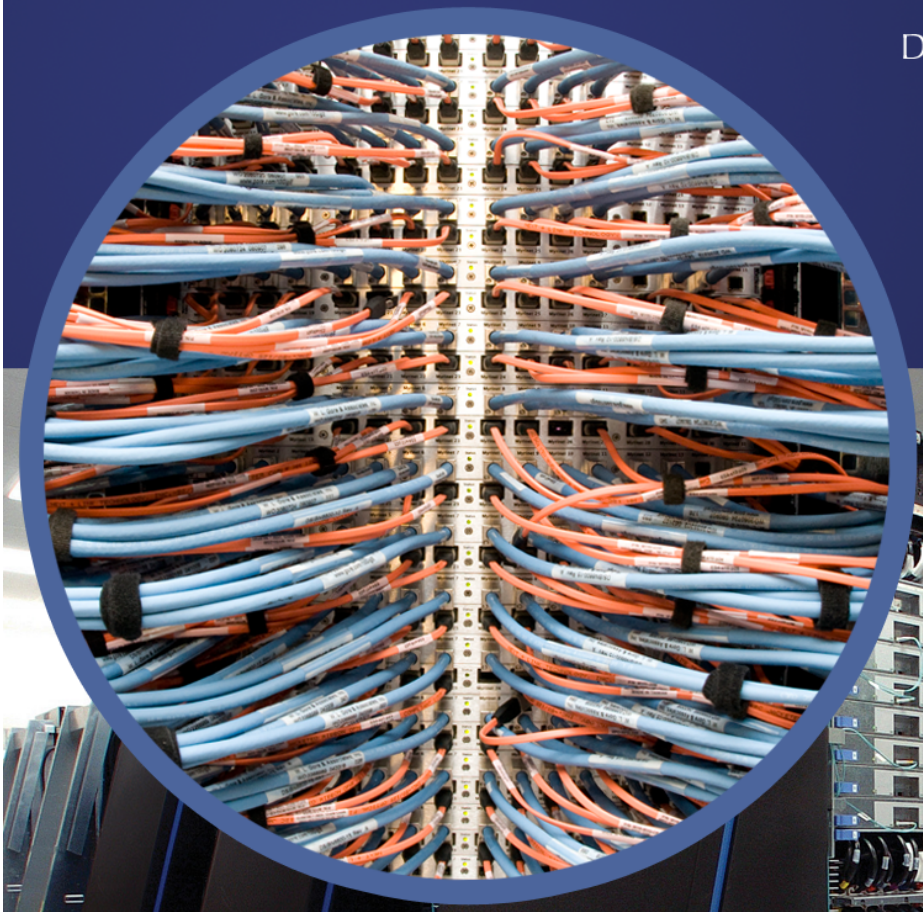
Scientific Grand Challenges

Architectures and Technology for Extreme Scale Computing

December 8-10, 2009 | San Diego, CA



U.S. DEPARTMENT OF ENERGY

Sponsored by:

Office of Advanced Scientific Computing Research, Office of Science

Office of Advanced Simulation and Computing, National Nuclear Security Administration

DISCLAIMER

On the cover: (background image) Argonne National Laboratory's IBM Blue Gene/P.

ARCHITECTURES AND TECHNOLOGY FOR EXTREME SCALE COMPUTING

Report from the Workshop held December 8-10, 2009

Sponsored by the U.S. Department of Energy Office of Advanced Scientific Computing Research and Office of Advanced Simulation and Computing, National Nuclear Security Administration

WORKSHOP PURPOSE

The purpose of this workshop was to identify and address the architectural and technology challenges to building exascale computers by 2018 that will meet the needs of next-generation scientific and engineering applications. Participants from the computer industry, national laboratories, and academia with expertise in architectures, technology, software, applied mathematics, and applications discussed public versions of vendor roadmaps in order to identify gaps and barriers to achieving usable and affordable exascale systems.

Workshop Chairs:

Rick Stevens, Argonne National Laboratory
Andrew White, Los Alamos National Laboratory

Workshop Session Leads:

Fault Management and Resiliency
Sudip Dosanjh, Sandia National Laboratories
Al Geist, Oak Ridge National Laboratory

Programming Models and Environments
Brent Gorda, Lawrence Livermore National Laboratory
Kathy Yelick, Lawrence Berkeley National Laboratory

Node Architecture and Power Reduction Strategies
John Morrison, Los Alamos National Laboratory
Horst Simon, Lawrence Berkeley National Laboratory
John Shalf, Lawrence Berkeley National Laboratory

Scalability and Concurrency
Jeff Nichols, Oak Ridge National Laboratory
Mark Seager, Lawrence Livermore National Laboratory

Sponsor Representatives:

Office of Advanced Scientific Computing Research, Office of Science, **Daniel A. Hitchcock**
Office of Advanced Scientific Computing Research, Office of Science, **Barbara Helland**

Office of Advanced Simulation and Computing, National Nuclear Security Administration, **Thuc T. Hoang**
Office of Advanced Simulation and Computing, National Nuclear Security Administration, **Sander Lee**

CONTENTS

EXECUTIVE SUMMARY

Achieving a thousand-fold increase in supercomputing technology to reach exascale computing (10^{18} operations per second) in this decade will revolutionize the way supercomputers are used. Predictive computer simulations will play a critical role in achieving energy security, developing climate change mitigation strategies, lowering CO_2 emissions, and ensuring a safe and reliable 21st century nuclear stockpile. Scientific discovery, national competitiveness, homeland security and quality of life issues will also greatly benefit from the next leap in supercomputing technology.

This dramatic increase in computing power will be driven by a rapid escalation in the parallelism incorporated in microprocessors. The transition from massively parallel architectures to hierarchical systems (hundreds of processor cores per central processing unit [CPU] chip) will be as profound and challenging as the change from vector architectures to massively parallel computers that occurred in the early 1990s. Without U.S. Department of Energy (DOE) leadership, the chasm between peak speed and sustained performance will grow exponentially, and the societal benefits of advances in component technologies will be delayed and greatly diminished. With DOE leadership of a collaborative effort among the national laboratories and key university and industrial partners, the architectural bottlenecks that limit supercomputer scalability and performance can be overcome. In addition, this effort will help make petascale computing pervasive by lowering the costs for these systems and dramatically improving their power efficiency.

The DOE's strategy for reaching exascale includes: 1.) Ensuring that a broad spectrum of DOE mission critical applications are full partners in the initiative; 2.) Collaborations with the computer industry to identify gaps; 3.) Prioritizing research based on return on investment and risk assessment; 4.) Leveraging existing industry and government investments and extending technology in strategic technology focus areas; 5.) Building sustainable infrastructure with broad market support i.e., extending beyond natural evolution of commodity hardware to create new markets, and creating system building blocks that offer superior price, performance, and programmability at all scales (exascale, departmental, and embedded); and 6.) Co-designing the hardware, system software and applications.

The last element, co-design, is a particularly important area of emphasis. Applications and system software will need to change as much as architectures over the next decade. This represents an unprecedented opportunity for the applications and algorithms community to influence future computer architectures. A new co-design methodology is needed to make sure that exascale applications will work effectively on exascale supercomputers.

The DOE's Exascale Initiative Steering Committee met with computer companies on three separate occasions. These meetings led to the identification of the following areas for emphasis.

Node architectures and power

Reducing power requirements and increasing energy efficiency is a critical issue. Some power projections for exascale supercomputers are over 100 megawatts (MW), making such a system impractical (typical energy costs are $1 million per MW). Throughout this discussion, 20 MW is taken as an upper limit on system power for an exascale system.

Node architectures are expected to change dramatically in the next decade, becoming more hierarchical and heterogeneous. Here a node is a compute node, an end point of the interconnect network. For simplicity, all compute nodes are assumed to be the same. Power and cooling

constraints are limiting increases in microprocessor clock speeds. Consequently computer companies are dramatically increasing on-chip parallelism to improve performance. The traditional doubling of clock speeds every 18 to 24 months is being replaced by a doubling of cores or other parallelism mechanisms. During the next decade the amount of parallelism on a single microprocessor will rival the number of nodes in the first massively parallel supercomputers that were built in the 1980s. In addition to reducing power requirements, there needs to be much tighter integration between the microprocessor, the memory, and the interconnect.

Our discussion of the possibilities for nodes diverged into two "swim lanes." Swim lane 1 has a node consisting of lightweight cores with 1,000 floating-point units (FPUs) and a nominal performance of 1 TF/s in the 2018 time frame. Thus, this swim lane would require more than one million nodes to achieve an exaflop/s. Swim lane 2 has a node consisting of aggressive cores with 10,000 FPUs and a nominal performance of 10 TF/s in the 2018 time frame. Thus, this swim lane would require more than one hundred thousand nodes to achieve an exaflop/s.

Scalability

The scalability of systems, systems software, and applications is a significant issue for exascale computing. Exascale systems will pose unprecedented challenges in parallelism. Systems will consist of one hundred thousand to one million nodes and perhaps as many as a billion cores. Managing and servicing a system of this size will be a challenge. Highly reliable and scalable operating systems and systems software will be needed. Applications must manage at least a thousand-fold increase in the available parallelism, which is certainly a challenge, but also an opportunity for new models and algorithms.

Reliability

Reliability is a significant concern as the number of processors increases and nodes become more complex. Other factors driving up the rate of faults include smaller circuits running at lower voltages, increased likelihood of low probability events, and the increasing use of heterogeneity. The exascale system reliability target is a system with one day between applications level interrupts. This will require development of a fault model, which will in turn enable co-designed advances in hardware and software reliability as well as new methods for application resilience. It is anticipated that there will be an additional layer of memory in future systems, non-volatile RAM (NVRAM). Data transfer speeds to NVRAM will be much higher than traditional disk systems. This NVRAM will provide the potential for input/output (I/O) caching and local recovery. In addition, methods for application migration are needed and research into fault-oblivious algorithms is recommended.

In what follows, we will classify errors as permanent or transient. Permanent errors, also referred to as hard errors, are repetitive, but not necessarily frequent, and require repair or replacement of hardware or software to return the system to normal operation. Transient errors, also referred to as soft errors, are not repetitive and do not require replacement or repair to return to normal operation, however their effect may be the same as a permanent error, e.g., node crash.

