, e.g., load balance issues are very 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 the 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 auto-tuning will maximize the ROI on code development and thus lower the effective cost of entry into HPC.

4.3.3 Application Support: Data Analysis and Visualization

Contributors: Michael E. Papka (ANL), Pete Beckman (ANL), Mark Hereld (ANL), Rick Stevens (ANL), John Taylor(CSIRO, Australia)

4.3.3.1 Technology and Science drivers for Data Analysis and Visualization

Modern scientific instruments eg in Synchrotron science, high energy physics, astronomy, 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 eg 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 both the advancement of current technology, as well as 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.

4.3.3.2 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 may not provide adequate solutions in the long run.

It is inevitable that 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** – Identify new mathematical and statistical research needed for analysis of exabyte data sets
- **Integrated adaptive techniques** – Development of these would 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.
- **Meta tools** – 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.
- **Facilitate Collaboration** – Plan deployment of global system of large scale high resolution (100 Mpixel) visualization and data analysis systems based on open source architecture to link universities and research laboratories

4.3.3.3 Recommended Research Agenda 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 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 especially critical in achieving this goal are the following:

- Identification of features of interest in exabytes of data
- Visualisation of streams of exabytes of data from scientific instruments
- Integrating simulation, analysis and visualization at the 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, and beginning as soon as possible.

Time Frame	Targets and Milestones
2010-11	Planning &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 visualisation 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 & visualisation @ 10^5 cores with petabyte data sets Support for heterogenity in nodes
2014-15	Integrate data analysis and visualisation tools into domain specific workflows Achieve data analysis & visualisation @ 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 & visualisation @ 10^6 cores with near exascale data sets
2018-19	Roll out data analysis and visualisation at the exascale

4.3.3.4 Crosscutting considerations

Architecture at coarse and fine grain. Analysis and visualization can use any or all of the computational, storage, and network resources that comprise 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 development timetable for 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 grain execution of numerical algorithms comprising the simulation, ongoing read/write operations, and system level resource scheduling. We 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 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. This 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 maximising the productivity of researchers working on exascale systems.

4.3.4 Scientific Data Management

Contributors: Alok Choudhary (Northwestern U.), …

4.3.4.1 Technology and Science drivers for Scientific Data Management

Management, analysis, mining, and knowledge discovery from data sets of this scale is a very challenging problem, yet a critical one in Peta-scale systems and would be even more so for Exascale systems. Most science applications at this scale will be extremely data intensive, and 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. Furthermore, Data Management for observational data, analysis of observational data and its use in validating simulations would be extremely important. Individual simulation would potentially produce Petabytes+ of data due to scaling, 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. Effectively generating, managing, and analyzing this information requires a comprehensive, end-to-end approach to data management that encompasses all of the stages from the initial data acquisition to the final analysis of the data. Many common questions arise across various application disciplines. Are there data management tools available that can manage data at this scale? Although scalable file systems are important as un underlying technology, they are not suitable for user level mechanism for scientific data management. What are the scalable algorithms 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? Clearly, at exascale level, all of the above data management issues much be enabled by massively scalable I/O and storage capabilities which must be used as a basis for designs for the data management software. However, I/O systems drivers, requirements and research agenda is discussed in a separate section in this report.

4.3.4.2 Alternative R&D strategies for Scientific Data Management

Scientific Data Management covers many subfields from data formats, workflow tools, 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.

1. Data Analysis and Mining Software and Tools: Knowledge discovery from massive datasets produced and/or collected would 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, multi-dimensional (and large number of dimensions), hierarchical and multiscale data at massive scales. Scalable tools based on these algorithms with a capability to incorporate other algorithms must be developed. Traditionally, analytics and mining specification languages have been sequential in nature and are unable to scale to massive datasets. Parallel languages for analysis and mining that can scale to massive data sets would 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 would be important. Techniques for On-line analytics, active-storage and co-processing models should be developed which can run concurrently (potentially on a subsystem) with the simulations, that can exploit multicore nature of the systems, maximizing the use of data while it is available should be investigated.