Errors are detected or undetected, also referred to as silent, errors. The effects and frequency of silent errors is a subject of considerable discussion in the community. Detected errors may be handled by hardware (e.g., ECC), by software (e.g., retry) and by applications (e.g., checkpoint/restart).

Programming Models
The principal programming environment challenges will be on the exascale node: concurrency, hierarchy and heterogeneity. The effects of these challenges will include a focus on asynchronous algorithms and moving away from a bulk synchronous programming model, a reliance on multi-threading and moving away from out-of-order execution to hide latency, a requirement for managing vertical, on node, data locality moving away from hardware managed caches, and more than a billion-way parallelism to fully utilize an exascale system. Thus, applications will need to change to achieve high performance, and to manage locality and resiliency on exascale systems. Portability will be a significant concern. Experience has show that application groups will not develop software for next-generation's supercomputers unless there is some assurance that the new software will run on multiple generations of multiple systems. In order to improve productivity a programming model that abstracts some of the architectural details from software developers is highly desirable.

Summary Priority Research Directions

Detailed recommendations and research directions are presented in the individual chapters. However, there are four overarching priorities that this workshop believes that an exascale initiative must address.

1. Collaboration and co-design. For the past two decades applications have used the same execution model, communicating sequential processes. This period of stability is at an end and co-design among applications, algorithms, programming models, software tools and hardware architecture is essential to effectively develop the next generation of computational capabilities. This includes system level simulation and emulation tools and prototypes and testbeds to enable co-design.
2. Focus on node software and hardware architecture. This is where most of the action will be – greater than 1,000-way, power management, new abstract machine model and programming models, resiliency, memory bandwidth and memory capacity, and data locality.
3. Managing greater than one billion-way parallelism. The shear size of the compute partition of an exascale machine will drive different behavior. Jitter will have a significant impact on system-wide synchronization and applications will be moving, as possible, to asynchronous models of computation. An effective exascale system interconnect (10-100x the extent of today's interconnects) will be a perfect opportunity for co-design given algorithmic, power, and cost constraints.
4. Managing errors. Existing fault tolerance techniques (global checkpoint/global restart) will be unpractical at exascale. Local, distributed checkpoint techniques for saving and restoring state need to be developed into practical solutions. A new fault model and new hardware detection and recovery mechanisms will be necessary to achieve a 24 hour mean time before application interrupt.

Chapter 1. Fault Management and Resiliency

Resilience is a measure of the ability of a computing system and its applications to continue working in the presence of system degradations and failures. Such degradations and failures increase as the scale and complexity of the applications increase to the point that at exascale, if the hardware and software are not fault tolerant, then even relatively short-lived applications are unlikely to finish; or worse, the applications may complete with incorrect results. New paradigms must be developed for handling faults within both the system software and user applications. Equally important are new approaches for integrating detection algorithms in both the hardware and software and new techniques to help simulations adapt to faults.

Studies have shown that failures are systemic so improving resilience at the exascale will require a more holistic approach to the detection and recovery from faults, allowing all the parts of the system to adapt to constant changes. The co-design of the hardware, system software, and applications is critical in order to create resilience at all levels. It will be increasingly important to validate that new extreme scale algorithms are solving the right problem and to verify that the answer produced is correct and not corrupted by numerical stability or errors from transient non-fatal faults.

Two constraints in the design goal for an exascale system are (1) a Mean Time Before Application Interrupt (MTBAI) of 24 hours and (2) a power consumption of no more than 20 MW. The resilience constraint is based on a level of reliability that (1) is "tolerable" to scientists using a supercomputer and (2) allows efficient machine utilization, i.e., most of the simulation time is used to accomplish useful work. Scientists tolerate today's systems, which are typically up for a few days, and they find a system unusable if their application terminates several times a day.

Today's applications obtain resilience by globally checkpointing their state periodically and then restarting from the last checkpoint if a fault terminates the application. Several recent efforts to analyze this approach have shown that the amount of data to be checkpointed and today's average fault rate of five years MTTI/proc will render traditional checkpoint/restart techniques ineffective on million node systems[1].

Switching to local checkpoint approaches where state is written to non-volatile memory would reduce checkpoint overhead by an order of magnitude and extend the time for which checkpointing can be used by a few years, but eventually the systems get so large that checkpointing is no longer effective. See Figure 1.1.

---

[1] Daly, J.T., A higher order estimate of the optimum checkpoint interval for restart dumps, *Future Generation Computer Systems*, volume 22, issue 3, pages 303-312, February 2006.

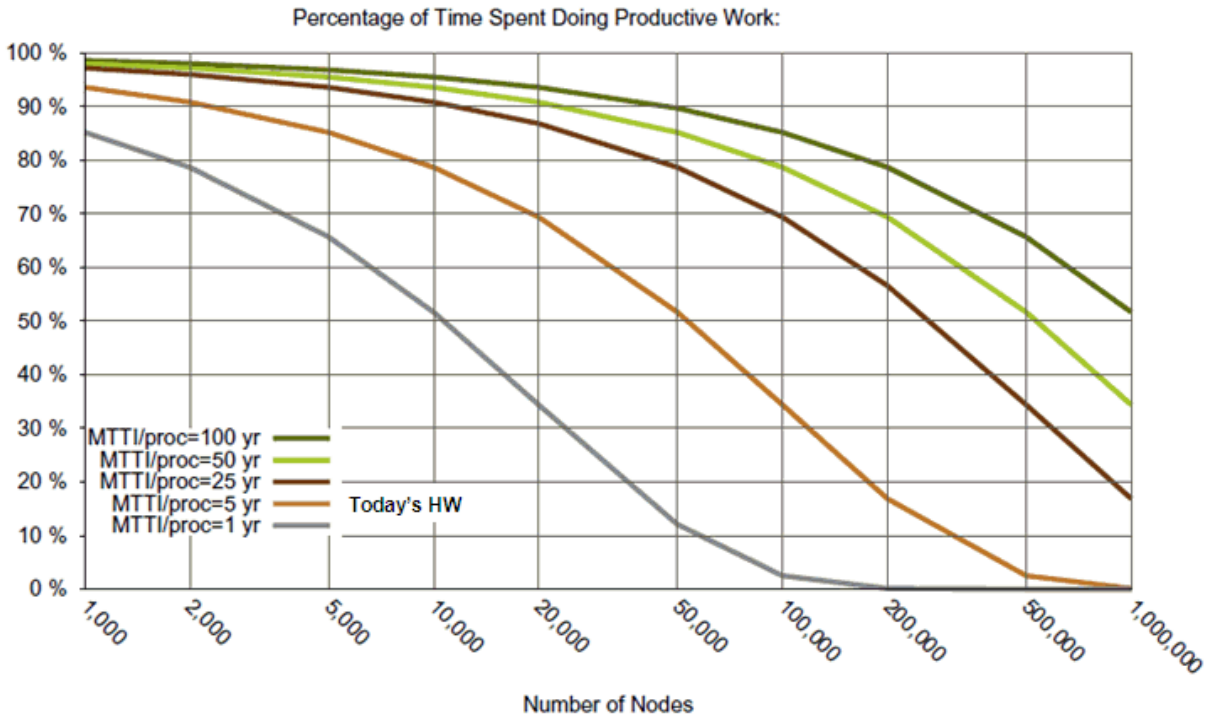


Figure 1.1 As the number of nodes increases, the time between faults decreases; hence, the frequency of checkpointing must also increase. The graph shows how the time spent doing checkpointing decreases the percentage of time doing productive work (assuming different hardware resilience targets).

1.1 Key Co-Design Areas

Co-design is critical to achieving the required exascale resilience because it is the only way to achieve integrated and coordinated detection, localization, notification, and recovery from the stream of faults expected at an exascale. Without it, one layer of the stack could interfere with or prevent recovery efforts by the other layers of the stack. Three key co-design areas where the resilience research should include representatives from different levels of the stack are:

- ***Behavior.*** Discussion of what kind of faults arise from different layers of the stack and where the error is most likely to be detected, how propagation could affect other layers, and which layer(s) would be responsible for fixing or recovering from the fault. An important opportunity in this area is to understand the effects of silent (undetected) errors on application codes, quantify the cost of detecting more of them, and to co-design an approach to handling silent errors.
- ***API and Protocol***. Develop and agree upon an API and protocol for how the software layers can interact with each other regarding faults.
- ***Communication.*** Agree upon the types of two-way communication between layers for improved fault awareness and coordinated recovery. For example, between the system

software and the application here are examples of knowledge, coordination, and awareness communications (three of many possible classes of communication):

- Don't kill me if a fault occurs (knowledge)
- I am a component that will fix all problems of type X (coordination)
- Tell me what fault is detected (awareness)

An exascale system with a million processors will be handling faults continuously; this produces new challenges for system software. Efficient scheduling and resource management become significantly harder with a dynamically changing configuration as does upgrading and monitoring. Several application performance issues are impacted by resilience; load balancing – the most important of these issues – is required to manage the performance of a system that continuously manages faults. Tools are needed to detect load imbalance problems and to assist the dynamic load balancing of applications.

1.2 Defining a Standard Fault Model

Before application developers and runtime software developers can begin to think about creating software that is capable of dynamically adapting to faults in an exascale system, they will need to understand the types of errors that are likely to occur, the methods of notification about faults, and the features available to enable dynamic adaptation and recovery. The answers to these questions would form a "fault model."

As soon as possible the community needs to meet to define a "Standard Fault Model" that formally spells out exactly what detection, notification, and recovery features will be portably supported across exascale systems as shown in Figure 1.2. The adoption of a Standard Fault Model will provide assurance to application developers that changes to make their codes more fault tolerant will be portable and supported into the future.

Further, development of a fault suite or metrics based on this Standard Fault Model would allow software developers to stress resilience solutions and compare them fairly. The testing environment would have the capability to inject faults into a running application to study its response. Since the types of faults depend on the fault model, the specification of the fault model must occur first.

Also needed is a standard "publish/subscribe" interface to increase awareness and response across the entire stack. To avoid being overwhelmed by constant notifications, components in each layer of the stack could "subscribe" to only those notifications that it wants to be aware of, and each component, whether application, system software, or hardware, has the option of "publishing" the errors that it has detected.

Specifying which notifications to get is just one of many kinds of specifications a fault model must support. Another is for an application to be able to specify "don't kill me" if a certain fault occurs, and more generally, for any component to be able to specify its reliability needs, e.g., "I can recover from a failure of type X but not from failures of type Y and Z."