2. 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 allowing 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 would be very important for exascale systems both for performance and productivity of scientists as well as 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 user histories etc. will simplify workflow design and increase dependability. As a part of workflow tools, creation, management, querying and use of data provenance must be investigated.

3. Extensions of Databases Systems: Commercial database systems such as those based on relational or object models, or derivation thereof have not proved to be suitable 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.

4. Deign 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 over-constraining usage models which are impediments to scalability in traditional databases. Scalable file systems would 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 needs to be optimized for limited functionality needed by data management layer, which in turn presents a 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.

5. 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. The changes in these in the past have mainly been driven by backward compatibility. Approaches to adapt these formats, enhance these formats and scaling the data access libraries must be investigated. Furthermore, new storage formats, that emphasize on scalability and the use of effective parallel I/O along with the capabilities to incorporate analytics and workflow mechanisms would be important to investigate and develop. 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.

6. Search and Query Tools: Effective searching and querying of scientific data is very critical. Technology 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 would be important. Users may be interested in querying specific events or 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.

7. 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 wide area integrated with scalable I/O, workflow tools, query and search techniques.

4.3.4.3 Recommended Research Agenda Scientific Data Management

The recommended research agenda for SDM systems is all items above except item 3.

Time Frame	Targets and Milestones
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

Crosscutting considerations

The Scientific Data Management clearly has crosscutting considerations with scalable storage and I/O, visualization techniques and tools, operating systems, fault-resiliency mechanisms, communication layer and to some extent with programming models.

4.4 Crosscutting Dimensions

4.4.1 Resilience

Contributors: Franck Cappello (INRIA, FR), Al Geist (ORNL) Sudip Dosanjh (SNL), Marc Snir (UIUC), Bill Gropp (UIUC), Sanjay Kale (UIUC), Bill Kramer (NCSA), Satoshi Matsuoka (TITECH), David Skinner (NERSC)

Since exascale systems are expected to have millions of processors and hundreds of millions of cores, resilience will be a necessity 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 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:

- Exponential increase in the number of sockets, cores, threads, disks and the memory size.

- 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 MTTF would be in the range of 1 minute.
- Silent soft errors will become significant and raise the issues of result and end-to-end data correctness
- New technologies such as Flash Mem (SSD), Phase-Change RAM and accelerators will both raise 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 MTBF of the overall system.

4.4.1.2 Gap analysis:

- Existing fault tolerance techniques (global checkpoint/global restart) and will be unpractical at 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 the killing of the entire application.
- Present Applications and system software are not 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 scale simulations
- There are no standard metrics, no standardized experimental methodology nor 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:

1. Global recovery versus fault confinement and local recovery,
2. Fault recovery versus fault avoidance (fault prediction + migration),
3. Transparent (system managed) versus Application directed,
4. 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:

- Extent 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, dynamic error handling by applications

- 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, fault oblivious applications

Time Frame	Targets and Milestones
2010-12	Target1: Extend 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: Demonstrate replication as alternative to rollback
2013-15	Target1: Extend the applicability of Rollback Recovery Milestone: Integrate Phase Change RAM technologies Milestone: Error and fault confinement, Local recovery, TMR (cores) Milestone: Fault aware system software Milestone: Language support & paradigm for Resilience Milestone: Application and System software Interactions (standard API) Milestone: Consistency across layers (CIFTS or CIFTS like mechanisms) Target2: Fault avoidance & oblivious software -Milestone: RAS collection and analysis (root cause), situational awareness -Milestone: H&S Integration
2016-19	Target2: Fault avoidance & 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

Contributors: John Shalf (LBNL), Satoshi Matsuoka (TITECH, JP)

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. 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 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 direct 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. However, in this thermally limited case you can deliver higher memory bandwidth to the application for the 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 savings of $1M/year 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, are 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. So the direct impact of improved power management technology is to reduce the operating cost for Exascale HPC systems, but the broader 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, make control decisions that coordinate power management decisions across the system, and reduced models of code performance for optimal control are not well developed. Furthermore, the interfaces for representing sensor data for the control system, interfaces to describe policies to the control system, and to distribute control decisions are not available at scale. Effective system-wide power management will require development of interface standards to enable both vertical (e.g. between local components and integrated system) and horizontal integration (e.g. between numerical libraries) of components in a complete solution. 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 of this include Dynamic Voltage-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 that 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 primarily derived from consumer technology, where the power savings decisions are all made locally. For a large parallel system, locally optimal solutions can be tremendously non-optimal at the system scale. When nodes go into low-power modes opportunistically based on local decisions, it creates a jitter that can substantially reduce system-scale performance. For this reason, 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 the latency for changing between different power modes. So 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 complete 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 currently exist. There is no common expression of policy or objective. There is no comprehensive monitoring or data aggregation. More importantly, there is almost NO tool support for integration of power management into libraries and application codes. Without substantial investments to create system-wide 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 system-wide 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 enabled 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 require 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 OS's must also manage new power efficient architecutres, heterogeneous computing resources, including devices such as GPUs, embedded CPUs, non-volatile low power memory and storage, and manage data movement and locality in memory hierarchy.

System-Scale Resource Management: We need to develop power Performance monitoring and aggregation that scales to 1B+ 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, so 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. This requires standards 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. This includes analytical power models of system response and empirical models based on advanced learning theory. We must also develop scalable control algorithms to bridge gap between global and local models. Systems to aggregate sensor data from across the system (scalable data assimilation and reduction), 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. However, the optimal trade-off is very context specific, so we must enable libraries to be annotated for 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 performance/energy efficiency models and power management interfaces in software libraries to be standardized. This ensures compatibility of the management interfaces and policy coordination across different libraries (horizontal integration) as well as supporting 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 non-library code. A more advanced research topic would be 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 discovered that enable predictive control of application execution, so that we can find ways of reducing power without having 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 that are developed by different research groups and to enable libraries from different 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**: There must be able to integrate information from OS, system level resource manager, and applications/libraries for unified strategy to meet objectives.

Time Frame	Targets and Milestones
2010-11	Energy Monitoring Interface Standards Energy aware/communication avoiding algorithms *Should we enumerate specific deliverables in crosscut areas for each epoch?* ▪ 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 Enable libraries to be annotated for parameterized model of energy to articulate a policy to manage those 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 1B+ way parallel adaptive control decision capability

4.4.3 Performance Optimization

Contributors: Bernd Mohr (Juelich, DE), Adolfy Hoisie (LANL), Matthias Mueller (TU Dresden, DE), Wolfgang Nagel (Dresden, DE), David Skinner (LBL) Jeffrey Vetter (ORNL)

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 like statistical profiling, techniques like automatic or automated analysis, advanced filtering techniques, 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 functionally correct but poorly performing application after the fact.

Finally, in addition to user-controlled analysis and tuning, especially on higher level (inter-node) components of the X-stack, self-monitoring, self-tuning frameworks, middle ware, 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 very likely will need fundamentally different approaches to performance optimization.

In the performance modeling area, we anticipate that in new methodologies will need to be developed that go beyond the 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 Performance Optimization

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

- Continue development of scalable performance measurement, collection, and 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 / autotuning.
- Reliable and accurate performance analysis in presence of noise, system adaptation, and faults requires inclusion of appropriate statistical descriptions.
- Performance optimization for other metrics than time (e.g. power).
- Hardware and software components need to provide performance observability and control through appropriate interfaces and mechanisms (e.g., counters) 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).
- Programming models should be designed with performance analysis in mind. Software and runtime systems must expose their model of execution and adaptation, and its corresponding performance through a (standardized) control mechanism in the runtime system.

Time Frame	Targets and Milestones
2012-13	• Support for hybrid programming models (mixing MPI, PGAS, OpenMP and other threading models, accelerator interfaces) • Support modeling, measurement, and analysis, and autotuning on/for heterogeneous hardware platforms
2014-15	• Handle observation of million-way concurrency • Predictive exascale system design
2016-17	• Handle 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 Crosscutting considerations

To ensure performance analysis and optimization at exascale, the various components and layers of the X-stack have to be designed to be transparent with respect to performance. This *performance intransparency* will result in escalation of unforeseen problems to higher layers, including the application. This is not a really 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 which has to be addressed by appropriate mechanisms and tools.
- Dynamic power management, e.g., at hardware level inside a CPU, will result in performance variability between cores and across different runs. The alternative to run 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 makes 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.