Formal specification of a Standard Fault Model including available features for dynamic adaption and recovery will require an MPI-like process of regular meetings that should begin as soon as possible because progress on several gaps in the resilience roadmap depend on having a fault model.

System Area	Essential	Lower Priority
Node	Detect/Notify complete failure	Notify degraded mode (property) where node may be running at slower speed or have some memory or core failure
Memory	Specify data that MUST be reliable; how reliability is implemented is up to vendors	Notify the different types of memory available on system
Interconnect	Notify of failures to deliver data (like MPI does today)	Specify QOS, Notify degraded bandwidth
System Services	Specify "don't kill me" Request a replacement node	Specify how to manage app in case of failure
File System	Request QOS status	Specify QOS
Miscellaneous	Specify app reliability needs	Specify dependence constraints

Figure 1.2 Possible detection, notification and specifications in a Standard Fault Model

1.3. Factors Driving Up Fault Rates

To date, most projections of increased fault rates at the exascale are based on the failure rate projections of the increased number of parts needed to build such a system, which on average is about five years-- and even that lifetime is probably optimistic because of the 20MW constraint. This wattage constraint leads to several additional factors, besides part count, that will drive the rate of faults up. These are:

- **Number of components.** Both memory and processors will increase by at least an order of magnitude, which will increase both permanent and transient errors in the system in an equivalent amount.
- **Power management cycling.** This factor significantly decreases the components' lifetimes due to the thermal and mechanical stresses on the connectors as found by a recent study of the effects of reducing power consumption by turning unused components off when not used.
- **Smaller circuit sizes, lower voltages to reduce power consumption.** This increases the probability of switches flipping spontaneously due to several factors including thermal and voltage variations as well as cosmic radiation. As the voltage approaches the band gap of CMOS (0.5V), the transient error rate increases exponentially, requiring additional circuits on the chips to detect and correct errors.
- **Heterogeneous systems and more complex node architectures.** These factors make error detection and recovery even harder, for example, detecting and recovering from an error in a

GPU can involve hundreds of threads simultaneously on the GPU and hundreds of cycles in drain pipelines to begin recovery. The effective failure rate is increased because an error in one thread can cause many other threads to be recovered as though they also had errors.

1.4 Major Risk Factors in Exascale Resilience

The workshop identified the top ten gaps in exascale resilience and established the investment priority of these gaps. These priorities correlate to two primary factors: The impending breakdown of global checkpoint/restart as an effective fault tolerance technique and the understandable reluctance of scientists to change their applications. The existing fault tolerance technique of periodically taking a global checkpoint and restarting the entire job from the most recent checkpoint is rapidly becoming obsolete. As shown in Figure 1.1, the productive time of a system will be consumed doing the checkpointing and backtracking. Scientists are reluctant to change their codes until forced to do so, which is understandable given the years of effort and millions of lines of programming in the present version. Moreover, even if they were willing to change their codes, resilience researchers have not defined a set of standard failure modes (a fault model) that the applications should be redesigned to tolerate. The third and fourth items in the top priorities are to understand the types of errors and their behavior in today's petascale systems, because this is needed to define a useful fault model.

Major risk factors and strategies for mitigation:

1. Existing fault tolerance techniques (global checkpoint/global restart) will be unpractical at exascale. Local checkpointing and caching techniques for saving and restoring state need to be developed into practical solutions. Local checkpoint techniques have the potential to extend application resilience through 2015 with minimal code changes.
2. There is no standard fault model, nor standard fault test suite or metrics to stress resilience solutions and compare them fairly. A fault model is needed to allow co-design of exascale resilience across the entire stack from the application down through the hardware.
3. Errors, fault root causes, and propagation are not well understood. Both hardware and software collection and analysis are needed to enable the development of a realistic fault model.
4. Understand rate and type of undetected errors so that hardware and software solutions can be developed to detect them and so that applications can improve verification and validation of the remaining undetected errors.
5. 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. A fault tolerant programming model and runtime must be developed for the software developers to use.
6. System software is not fault tolerant nor fault aware and is not designed to confine faults to limit their propagation. Resilient system software needs to be developed to be aware of faults and support recovery from them when possible.
7. There is no communication or coordination between the layers of the software stack in fault detection and management, nor coordination for preventive or corrective actions. Such an infrastructure is needed to support the co-design of fault tolerance throughout the stack.
8. Present applications are neither fault tolerant nor fault aware. Once a fault model is defined applications can begin thinking about approaches to fault tolerance and when a supporting

infrastructure exists they can be rewritten to continuously manage the stream of faults expected at the exascale.

9. There is no effective fault prediction capability. Research in fault prediction can proceed in parallel with understanding petascale system faults and behavior. It provides the potential to avoid faults by migration of parts of a job from suspect resources.
10. Resistance to adding additional detection and recovery logic right on the chips to detect previously undetected errors, because it will increase the chip design costs, lower overall performance and increase power consumption by an estimated 15%. Further, the consumer market does not need the additional circuits nor wants the increased cost.

1.5 Research Priorities

The following lists of near-term and longer-term research priorities are recommended:

Near-term

1. Develop local checkpoint techniques for saving and restoring state and begin incorporating these solutions into filesystems, system software, and science codes.
2. Develop a standard fault model and complementary standard fault test suite or metrics to allow hardware, software, and application teams to stress resilience solutions and compare them fairly.
3. Study and characterize errors, fault root causes, and propagation in existing petascale systems.
4. Understand the rate and type of undetected errors in existing systems and develop methods to reduce undetected errors.

Longer-term

1. The most common programming model, MPI, does not offer a paradigm for resilient programming. A fault tolerant programming model needs to be developed to support the failure modes defined in the standard fault model.
2. Develop system software that is fault tolerant, fault aware, and a designed to limit fault propagation. Where possible support recovery from faults.
3. Develop a communication and coordination infrastructure between the layers of the software stack for fault detection, management, and corrective actions.
4. Work with application teams to modify their codes to adapt to faults rather than fail or restart.
5. Research in fault prediction and methods to mitigate failures through dynamic task migration
6. Research into methods to allow the continuous repair of hardware and software in a running exascale system.

Chapter 2: Programming Models and Environments

Much of the programming models discussion centered on how to make use of existing language models as it was universally agreed that for the 2015 system, new programming model efforts are not feasible. There was also general agreement that the abstract machine model, which forms the foundation of programing models, is changing. Finally, there was discussion on current "points-of-pain" by the application and tool developers.

The group felt that the main programming environment challenges would be within the new node rather than across nodes, since that is where the biggest changes appear to be headed. The total number of nodes is not changing dramatically, so current practices of MPI between nodes to this scale provides one option of utilizing the exascale systems. Another option is to utilize unified programming models at the global level (UPC, Co-Array Fortran, Chapel, X10, etc.). In this case it would be helpful that the hardware have the capability to support global addressing. Finally, interactions between the programming model and RAS offer the potential for applications to handle errors and recover versus today's environment, which aborts the application. There was further discussion of enabling performance tools from within applications with extensions enabling more inline collection of data.

2.1 Approach and Priority

Past experience indicates that programming model activities are multi-year ventures; starting now to hit a 2015 target is unrealistic for a brand new language. The group chose to focus on existing language efforts, targeting support of the 2015 system. With a changing abstract machine model for the compute node, the biggest issue is the target architecture both in definition and availability of runtime libraries supporting communication and remote task initiation. It is strongly recommended that a professionally developed open source RT be provided to advance all funded (and non-funded) language efforts.

The node model changes are only thinly understood and based on the report from the node architecture group. Until the main components of the architecture definition of this activity settle down, the programming model efforts are preparatory at best. Given the expectation of a fundamental shift in architecture, it is advisable to involve members of the programming models community in the co-design efforts.

We felt that with candidate abstract machine models in hand (and published), the programming models community could work concurrently with other efforts. In addition to funding the open source runtime (RT), the exascale initiative needs to fund a number of competing programming models efforts targeted at this. The programming models efforts should further invite and support application and performance experts who can help define the metrics by which the efforts will be measured in anticipation of a narrowing-down of the programming model funding to focus on the most promising efforts.

Once the leading models are identified, the exascale initiative activity could place more emphasis on the application efforts to assist moving to the new platform/models. It is critical that support be sustained and that the community band together to deter preliminary judgment of results.

2010-2011	Node / abstract machine model development
2010-2012	Investment in multiple hierarchical programming model development/research efforts
2012-2013	Professional open source RT implementation + simulation platform
2012	Early demonstration of programming models + motifs on simulation systems and clusters
2013	Re-prioritize programming model investments based on results
2014	Integrate debuggers / performance tools into programming model efforts
2013-2015	Larger application efforts (utilizing chosen programming models)
2015	Deployment on 100 petaflop systems

Figure 2.1 Notional schedule for programming models and runtime system

2.2 New Opportunities for Co-Design

The discussion with the node architecture group led us to believe that our current abstract machine model will have to evolve to describe additional hierarchy and heterogeneity within each node. This model is a key abstraction upon which modern languages are designed and programing models are implemented. It is important to have programming models expertise in the design of the node model and runtime definition.

2.3 Weaknesses in the Roadmap and Suggested Changes

Productivity is in the roadmap under the title of "Technical Gap" with a suggestion that the productivity will increase 10X. The term "productivity" has been overused in the past decade and its definition and measurement is difficult to assess. We recommend that the goal be measured as the number of applications able to make effective use of the exascale system relative to the target set of applications. We further suggest that application efficiency should be measured in terms of how well the application makes use of the resources on the system that are most critical to the application. For example, if an application is limited by memory performance, it should be measured against how well it can make use of the memory system. Rating applications by flops is becoming less meaningful in a broad sense. We are learning that the expensive parts of these systems will be the energy expended in moving data, not use of arithmetic units.

2.4 Major Risk Factors and Strategies for Mitigation

Programming models are the long pole in the tent with respect to the future of HPC. They are the mechanism by which users interact with and perceive the systems that we are trying to architect. Changes, even necessary ones, from the current de facto standards are likely to be met with

resistance from those with substantial investment in applications under the current model. In addition, industry is marching forward, in part due to the arrival of multi-core everywhere and highly capable GPU technologies and languages to harness them. The HPC programming models activity may be destined to follow rather than lead in this department, however all efforts need to be made in order to enable exascale computing to meet the needs expressed in the exascale initiative.

The “Killer Micro” transition of the early 1990s yielded the arrival of the greatest common denominator of architectural-driven programming models-- MPI. The experience taught the programming models community that without sustained efforts (technical, as well as political and social) new paths to productivity and performance on next-generation systems is impossible. It is likely the case that the same issues and more will be faced in this transition. This is especially true given that clusters are here for the foreseeable future and mainstream industry is working on the programming issue for multi-core and GPU accelerated computing.

One underlying theme from the workshop was the sense that applications need to migrate away from the current popular "bulk synchronous" approach. The new model, a focus on asynchronous design, will enable applications to be more resilient, latency tolerant, and less impacted by jitter in the large systems. This approach is compatible with and beneficial for current systems (although considered more difficult due to being less well-understood at this point).

The following is a brief list of the risk factors and mitigation strategies discussed:

Risk	**Mitigation**
Architectures fail to converge and a common runtime is not possible (incompatible swim lanes).	Co-design between the architects and pmodels teams including vendors is necessary to ensure the abstract machine model is appropriately targeted.
Programming models and common runtime are not available for pre-exascale applications and systems.	Early analysis of MPI + existing on-node programming models and narrowing the field is necessary to provide suitable pre-exascale environment.
Programming models and common runtime are overly restrictive and limit innovation of the architectures and applications.	Early experience with the runtime and interaction with vendors should help.
Lack of consensus - in design, vendor buy-in, and use by applications.	Difficult. While we are taking the task of paying close attention to memory movement with a node, we hope to provide an alternative to explicit messaging to accomplish this.

Premature dismissal by application programmers, a group notably resistant to change if it necessitates rewriting of their code.	Be as upward compatible as possible and emphasize incremental editing in the applications, where application programmers can port and run on the new systems as is, but then take advantage of portable code modifications that open node architecture features to enhance the performance.
Competing technologies win popularity – especially those that exist today with hardware acceleration (OpenCL, CUDA).	Compatibility with emerging models and open interfaces so those models can align themselves with the high-end systems. Additionally, support for cluster implementations for these models to ensure portability of applications.

2.5 Need for Testbeds, Prototypes, and Simulators

Once the abstract machine model is defined, the programming models community will have a critical need for a runtime/simulator to bootstrap language efforts. While compatibility with current systems is key, it is very important to have access to the new architectural model in order to study/understand and optimize for future systems. Much can be learned with testbed and prototype hardware that is able to mimic the features of the new model.

2.6 Programming Models Discussion

We believe that the following characteristics will be important for exascale programming models:

- ***Post-SPMD execution models*, including increasingly dynamic and/or nested models of parallelism:** to address the heterogeneity and hierarchy in the target architecture; and to tolerate the expected increase in execution time variations in the system.
- ***Control over locality* with an increased focus on 'vertical' or intra-node locality issues:** to control affinity given the increased heterogeneity and hierarchy in the architecture.
- ***Multi-resolution design:*** high-level abstractions to help manage the system complexity while supporting the ability to drop to lower, more manual levels within the same programming model.
- ***Portability:*** ability to easily port existing codes, with the understanding that it may require tuning or the use of new features to optimally map to the exascale architecture. We anticipate that this will be more challenging than during the past decade due to the variety of node architectures being pursued and the lack of familiar abstract models for targeting these node designs.

We also believe that it makes sense to invest in a number of programming models as a risk mitigation strategy by supporting a combination of more incremental and aggressive approaches.

Models to Monitor. Programming models in this category are expected to continue evolving and receiving considerable investment independently of the exascale community. Examples include

CUDA, OpenCL, pthreads, and TBB (Matlab was suggested in the final brief). We believe that such models should be tracked to evaluate their role in exascale programming, yet we do not anticipate that they will require significant input or funding from the program. These technologies may play a role in our broader solution, either as part of a hybrid-programming model (e.g., MPI+CUDA), or as building blocks for higher-level software.

Evolving Established Hybrid Models. This category consists of well-established HPC programming models that ought to be evolved to maintain legacy codes and to better support exascale architectures. MPI and OpenMP are the two main examples here. For hybrid MPI+OpenMP programming (or MPI+OpenCL, MPI+CUDA, MPI+pthreads) where the scale of MPI ranks is roughly equivalent to today's largest runs, it was our sense was that MPI was unlikely to require major changes, since current MPI implementations are likely to scale to order 1M ranks. In order to support effective MPI-only programming on exascale architectures (where the MPI ranks would be >> 1M), we believe that extensions will need to be explored. Examples might include hierarchical notions of MPI ranks and communicators. We were also generally supportive of exploring single-sided communication and active message capabilities within MPI-3 to support more dynamic and loosely coupled execution models.

Our group was skeptical that OpenMP as currently defined would be a natural match for the emerging node architectures due to the heroic compiler efforts that we believe would be required to partition work across hierarchical or heterogeneous node resources. We also cited the lack of ability to control locality within OpenMP as a drawback. We were supportive of directions to extend OpenMP with new directives that would better address exascale node architectures while retaining its current division of labor philosophy between user and compiler. The PGI and CAPS notations for accelerator programming should serve as good input for possible directions to pursue.

New Hybrid Models. Given our anticipated need for increased locality control within a node, one option that was discussed was to using a hybrid MPI+PGAS model in which MPI would be used for the inter-node communication and parallelism along with a PGAS language such as UPC, CAF, X10, or Chapel for expressing computation within the node. It is expected that the PGAS language in question would need to be extended to reflect the hierarchy within the node architecture, but that such models may be a better fit than OpenMP due to their support for locality/affinity. Other novel hybrid models could use some other inter-node technology like SHMEM, Global Arrays, UPC, or CAF in combination with an intra-node technology like CUDA, OpenCL, or an extension of OpenMP.

Holistic Models. This category involves standalone programming models designed to address both inter- and intra-node concerns such as the HPCS languages (Chapel and X10) or traditional PGAS languages (CAF and UPC). The HPCS languages have the advantage of being designed with most of our desiderata in mind including dynamic/nested execution models, locality/affinity control, and multi-resolution design, yet their disadvantage is that they are not yet mature or widespread technologies. They may require work to map to exascale nodes due to the increased hierarchy/heterogeneity. Other dynamically parallel models that should be considered include Charm++ and ParalleX, though these were not well represented within the group.

While the traditional PGAS languages are better established and also support locality control, they would likely require modification to serve as a standalone programming model for exascale due to their use of the SPMD programming/execution model.

Domain-Specific Languages. We touched on domain-specific languages, but did not consider them to be a major part of the program effort because (1) we believed they should be built upon one of the general programming models above; and (2) their inherent narrowness limits the breadth of their applicability.

More Aggressive Models. We do not believe that there is necessarily a need for more revolutionary programming models in the exascale program for the following reasons: (1) due to the extreme changes in the architecture, fitting existing models to the architecture will be sufficiently challenging without taking on new languages; (2) the HPCS languages are still under development and evaluation and were intended to be applicable beyond the petascale; (3) the timeframe: it is currently thought that the programming models will need to be available on the early side of the program to support evaluation and the 2015 machine; (4) a lack of hot ideas or enthusiasm to develop some within the group.

Evaluation/Adoption of Programming Models. We discussed barriers to the adoption of new programming models by the application community, particularly as they become more exotic. There seems to be a tension between the "we want new, more productive alternatives to MPI+OpenMP" attitude and the understandable unwillingness to change technologies. We also discussed the inherent tension between more innovative programming models and backwards compatibility, as well as the role that good interoperability support can play in helping with this. We discussed the danger of prematurely dismissing new programming models based on early hands-on evaluations rather than a measured evaluation of the language's potential and optimizability. Finally, we discussed the challenge of scaling programming models from small benchmarks and kernels to full-size applications due to the perceived lack of existing intermediate-sized applications. In all these areas, we believed that increased involvement and investment from the applications community would help make new programming models more effective and adoptable.

To this end, we discussed funding models in which application groups would be given nontrivial amounts of money to work alongside programming models groups to help develop mini-applications, study and use new languages, provide feedback, and generally increase the chances that the resulting language would be useful to them. It was generally believed that this money should be closely monitored to ensure that it was not diverted to support other activities (e.g., development of the base MPI implementation) or to fund people that were not deeply embedded in the application groups.

Exascale Runtime

The second major component of our discussion was support for a runtime library that could serve as an implementation layer for several of the language-based programming models described above. The main goal of the runtime library would be to abstract away architectural details such as the speeds and feeds and topological details of a compute node in order to support portability across distinct points in the exascale architectural design space. The runtime's purpose would be to support the communication required to map data and tasks to hardware resources, and to access remote data. The runtime could also have a role in tasking and/or memory management, depending on requirements. We thought of the runtime as supporting inter- and intra-node capabilities, with the option of disabling one of those modes at configuration time.

We imagined that the design of the runtime library would be a cooperative effort between language implementers, the runtime team, and the hardware architects. We discussed both top-down designs, such as specifying the runtime as a proposed MPI extension, as well as bottom-up

approaches more similar to what has been done in the past with SHMEM, GASNet, and ARMCI. Given the novel aspects of exascale architectures, our sense was that the bottom-up approach might be more appropriate due to the lack of experience with exascale architectures and our sense that it was a lighter-weight process. We proposed that the runtime should be developed by a dedicated professional team in order to ensure a stable code foundation for the other programming models.

Tools and Libraries

We spent the least amount of time discussing tools and libraries, due primarily to the composition of our team. We believe that tools, particularly debuggers and performance analysis tools, as well as libraries will continue to play an important role in the exascale timeframe. The general theme we discussed was tools that do a good job of synthesizing massive amounts of data such that potential issues could be easily identified without work proportional to the number of nodes, tasks, or threads. Once a correctness or performance issue is identified, however, we discussed the importance of being able to (a) associate it back to the code at a level (and in terms) that naturally suits the programming model, and (b) to be able to dive down as close to the architecture as the programmer requires/desires. There was a general sense that we need alternatives to vendor-supplied tools as a risk mitigation strategy for shipping delays.

We did not discuss libraries in depth other than to acknowledge their value and mention the need for lower-level programming models that support tuning right down to the hardware.