Altogether this will require a integrated and collaborative approach to handle performance issues and correctly detect and analyze performance problems.

4.4.4 Programmability

Contributors: Thomas Sterling (LSU), Hiroshi Nakashima (Kyoto U., JP)

Programmability is the crosscutting property that reflects the ease by which application programs may be constructed. Although quantitative metrics are uncertain (e.g., SLOC) in their effectiveness, qualitatively 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 including 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 that 10 billion-way operation concurrency will be required.
- Distributed Resource Allocation and Locality Management – balancing the tension between spreading the work among enough execution resources for parallel execution and co-locating tasks and data to minimize latency is required to make such systems programmable.
- 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, accelerators, and other characteristics must be managed in a way that circumvents their negative impact while exploiting their potential opportunities without demanding explicit user control, making programming much more difficult.
- 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 over constraining control methods must be eliminated and replaced by lightweight synchronization overlapping phases of computation.
- Data Structure Representation – and distribution.

4.4.4.1 Technology and Science drivers for Programmability

As a crosscutting property of future Exascale systems programmability is directly impacted by all layers of the system stack that constitute the technology and science drivers. 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 will assist 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. While not part of the software system, the architecture directly impacts 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 alternative 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. It is anticipated that the evolutionary strategy will be pursued as part of the efforts of the community to extend common practices as far in to 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 it 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 anyway 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 towards 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. The common scalable execution model for STEM application targeted systems is CSP which is now unduly stressed in support of present day multi/many-core heterogeneous systems and cannot, in its current form, be expected to achieve the required functionality for scalability, efficiency, and dynamic scheduling. Therefore, it is recommended that research be conducted to devise a new over-arching execution model either as a dramatic extension of current practices or (as is expected by some) an entirely new (likely based in part on experimental prior art) model of computation explicitly derived to address the unique challenges of Exascale computing. Such a model of computation will have strong impact on programmability, one of the strategic requirements.

New Programming Models and Methods – Research of new programming models and ultimately API, tools, and methods will be required to provide the user interface to construct new application (and system software) programs and to determine what 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. Included in future models needs to be 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 best use of them from a programmer perspective.

New Runtime Systems – Research for advanced runtime systems will be an important means of dramatically improving programmability supporting dynamic software behavior like load balancing, thread scheduling, processing and memory resource allocation, power management, and recovery from failures. Only runtime systems (and OS 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 will be particularly true for systems of up towards a billion cores and constantly changing system configurations. In particular, new runtime software will move most programming practices from the static methodology to dynamic adaptive techniques exploiting runtime information for improved performance optimization. Examples include the user lightweight thread scheduling, context switching, and suspension management, as well as inter-thread 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 further 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, taking advantage of more sophisticated messaging methods (e.g., message-driven mechanisms), and involving new forms of active Global Address Space (GAS) and its management.

X-gen Architectures – Although the actual development of future Exascale system hardware architectures is beyond the scope of the IESP program agenda, research towards critical systems software and programming methods will be sensitive to and have to respond to the emergence of new architectures needed 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 for 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 providing many services to that layer. A new operating system will be essential for the X-gen architectures and it 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 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 inter-job 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-ware resource management techniques can be more effectively developed and applied for fail-safe power-efficient scalable operation.

4.4.4.4 Crosscutting considerations

Programmability is a crosscutting factor affected by all layers of the system stack including software and hardware. It also is interrelated with other crosscutting factors such as performance and potentially resilience. It is not clear if there is a relationship between programmability and power management although when writing system software, the need to develop power management software for the operating system and possibly the runtime system is certain.

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. It is anticipated that these components will be important even more so for future Exascale systems. The complexity in that extreme case will require that the responsibility for all but parallelism (and even not all of that) be removed from the programmer and handled by the combination of compiler and runtime in cooperation with the operating system and system architecture.