2.7 Conclusions

The current view of exascale architectures presents the following challenges:

- Increased heterogeneity and hierarchy in the node architectures, constituting the first significant departure from the abstract node model that has served our community well for the past 15-20 years.
- The need for identifying and managing massive degrees of parallelism.
- Increased complexity in the memory hierarchy and memory types, particularly the inclusion of explicitly controlled (scratchpad) memories.
- Interaction with hardware power throttling features.
- Increased need for application interaction with resiliency features due to the increased number of parts and consequent likelihood of failure.
- Increased desire for applications to interact with the system-performance-monitoring infrastructure so as to be self-aware of performance issues.

The consensus of the group was to favor pursuing three technologies:

1. A well-defined abstract machine model and an open source (professionally developed) runtime layer to serve as a compiler target for programming model implementations and to initiate new topics such as application ←→ RAS system interactions and application visibility into runtime performance metrics.
2. Multiple diverse programming models: a multi-layered approach with MPI between nodes and an on-node model made up of one or more existing languages, a unified/global-view model, or other approaches such as domain specific models.
3. Tools (debugger, performance) that help the user grapple with the massive amount of parallelism that exascale applications will need to use.

Chapter 3. Node Architecture and Power Reduction

Node architectures are expected to change dramatically in the next decade as power and cooling constraints are limiting increases in microprocessor clock speeds. Consequently computer companies are dramatically increasing on-chip parallelism to improve performance. The traditional doubling of clock speeds every 18-24 months is being replaced by a doubling of cores or other parallelism mechanisms. During the next decade the amount of parallelism on a single microprocessor will rival the number of nodes in the first massively parallel supercomputers that were built in the 1980s. Applications and algorithms will need to change and adapt as node architectures evolve. In particular, they will need to manage locality to achieve performance. There is an unprecedented opportunity for application and algorithm developers to influence the direction of future architectures so that they meet DOE mission needs.

In order to capture areas of agreement and to document areas of disagreement among the participants, we adopted the concept of "swim lanes" to describe two different viable approaches in the design space. The first swim lane describes the manycore design point, which extrapolates trends involving large numbers of simple processor cores each with a few hardware threads. The second swim lane extrapolates trends in GPU architecture characterized by an order of magnitude more threads over many core designs modestly different design parameters and substantially different semantics for programming models (OpenCL/CUDA/Streaming).

3.1 Areas of Substantial Agreement

The most prominent areas of agreement in the workshop are as follows:

- The primary design constraint for future HPC systems will be power consumption.
- The biggest energy cost is in data movement, especially moving data on and off chip.
    - Data movement will be a bigger factor for system energy consumption and cost than FLOP/s.
    - The high cost of data movement places very strong constraints on memory and interconnect bandwidth.
    - The cost of data movement also increases importance of both vertical and horizontal locality management techniques -- both hardware mechanisms and the programming models and abstractions to elegantly expose locality management control.
- Primary growth in explicit parallelism is on-chip
    - 100x growth in parallelism on-chip
    - 10x growth in parallelism off-chip
- Energy and performance costs should be reflected in abstract machine model
    - Current abstract machine model has flat or 2-level costs, which do not match the above specified technology trends
- Clock rates will remain nearly the same as today's chips (we will assume 1 GHz for simplicity)
- Cost considerations may limit an exascale system to a memory capacity that improves only by a factor of 100x in comparison to the system peak floating point rate, which will improve by 1000x.
- Power and complexity costs make it clear that we cannot depend on out-of-order instruction streams to hide latency and improve performance.
- Off-chip latencies are unlikely to improve substantially over existing systems.

- By 2015 it will be feasible from a market standpoint to integrate scalar cores with an accelerator.
- SoC (System on a Chip) level integration would play an increasingly important role in future HPC node designs
- Could we create a non-profit model for sustaining a processor design team focused on high performance computing over several processor design cycles? There was agreement that this would only succeed if the design effort were heavily leveraged from other multi-billion dollar markets.
- There will likely be 2-4-levels of on-chip memory hierarchy that can be managed explicitly or flipped to implicit state.

3.2 Areas of Substantial Disagreement

The most prominent areas of substantial disagreement in the workshop are as follows:

- Whether 1 or 10 Teraflops per node/chip would be achievable in the 2018 timeframe. We have two design points to represent this divergence.
    - Swim lane 1: ~1K FPUs per chip
    - Swim lane 2: ~10k FPUs per chip
- How many address bits will be supported in mainstream implementation of PGAS languages to support noncoherent global addressing.
- Although there was agreement that globally addressable memory is generally good for lowering the cost of explicit data movement, there was disagreement on whether PGAS models are inherently better at locality management.
- There was disagreement about the extent to which aggressive investments in optical/photonic technology will be necessary or effective in mitigating the energy cost of data movement.

3.3 Cost and Power Constraints

In an ideal world, we would design systems that would never subject applications to any performance constraints. However, power and cost of different components of an HPC system force system architects to consider difficult trade-offs that balance the actual cost of system components against their effect on application performance. For example, if doubling floating point execution rate nets a 10% gain in overall application performance, but only increases system costs by 5%, then it is a net benefit despite the counter-intuitive effect on system balance. Co-design is important here to fully understand the cost impacts of key design choices so that they can be evaluated against their benefit to the application space.

For the purpose of this evaluation, we adopt a limit of $200M for the capital cost of procuring a system and 20MW as the feasible design limit for the power consumed by an exascale system in 2018.

The cost of power: Even with the least expensive power available in the U.S., the cost of electricity to power supercomputing systems is a substantial part of the Total Cost of Ownership (TCO). When burdened with cooling and power distribution overheads, even the least expensive power in the U.S. (< 5cents/KWH) ultimately costs $1M per Megawatt per year to operate a system. To keep the TCO less than the capital cost of procuring a system and based on the limits of reasonable power densities for a feasible system design, we have generally adopted 20MW as the upper limit for reasonable system design. This figure is movable, but at great cost and design risk.

The cost of a FLOP: Floating point used to be the most costly component of a system both in terms of design cost and power. However, today, FPUs consume a very small fraction of the area of a modern chip design and a much smaller fraction of the power consumption. On modern systems, a double-precision FMA (fused multiply add) consumes 100picojoules. Reading the double precision operands from DRAM costs about 2000pJ by contrast. By 2018 it will consume about ~10.6pJ/op on 11nm lithography technology, and the cost of reading from DRAM will only improve modestly to 1000pJ unless more energy-efficient memory technology is developed.

With these figures of merit, it would only consume 100W to put 10 Teraflops on a chip, which is easily achievable. However, the 2000 watts of power required to supply memory bandwidth to those floating point units at a modest memory bandwidth to floating point ratio of 0.2. The consequence is that we can engineer far more floating point capability onto a chip than can reasonably be used by an application. Engineering FLOP/s is not a design constraint – data movement presents the most daunting engineering and computer architecture challenges.

The cost of moving data: Memory interfaces and communication links on modern computing systems are currently dominated by electrical/copper technology. However, wires are rapidly being subsumed by optical technology. To understand why this transition is occurring, it is best to look at Miller and Ozaktas' journal article[2] that relates the energy-cost of moving data on a copper wire to the Telegraph Equation, which says

$$Energy\_to\_move\_data = bitrate * length^2 / cross\_section\_area\_of\_wire$$

Ozaktas and Miller point out that the Telegraph Equation has the following consequences to system design:

- The energy consumed increases proportionally to the bit-rate, so as we move to ultra-high-bandwidth links, the power requirements will become an increasing concern.
- The energy consumption is highly distance-dependent (the square of the length term), so bandwidth is likely to become increasingly localized as power becomes a more difficult problem.
- Improvements in chip lithography (making smaller wires) will not improve the energy efficiency or data carrying capacity of electrical wires.

In contrast, optical technology does not have significant distance-dependent energy consumption. It costs nearly the same amount of energy to transmit an optical signal one inch as it does to transmit it to the other end of the room. Also, signaling rate does not strongly affect the energy required for optical data transmission. Rather, the fixed cost of the laser package for optical systems and the absorption of light to receive a signal are the dominant power costs for optical solutions.

As the cost and complexity of moving data over copper will become more difficult over time, so the cross-over point where optical technology becomes more cost-effective than electrical signaling has been edging closer to the board and chip package at a steady pace for the past two decades. Contemporary short-distance copper links consume about 10-20picojoules/bit, but could be improved to 2pJ/bit by 2018. However, the efficiency and/or data carrying capacity of the copper links will fall off rapidly with distance (as per Telegraph Equation) forcing a

[2] D. A. B. Miller and H. M. Ozaktas, "Limit to the Bit-Rate Capacity of Electrical Interconnects from the Aspect Ratio of the System Architecture," Journal of Parallel and Distributed Computing, vol. 41, pp. 42-52 (1997) article number PC961285.

movement to optical links. Contemporary optical links consume about 30-60pJ/bit, but solutions that consume as little as 2.5pJ/bit have been reported in the lab. There was general agreement that in the 2018 timeframe optical links are likely to operate at 10pJ/bit efficiency.

The consequence of the power consumption of these links is that it will not be feasible to support a globally flat bandwidth across a system due to power limits. Therefore, we should expect is highly localized bandwidth in an attempt to exploit locality in scientific computations. Therefore, algorithms, system software, and applications will need to aware of data locality. The programming environment must enable algorithm designers to be able to express and control data locality more carefully. The system must have sufficient information and control to make decisions that maximally exploit information about communication topology and locality. Flat models of parallelism (e.g. flat MPI or shared memory/PRAM models) will not map well to future node architectures.

Consequences for locality management

The energy cost of moving data to different levels of the system is large relative to the cost of a floating-point operation. See Figure 3.1. The cost of data movement will not improve substantially whereas the cost of performing a floating -point operation will likely improve between 5x to 10x. The effect on chip architecture is similar to the effect of the rising cost of gasoline relative to the cost of a new automobile where rising gasoline costs. Imagine the effect if gasoline cost 10x more than it did today? You would pay much more attention to optimizing your use of an automobile to minimize gas usage. Likewise, the increased cost of moving data relative to the power consumed by the FPUs has dramatically shifted attention away from FLOP/s and more towards locality management.

There are two primary categories of locality management – vertical locality management and horizontal locality management. Vertical locality management is management of data locality for data that moves up and down the memory hierarchy to a processor core whereas horizontal locality management refers to managing data movement and communications between peer processors.

Controlling vertical locality management involves use of data prefetch instructions, large register sets, and other forms of software managed memory. Caches are the most convenient to program, but they virtualize the notion of on-chip vs. off-chip memory, which complicates vertical locality management as users must reverse-engineer the behavior of the cache hierarchy. Software managed memory, such as the Cell processor "local store" offers the most user control over vertical locality, but they can be very difficult to program because there are few good programming abstractions for managing them. The GPU swim lane makes use of very large explicitly managed register sets to carefully control data locality, but requires a substantially different programming semantics (CUDA and streams) to manage those resources. There is increased interest in caches that can be dynamically switched between automatic management and software management, such as the programmable cache on NVIDIA's most recent Fermi. There was agreement that we may see more use of software-managed memory on future systems if the programming model/abstraction problem can be overcome.

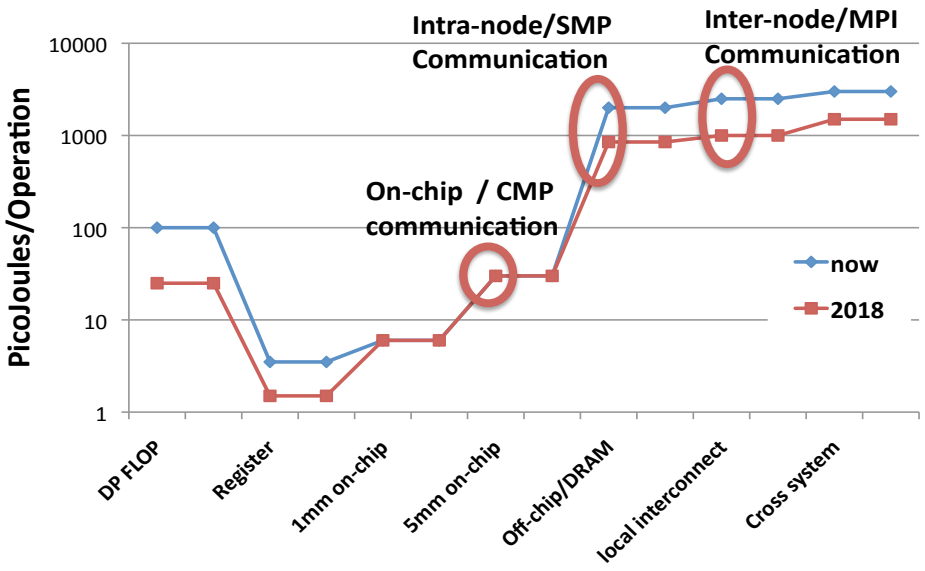


Figure 3.1 Energy cost of data movement relative to the cost of a flop for now and for 2018 systems, without further investment. The biggest delta in energy cost is movement of data off-chip. Therefore, future programming-environments must support the ability of algorithms and applications to exploit locality, which will, in turn, be necessary to achieve performance and energy efficiency.

Horizontal locality management primarily involves managing horizontal data movement. Cache-coherent systems already use snoop filters (such as AMD's latest Opteron chips) to reduce redundant or non-useful cache coherence traffic between chips that comprise an SMP. Scaling up cache-coherence will require even more sophisticated methods in the future, but non-uniform bandwidth and latencies between cores require more explicit control over data and process placement to mitigate what are termed "NUMA effects." There was general agreement that lightweight explicit data movement protocols (such as Global Address Space) further improve our ability to control horizontal data locality, but there was disagreement as to whether PGAS (Partitioned Global Address Space) programming models improve horizontal locality management in practice.

There is substantial agreement that both vertical and horizontal locality management will be of penultimate concern for both swim lanes. Past attempts to exploit intra-node parallelism did not show significant benefits primarily because the cost of moving data within a node was not substantially lower than the cost of moving data across the interconnect because the cost of moving data off-chip dominated the energy costs. However, modern chip multiprocessors have CPU's co-located on the same chip. Consequently, there is a huge opportunity to capture energy-efficiency and performance benefits by directly taking advantage of intra-chip communication pathways.

3.4 Memory Subsystem

Ultimately, memory performance is primarily constrained by the dynamics of the commodity market. One key finding of the workshop was that memory bandwidth is primarily constrained by power and efficiency of the memory interface protocols, whereas memory capacity is primarily constrained by cost. Early investments in improving the efficiency of DRAM

interfaces and packaging technology may result in substantially improved balance between memory bandwidth and floating point rate. Investments in packaging (mainly chip-stacking technology) can also provide some benefit in the memory capacity of nodes, but it is unclear how much the price of the components can be affected by these investments given commodity market forces.

Memory Bandwidth: The power consumed by data movement will dominate the power consumption profile of future systems. Chief among these concerns is the power consumed by memory technology, which would easily dominate the overall power consumption of future systems if we attempt to maintain historical bandwidth/performance ratios of 1 byte/flop. A limit of 20 MW as the limit for feasible designs, it will force us to very difficult trade-offs regarding power consumption and breadth of applications that can run effectively on the system.
For example, today's DDR-3 memory interface technology consumes about 70picoJoules/bit – which comes to approximately 5000 pJ to load a double-precision operand (accounting for ECC overhead). If we extrapolate the energy-efficiency of memory interfaces to DDR-5 in 2018, the efficiency of the memory could be improved to 30pJ/bit. A system with merely 0.2 bytes/flop of memory bandwidth would consume > 70 MW of power, which is not considered a feasible design point. Keeping under the 20 MW limit would force the memory system to < 0.02 bytes/flop, which would severely constrain the number of applications that could run efficiently on the system. See Figure 3.2.

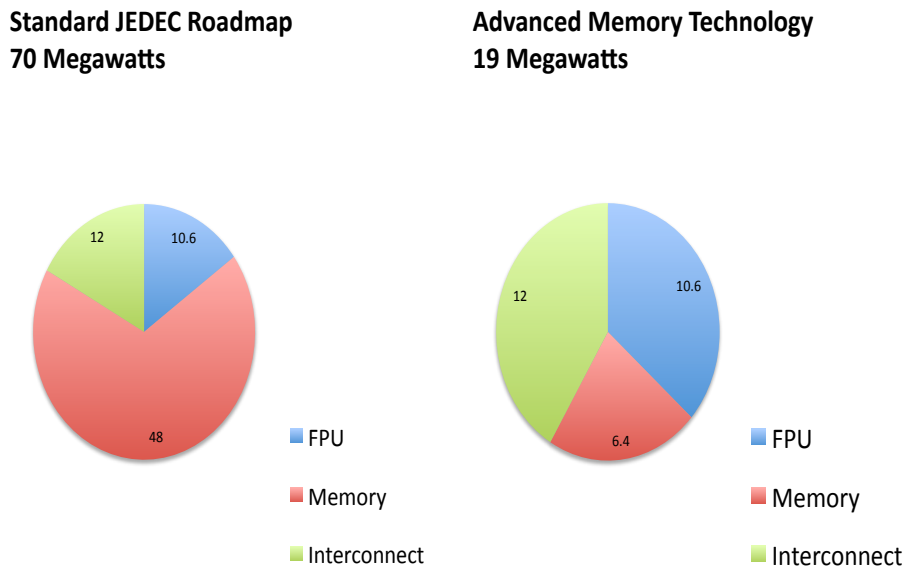


Figure 3.2. If we follow standard JEDEC memory technology roadmaps, the power consumption of a feasible exascale system design (using 0.2 bytes/flop memory bandwidth balance) will be >70 MW due to memory power consumption, which is an impractical design point. Keeping memory power under control will either require substantial investments in more efficient memory interface protocols, or substantial compromises on memory bandwidth and floating point performance (< 0.02 bytes/flop).

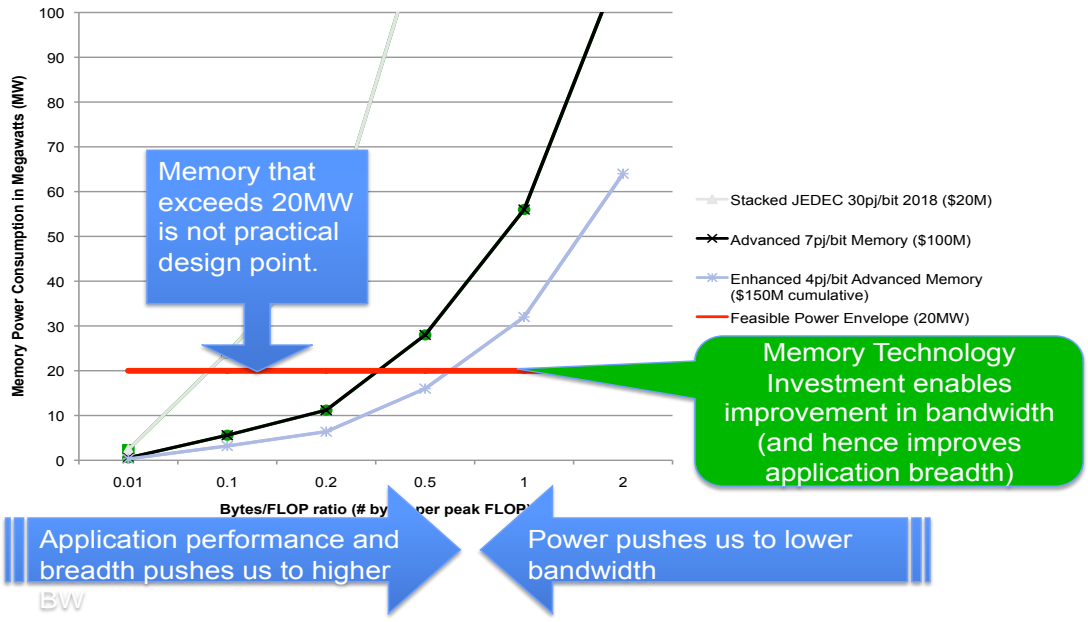


Figure 3.3 This figure illustrates the trade-offs between memory power consumption and the desire for a more broadly applicable exascale system design under different assumptions about investment in advanced memory technology.

We cannot reach reasonable memory energy efficiency by following the JEDEC roadmap. Getting to reasonable energy efficiency requires development of new, more efficient interface designs and memory protocols. Advanced memory technology can get to about 7pJ/bit with investments to bring the technology to market. The upper limit of this new technology is estimated to be 4pJ/bit (excluding memory queues and controller logic). Therefore, in order to maintain 0.2 byte/flop system balance and stay under a 20 MW design limit for power requires either substantial investments in advanced memory technology, or a substantial degradation in system memory balance. See Figure 3.3. As always, these ratios are movable. For example, the power limit could be relaxed, but would put the feasibility of field siting such a system in jeopardy.