With respect to reliability, it may be of value for the programmer to have the option to dictate 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 to minimize programmer intervention and therefore improve programmability.

4.4.5 Matrix of applications and software components needs

Science and Engineering Disciplines	Sub Areas	New Programming Models and New ways to specify [illegible]	Programmability - Improved Code Development and Application Building [illegible]	Resource management, power management and	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
Material 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
Earth Systems	• Climate • Weather • Earthquake/ Seismic • Subsurface Transport • Water	X	X	X	X	X	X	X		X	X

	Resources										
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

5. Appexdix IESP Attendees

				SC08	Santa Fe	Paris	Japan	
Giovanni	Aloisio	Euro-Mediterranean Centre for Climate Change	Italy			x	x	giovanni.aloisio@unile.it
Patrick	Aerts	NWO	NL		x	x	x	aerts@nwo.nl
Dong	Ahn	LLNL	US	x				ahn1@llnl.gov
Yutaka	Akiyama	Tokyo Tech	Japan				x	akiyama@cs.titech.ac.jp
Jean-Claude	Andre	CERFACS	France			x	x	Jean-Claude.Andre@cerfacs.fr
Phil	Andrews	UT	US	x				pandrew2@mail.tennessee.edu
Mutsumi	Aoyagi	U Kyushu	Japan			x	x	aoyagi@cc.kyushu-u.ac.jp
Mike	Ashworth	Daresbury	UK			x		mike.ashworth@stfc.ac.uk
Franck	Barbier	ANR	France			x		franck.barbier@agencerecherche.fr
David	Barkai	Intel	US		x	x	x	david.barkai@intel.com
Sanzio	Bassini	CINECA	Italy			x		bassini@cineca.it
Kyriakos	Baxevanidis	EU	EU			x		Kyriakos.Baxevanidis@ec.europa.eu
Pete	Beckman	ANL	US	x	x	x	x	beckman@mcs.anl.gov
Jean-Yves	Berthou	EDF	France	x	x	x	x	jy.berthou@edf.fr
Richard	Blake	Daresbury	UK		x			r.j.blake@dl.ac.uk
Jay	Boisseau	TACC	US	x				boisseau@tacc.utexas.edu
Taisuke	Boku	U of Tsukuba	Japan		x	x	x	Taisuke@cs.tsukuba.ac.jp
Bertrand	Braunschweig	ANR	France		x	x	x	Bertrand.BRAUNSCHWEIG@agencerecherche.fr
Bill	Camp	Intel	US		x			william.j.camp@intel.com
Franck	Cappello	INRIA	France	x	x	x	x	fci@lri.fr
Barbara	Chapman	U of Houston	US		x	x	x	bmchapman@earthlink.net
Xuebin	Chi	CAS	China				x	chi@sccas.cn
Alok	Choudhary	NWU	US		x	x	x	alok.choudhary@eecs.northwestern.edu
Iris	Christadler	LRZ	Germany			x		christadler@lrz.de
Almadena	Chtchelkanova	NSF	US		x			achtchel@nsf.gov
Guillaume	Colin de Verdière	CEA	France			x		guillaume.colin-de-verdiere@CEA.FR
Frederica	Darema	NSF	US		x			fdarema@nsf.gov
Bronis	de Supinski	LLNL	US	x				desupinski1@llnl.gov
David	Dean	ORNL/DOE	US				x	deandj@ornl.gov
Jack	Dongarra	U of Tennessee	US	x	x	x	x	dongarra@cs.utk.edu
Sudip	Dosanjh	SNL	US		x	x	x	sudip@sandia.gov
Thom	Dunning	NCSA	US	x				tdunning@ncsa.uiuc.edu
Hugo	Falter	ParTec	Germany		x	x	x	falter@par-tec.com
Fabrizio	Gagliardi	Microsoft	US			x		Fabrizio.Gagliardi@microsoft.com
Alan	Gara	IBM	US		x	x		alangara@us.ibm.com
Al	Geist	ORNL	US		x			gst@ornl.gov
Luc	Giraud	CERFACS	France		x	x		Luc.Giraud@cerfacs.fr
Kostas	Glinos	EU	EU			x		konstantinos.glinos@ec.europa.eu
Jean	Gonnord	CEA	France	x		x		jean.gonnord@cea.fr
Robert	Graybill	ISI	US		x			graybill@east.isi.edu
Bill	Gropp	UIUC	US	x		x	x	wgropp@illinois.edu
Jim	Hack	ORNL	US	x				jhack@ornl.gov
Jean-Francois	Hamelin	EDF	France		x		x	jean-francois.hamelin@edf.fr
Robert	Harrison	ORNL	US				x	harrisonrj@ornl.gov