Memory Capacity: One figure of merit for improvements to HPC systems is the total memory capacity. More aggregate memory enables systems to solve problems that have either proportionally higher resolution, or more physics fidelity/complexity – or both. However, there was consensus at the meeting that cost considerations may limit an exascale system to a memory capacity that improves only by a factor of 100x in comparison to the system peak floating point rate which will improve by 1000x. This is a movable parameter in the design space of the machine, but the consequence of moving this parameter is increased cost for the memory subsystem.

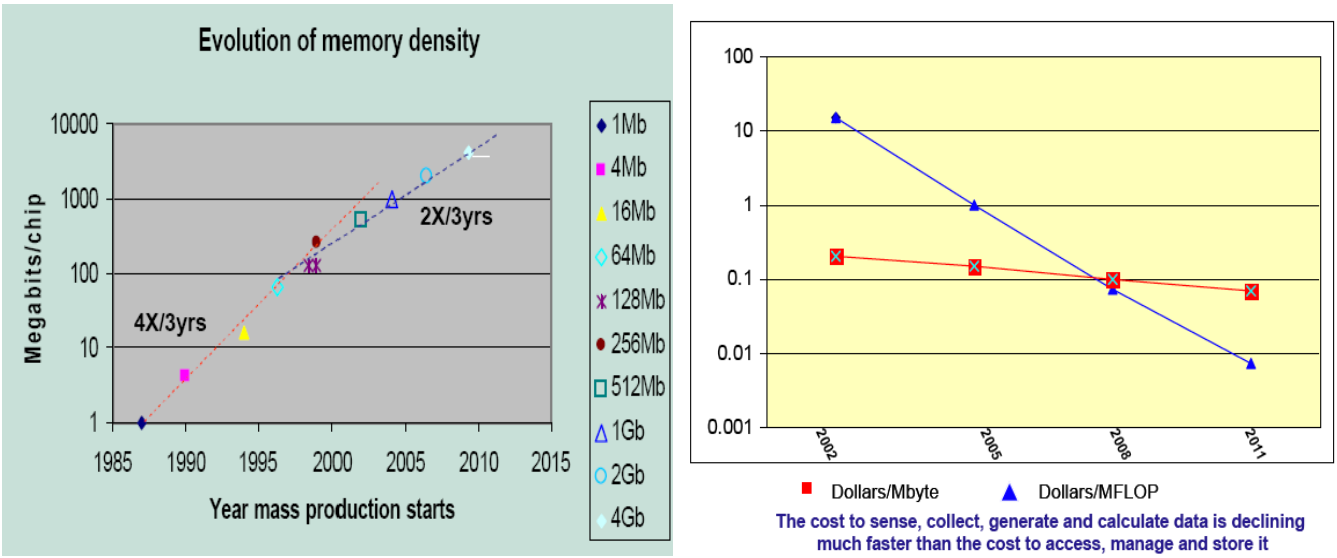


Figure 3.4 The rate of improvement in memory technology improving at slower rates. Images courtesy of International Business Machines, © International Business Machines Corporation.

The DRAM capacity of a system is primarily limited by cost, which is defined by the dynamics of a broad-based high-volume commodity market. The commodity market for memory makes pricing of the components highly volatile, but the centroid of the market is approximately $1.80/chip. Figure 3.4 illustrates that the rate of memory density improvement has gone from a 4x improvement every three years to a 2x improvement every three years (a 30% annual rate of improvement). Consequently the cost of memory technology is not improving as rapidly as the cost of floating point capability. Given the new rate of technology improvement, eight gigabit memory parts will be widely available in the commodity market in the 2018 timeframe and 16 gigabit parts will also have been introduced. It is unclear which density will be the most cost-effective in that timeframe.

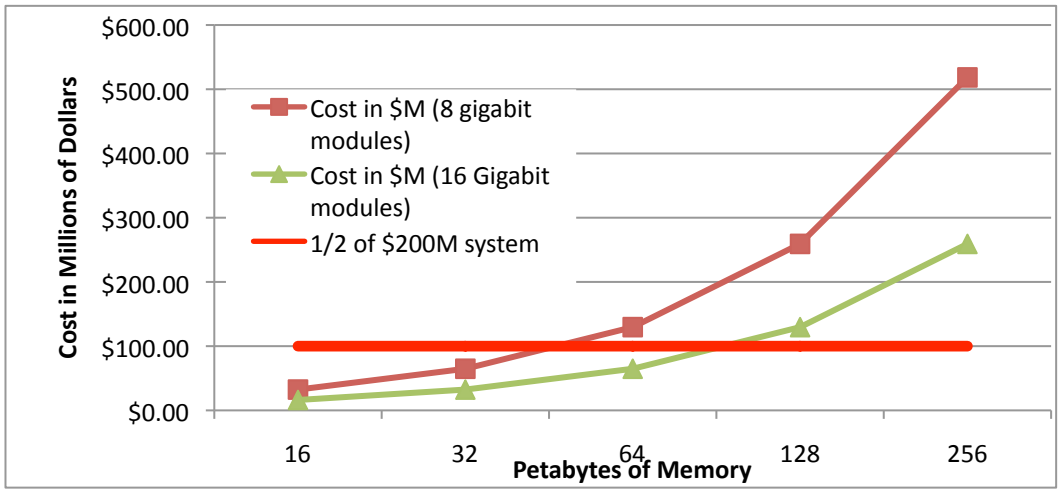
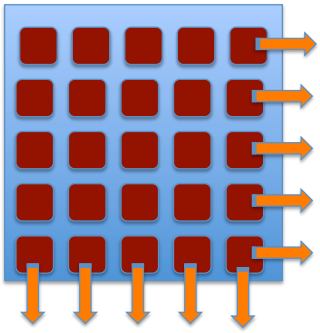


Figure 3.5 There are two different potential memory chip densities are possible in the 2018 timeframe. It is less certain which option will be at the apex of the commodity cost scaling.

If we assume that memory should not exceed 50% of the cost of a computer system, and that the anticipated capital cost of an exascale system is $200 million, then Figure 3.5 shows the approximate memory capacity that we could afford assuming either eight gigabit chips or 16 gigabit chips lies somewhere between 50 and 100Petabytes. Again, these are not hard limits on capacity, but do have a substantial effect on the cost of the system, so the trade-off of memory capacity against other system components must be considered carefully given a finite budget.

3.5 Strawman Node Architecture for 2018

There are many opportunities for major reorganization of our model-of-computation to take better advantage of future hardware design constraints. However, as a general design principle, it is better to take a bunch of pre-proposed ideas and synthesize them into something productive rather than to come up with something completely off-the-wall. Therefore, much of the discussion of inter-processor communication semantics and node organization focused on evolutionary rather than revolutionary features. However, much of what we term evolutionary here is revolutionary from the perspective of an application implements wholly in MPI.



- ~1000-10k simple cores
- 4-8 wide SIMD or VLIW bundles
- Either 4 or 50+ HW threads
- On-chip communication Fabric
  - Low-degree topology for on-chip communication (torus or mesh)
  - *Scale cache coherence?*
  - cache coherent clusters
  - HW msg. passing
  - Global (nonCC memory)
  - Shared register file (clusters)
- Off-chip communication fabric
  - Integrated directly on an SoC
  - Reduced component counts
  - Coherent with TLB (no pinning)

Figure 3.6 Schematic of future node architecture. The number of functional units on the chip will need to scale out in a 2D planar geometry where locality of communication between the functional units will be increasingly important for efficient computation.

What range of clock rates: There was general agreement that the clock-rates for 2018 chip designs will remain nearly the same as today's chips. For the sake of clarity, we will assume the clock to be 1 GHz. This sets clear design constraints for the number of floating point functional units will be present on a future chip design. In order to keep the component counts for future systems within practical limits (< 100k nodes), a node must perform between 1-10 Teraflops. At 1 GHz, that means there will be between 1000 and 10,000 discrete FPUs on the chip.

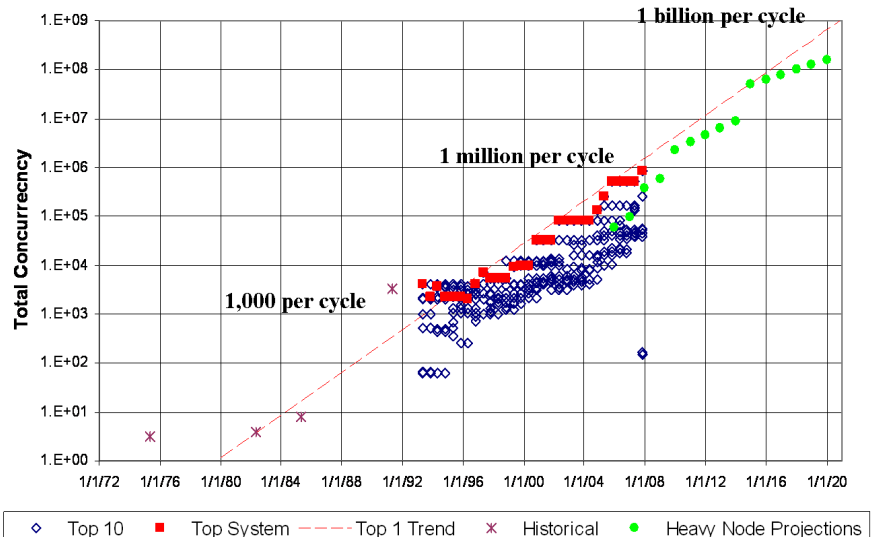


Figure 3.7 Due to the stall in clock frequency improvements, future performance improvements will be derived from increased explicit parallelism. 2018 systems may have as many as 1-billion-way parallelism.

Instruction Level Parallelism: Until recently, microprocessors depended on Instruction Level Parallelism and out-of-order execution to make implicit parallelism available to a programmer and to hide latency. There was general agreement that power and complexity costs make it clear that we cannot depend on out-of-order instruction streams to hide latency and improve performance. Instead, we must move to more explicit forms of exposing parallelism such as SIMD units and chips with many independent CPUs.