Bill	Harrod	Darpa	US	x				William.Harrod@darpa.mil
Stefan	Heinzel	Max Planck DEISA	Germany		x	x	x	heinzel@dkrz.de
Barb	Helland	OS	US	x				helland@ascr.doe.gov
Mike	Heroux	Sandia	US	x	x	x	x	maherou@sandia.gov
Ryutaro	Himeno	RIKEN	Japan		x		x	himeno@riken.jp
Kimihiko	Hirao	Riken	Japan				x	hirao@riken.jp
Dan	Hitchcock	OS	US	x				Daniel.Hitchcock@science.doe.gov
Thuc	Hoang	NNSA	US		x			Thuc.Hoang@ns.doe.gov
Adolfy	Hoisie	LANL	US				x	hoisie@lanl.gov
Charlie	Holland	DARPA	US	x				Charles.Holland@darpa.mil
Koh	Hotta	Fujitsu	Japan				x	hotta@jp.fujitsu.com
Yuichi	Inoue	MEXT	Japan				x	yinoue@mext.go.jp
Yutaka	Ishikawa	U of Tokyo	Japan		x	x	x	ishikawa@is.s.u-tokyo.ac.jp
Satoshi	Itoh	MEXT	Japan				x	satoshi.itoh@toshiba.co.jp
William	Jalby	U of Versailles	France		x			William.Jalby@prism.uvsq.fr
Jean-Pascal	Jégu	Teratec	France			x		jean-pascal.jegu@teratec.fr,
Zhong	Jin	CAS	China				x	zjin@sccas.cn
Fred	Johnson	DOE	US	x		x	x	fjohnson@mreg.com
Andrew	Jones	NAG	UK			x	x	Andrew.Jones@nag.co.uk
Laxmilkant	Kale	UIUC	US			x		kale@uiuc.edu
Richard	Kenway	EPCC	UK		x		x	r.kenway@epcc.ed.ac.uk
David	Keyes	Columbia U.	US		x	x		david.keyes@columbia.edu
Moe	Khaleel	PPNL	US				x	moe.khaleel@pnl.gov
Kimmo	Koski	CSC	Finland			x		Kimmo.Koski@csc.fi
Bill	Kramer	NCSA	US		x	x	x	wkramer@ncsa.uiuc.edu
Dimitri	Kusnezov	NNSA	US	x				Dimitri.Kusnezov@nnsa.doe.gov
Jesus	Labarta	BSC	Spain		x	x	x	jesus.labarta@bsc.es
Jean-Francois	Lavignon	Bull	France		x	x	x	jean-francois.lavignon@bull.net
Alain	Lichnewsky	Genci	France		x	x		lichnewsky@genci.fr
Volker	Lindenstruth	Heidelberg U	Germany			x		ti@kip.uni-heidelberg.de
Thomas	Lippert	Juelich	Germany	x		x	x	th.lippert@fz-juelich.de
Bob	Lucas	ISI	US		x	x		rflucas@ISI.EDU
Barney	Maccabe	ORNL	US		x	x	x	maccabeab@ornl.gov
Satoshi	Matsuoka	TiTech	Japan	x	x	x	x	matsu@is.titech.ac.jp
Bob	Meisner	NNAS	US	x				Bob.Meisner@nnsa.doe.gov
Paul	Messina	ANL	US	x		x	x	messina@mcs.anl.gov
Peter	Michielse	NWO	NL		x	x		michielse@NWO.NL
Kazunori	Mikami	Cray	Japan				x	mikami@cray.com
Leighanne	Mills	U of Tennessee	US				x	mills@eecs.utk.edu
Bernd	Mohr	Juelich	Germany		x	x	x	b.mohr@fz-juelich.de
Terry	Moore	U of Tennessee	US	x	x	x	x	tmoore@cs.utk.edu
Hervé	Mouren	Teratec	France			x		Hervé Mouren (h.mouren@noos.fr)
Jean-Michel	Muller	CNRS	France			x		Jean-Michel.Muller@ens-lyon.fr
Matthias	Müller	Dresden	Germany				x	matthias.mueller@tu-dresden.de
Wolfgang	Nagel	Dresden	Germany		x	x	x	wolfgang.nagel@tu-dresden.de
Kengo	Nakajima	U of Tokyo	Japan				x	nakajima@cc.u-tokyo.ac.jp
Hiroshi	Nakashima	Kyoto U.	Japan			x	x	h.nakashima@media.kyoto-u.ac.jp
Mamoru	Nakono	Cray	Japan				X	nakano@cray.com
Jeff	Nichols	ORNL	US		x		x	nicholsja@ornl.gov

Jane	Nicholson	EPSRC	UK				x	Jane.Nicholson@epsrc.ac.uk
Jean-Philippe	Nominé	CEA	France			x		Jean-Philippe.NOMINE@CEA.FR
Nick	Nystrom	PSC	US		x			nystrom@psc.edu
Per	Oster	CSC	Finland	x	x			Per.Oster@csc.fi
Abani	Patra	NSF	US		x	x	x	apatra@nsf.gov
Rob	Pennington	NSF	US		x			rpenning@nsf.gov
Serge	Petiton	CNRS	France			x		Serge.Petiton@lifl.fr
Claude	Puech	INRIA	France		x	x		Claude.puech@inria.fr
Tracy	Rafferty	U of Tennessee	US	x	x	x	x	rafferty@cs.utk.edu
Dan	Reed	Microsoft	US		x	x		Daniel.Reed@microsoft.com
Michael	Resch	HLRS Stuttgart	Germany		x			resch@hlrs.de
Catherine	Rivière	GENCI	France		x	x		catherine.riviere@genci.fr
Ralph	Roskies	PSC	US	x				roskies@psc.edu
Faith	Ruppert	ANL	US		x			ruppert@alcf.anl.gov
Christian	Saguez	Teratec	France			x		Christian.Saguez@ecp.fr
Vivek	Sarkar	Rice	US		x			vsarkar@rice.edu
Mitsuhisa	Sato	U of Tsukuba	Japan	x	x	x	x	msato@cs.tsukuba.ac.jp
Stephen	Scott	ORNL	US		x			scottsl@ornl.gov
Mark	Seager	LLNL	US		x			seager@llnl.gov
Ed	Seidel	NSF	US	x		x	x	hseidel@nsf.gov
Akiyuki	Seki	MEXT	Japan				x	a-seki@mext.go.jp
Satoshi	Sekiguchi	AIST/METI	Japan				x	s.sekiguchi@aist.go.jp
Hideo	Sekino	Toyohash Inst Tech	Japan				x	sekinoh@gmail.com
John	Shalf	LBNL	US			x	x	JShalf@lbl.gov
Horst	Simon	LBNL	US	x	x			simon@nersc.gov
David	Skinner	LBNL	US		x	x	x	DESkinner@lbl.gov
Marc	Snir	UIUC	US	x				snir@illinois.edu
Mary	Spada	ANL	US	x				ms@digitale-inc.com
Thomas	Sterling	LSU	US		x	x	x	tron@cct.lsu.edu
Rick	Stevens	ANL	US	x	x		x	stevens@anl.gov
Michael	Strayer	DOE OS	US	x	x			michael.strayer@science.doe.gov
Fred	Streitz	LLNL	US				x	streitz1@llnl.gov
Bob	Sugar	UCSB	US				x	sugar@savar.physics.ucsb.edu
Shinji	Sumimoto	Fujitsu	Japan				x	s-sumi@flab.fujitsu.co.jp
Makoto	Taiji	Riken	Japan			x	x	taiji@riken.jp
Toshikazu	Takada	Riken	Japan				x	tz-takada@riken.jp
Bill	Tang	PPPL	US				x	tang@pppl.gov
John	Taylor	CSIRO	AU				x	John.A.Taylor@csiro.au
Rajeev	Thakur	ANL	US				x	thakur@mcs.anl.gov
Anne	Trefethen	Oxford	UK		x	x	x	anne.trefethen@oerc.ox.ac.uk
Akira	Ukawa	U of Tsukuba	Japan				x	ukawa@ccs.tsukuba.ac.jp
Mateo	Valero	BSC	Spain	x		x		mateo.valero@bsc.es
Aad	van der Steen	NCF	NL				x	steen@hpcresearch.nl
Jeffrey	Vetter	ORNL	US		x	x	x	vetter@ornl.gov
Vladimir	Voevodin	Moscow State U	Russia			x		voevodin@parallel.ru
Andy	White	LANL	US	x	x			abw@lanl.gov
Peg	Williams	Cray	US		x	x	x	pegwms@cray.com
Robert	Wisniewski	IBM	US				x	bobww@us.ibm.com
Kathy	Yelick	LBNL	US	x	x	x		yelick@eecs.berkeley.edu

Akinori	Yonezawa	U Tokyo	Japan			x	yonezawa@is.s.u-tokyo.ac.jp
Thomas	Zacharia	ORNL	US	x			zachariat@ornl.gov

6. Appendix - Computational Challenges and Needs for Academic and Industrial Applications Communities

The IESP **Application subgroup** was given two main objectives: establish a roadmap to Exascale for scientific domains and document software issues (type of issues, time frame).

The Application Subgroup identified the application domains to be considered, listed the scientific and technical questions raised by Exascale simulation and finally established a list of experts in US, Japan and Europe that could provide inputs between the Paris and the Tsukuba meetings. The application domains identified were:
- Weather, Climate and Earth Sciences,
- Astrophysics, HEP and Plasma Physics,
- Materials Science, Chemistry and Nanoscience,
- Life Sciences,
- Engineering, Finance and Optimization.

A contact person has been identified for each expert among the member of the IESP Application Subgroup and was in charge of interviewing the experts, addressing the following issues:
- Scientific and computational challenges: brief overview of the underlying scientific and computational challenges and potential impact,
- Software issues – 2009: brief overview of identified software issues for addressing state of the art machines,
- Software issues – 2012, 2015, 2020: expected scientific and technical hurdles,
- Expert feedback: identification of the impact of the last machine change the expert has been face to on the applications (porting, optimization, re-writing, …) and ways of doing simulation and expected impact of the next machine change (going from Tflops to Pflops, Pflop to Eflops).

Twenty contributions have been received before the Tsukuba meeting, eleven more just after: five contributions in "Weather, Climate and Earth Sciences", two in "Astrophysics, HEP and Plasma Physics", seven in "Materials Science, Chemistry and Nanoscience", five in "Life Sciences" and twelve in "Engineering and Finance & Optimization".

The expert contributions have been first briefly presented to the Tsukuba Application Subgroup. Following this presentation and discussions between the participants, four technical transversal items were identified of particular importance for addressing Exascale computing in the different applications domains. The first one, "**Validation – verification - uncertainty quantification",** proposed to address the comparison of simulation results with experiment, the evaluation of how realistic is a simulation and how software tools can help that (i.e. visualisation). The second one, "**Mathematical methods",** focused on algorithms and solvers. The third one, "**Productivity and efficiency of code production",** dealt with load-balancing, scalability, tools for code development (debugging, performance analysis), programming model for actual and next computer generation and use of scientific libraries. The fourth one, "**Integrated framework**", addressed **the** integration and interoperability of multi-code/models/scales, interoperability of CAE, computation and visualisation and the management and supervision of workflows.

The Tsukuba Application Subgroup then divided into four working groups, addressing these four items. The conclusions of the four working groups once presented, the Application Groups divided into five disciplinary working groups in order to proceed to a classification of issues with respect to expectation from the SW groups.

These different contributions are summarized in the attached slides.