Instruction Bundling (SIMD and VLIW): One way to organize floating-point functional units to get implicit parallelism is to depend on bundling multiple operations together into SIMD or VLIW bundles. The benefit of such bundling is that they enable finer-grained sharing of data among the instructions, which lowers energy costs and control complexity. Although SIMD is the most popular approach to organizing FPUs today, there may be movement towards more of a VLIW organization because it is more flexible in the mixing of instructions.

Recently, SIMD units on x86 chips have doubled in recent years, but the ability to fully exploit wider SIMD is more questionable. GPUs also depend on very wide SIMD units, but the semantics of the GPU programming model (CUDA for example) make it easier to automatically use SIMD or VLIW lanes. Currently, NVIDIA uses 32-wide SIMD lanes, but there is a pressure to shrink this down to 4-8. Current CPU designs have a SIMD width of 4 slots, but will likely move up to 8 slots. Overall, this indicates a convergence in the design space towards 4-8 wide instruction bundles (whether it be SIMD or VLIW).

Latency

It was generally agreed that off-chip latencies are unlikely to improve substantially over existing systems. With a fixed clock rate of 1 GHz, the distance to off-chip memory on modern systems is approximately 100ns (100 clock cycles away), and will potentially improve to 40-50ns (40-50 clock cycles away from memory) in the 2018 timeframe. A modern interconnect has a messaging

latency of 1 microsecond. Most of that latency is on the end-points for the message (message overhead of assembling a message and interrupt handling to receive it). By 2018, this could improve to as little as 200-500ns for message latency, which is at that point limited by the speed of light (0.75c in optical fiber comes to about 5ns latency per meter of cable).

Lastly, the message injection rates of modern systems (an indirect measure of the overhead of sending messages) are millions of messages/second per network port on a leading-edge design. If the interconnect NIC is moved on-chip for an SoC design, it may be feasible to support message injection rates of up to billions of messages per second for lightweight messaging such as one-sided messages for PGAS languages.

With no substantial improvements to latency off-chip and cross-system, the bandwidth-latency product for future systems (which determines the number of bytes that must be in flight to fully saturate bandwidth) will be large. This means there must be considerable attention to latency hiding support in both algorithms and in hardware designs. The approach to latency hiding was a source of substantial disagreement.

Multithreading to Hide Latency: Little's Law[3] is derived from general information theory, but has important application to understanding the performance of memory hierarchies. Little's Law states:

#outstanding_memory_requests = bandwidth * latency

In order to fully utilize available bandwidth of a memory interface, this equation must be balanced. If you have a high bandwidth memory interface, the bandwidth will be underutilized if there are an insufficient number of outstanding memory requests to hide the latency term of this equation (latency limited). Since we will no longer be depending on complex out-of-order instruction processors to hide latency in the memory hierarchy, there will be increased dependence on hardware multithreading to achieve latency hiding (i.e., existing superscalar CPUs have latency to local memory of order 100ns, but don't have to hide all of the time due to cache reuse).

In swim lane 1, the manycore chip architectures currently support 2-4-way multithreading, and may increase that to 4-8 way multithreading in future architectures, depending on the energy cost. GPUs currently depend on 48-64-way hardware multithreading and will likely stay there. While this was an area of substantial disagreement, the notion of two "swim lanes" was enthusiastically adopted at the workshop.

The consequence for programming models is that the baseline expression of parallelism will require 1 billion-way parallelism to achieve an exaflop if a 1 GHz clock-rate is used. The additional hardware threading required to hide latency will increase the amount of parallelism by a factor of 10-100x, depending on which swim lane you follow!

FPU Organization: Floating point used to be the most costly component of a system both in terms of design cost and power. However, today, FPUs consume a very small fraction of the area of a modern chip design and a much smaller fraction of the power consumption. On modern systems, a double-precision FMA (fused multiply add) consumes 100picojoules. An FPU is 0.02 square mm/FMA (400square mm chip). By 2018 it will consume about ~10.6pJ/op on 11nm lithography technology.

[3] Proof that in equilibrium the number of tasks in a system is equal to the arrival rate x the response time.

In order to reduce failure rates and component counts, it is desirable to build a system that reduces the total number of nodes by maximizing the performance of each node. Putting 10,000 FPUs on a chip would only cost 100 watts in this timeframe, and is entirely reasonable in terms of area and power consumption. However supplying memory bandwidth and capacity to a 10 Teraflop chip is the primary barrier to this design point. Without advanced packaging technology and substantial improvements in DRAM interface energy efficiency, the upper limit for per-chip performance will likely be 1-2 Teraflops/chip.

There was disagreement on whether 1 or 10 Teraflops would be achievable in the 2018 timeframe. So we have two design points to represent this divergence.

swim lane 1: 1k FPUs per chip
swim lane 2: 10k FPUs per chip

To support full floating point performance, the on-chip register file bandwidth would need to supply 24 bytes per op. Therefore, for a 10 Teraflops/chip * 24 → 640TB/s of register file bandwidth and 64TB/s register file bandwidth for a 1TF chip. The upper limit of feasible off-chip memory bandwidth will be 4TB/s. Therefore, the design point for swim lane 2 would require O(100) data reuse on chip and the design point for swim lane 1 would require O(10) data reuse on chip if a 4TB/s memory interface was used. In both cases, the assumed quantity of on-chip memory is on the order of 0.5-1GB/chip, so all temporal recurrences necessary to achieve on-chip data reuse would need to be captured within this memory footprint.

For node organizations that use more than one chip for a node, the bandwidth would likely be more on the order of 0.5 to 1TB/s to remote DRAM (1/4 to 1/8 of local DRAM BW). Therefore, NUMA effects on a multi-chip node will have a substantial performance impact.

System on Chip (SoC) Integration: To reduce power, and improve reliability it is useful to minimize off-chip I/O by integrating peripheral functions, such as network interfaces and memory controllers, directly onto the chip that contains the CPUs. There are fringe benefits, such as having the communication adaptor be TLB-coherent with the processing elements, which eliminates the need for expensive memory pinning or replicated page tables that is required for current high-performance messaging layers. It also reduces exposure to hard-errors caused by mechanical failure of solder joints. From a packaging standpoint, the node design can be reduced to a single chip surrounded by stacked memory packages, which increases system density. There was broad agreement that SoC integration would play an increasingly important role in future HPC node designs.

Other Functional Unit (FU) organizations:

Accelerators and Heterogenous Multicore Processors: Accelerators and heterogeneous processing offer some opportunity to greatly increase computational performance within a fixed power budget, while still retaining conventional processors to manage more general-purpose components of the computation such as OS services. Currently, such accelerators have disjoint memory spaces that are at the other end of a PCIe interface, which makes programming them very difficult.

There is a desire to have these accelerators fully integrated with the host processors memory space. At low end, accelerators already are integrated in a unified memory space, but at high end they are not because of differences in the memory technology used for the accelerator and the host processor. There was general agreement that by 2015 it will be feasible from a market

standpoint to integrate scalar cores with accelerator (and not copy data around). This was true for NVIDIA GPU solutions and possibly for heterogeneous manycore architectures like Larrabee.

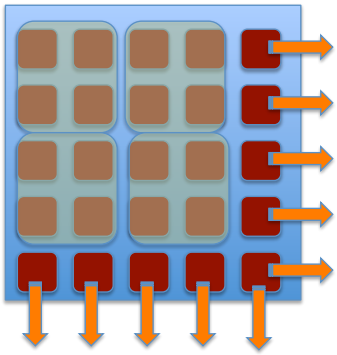
FPGAs and Application-Specific Accelerators: Application specific functional unit organizations may need to be considered in order to tailor computation and power utilization profiles to more closely match application requirements. However, the scope of such systems may be limited and therefore impact the cost-effectiveness of the resulting system design. FPGAs enable application-tailored logic to be created on the fly, but are currently too expensive. Otherwise, FPGA's could be used to implement application-specific primitives were this not the case. So the barriers to using FPGAs as a solution are primarily limited by the cost of the technology.

One view at the workshop was that there is some evidence that power considerations will force system architects to rely on application-tailored processor designs in the 2020 timeframe. One example of such design specialization can be found in the GPU, which derives some of its original performance benefit from tailoring to graphic requirements. The same is true from the embedded/handheld electronics space that is built around the concept of design specialization. However, the fixed (NRE) costs of design and verification for specialization remain high for full custom design, so specialization would need to be targeted judiciously. Economics will likely constrain the number of application tailored processor designs to a small number and the high performance computing marketplace may not be of sufficient size to warrant its own application-tailored processor. There was also discussion of whether we could we create a non-profit model for sustaining a processor design team focused on high performance computing over several processor design cycles? There was agreement that this would only succeed if the design effort were heavily leveraged from other multi-billion dollar markets.

On-Chip Memory/Cache Hierarchy

Levels of Cache Hierarchy: There was general agreement that there will be 2-4-levels of on-chip hierarchy that can be managed explicitly or flipped to implicit state. The reason for a multi-level hierarchy is mostly governed by the cost of data movement across the chip. Moving data 1mm across the chip costs far less than a floating-point operation, but movement of 20mm (to the other end of the chip) cost substantially more than a floating-point operation. So the computation and memory hierarchy on the chip will likely be grouped into clusters or hierarchies of some form to try to exploit spatial locality of data accesses.

There will need to be more effort to create Hardware Block Transfer support to copy things between levels of the memory hierarchy with gather/scatter (multi-level DMA).



- Cost of moving long-distances on chip motivates clustering on-chip
  - 1mm costs ~6pj (today & 2018)
  - 20mm costs ~120 pj (today & 2018)
  - FLOP costs ~100pj today
  - FLOP costs ~25pj in 2018
- Different Architectural Directions
  - GPU: WARPs of hardware threads clustered around shared register file
  - CMP: limited area cache-coherence
  - CMT: hardware multithreading clusters

Figure 3.8 Processor cores or functional units will likely be organized into groups or a hierarchy in order to exploit spatial locality of data accesses.

Private vs. shared caches: Most codes make no use of cache coherence and thus it is an overhead. So it is likely the cache hierarchy will be organized to put most of the on-chip memory into private cache. Performance analysis says less sharing is best (i.e., code written in threads to look like MPI or shared generally performs better).

Explicitly managed caches vs. conventional caches: Automatically managed caches virtualize the notion of on-chip and off-chip memory, and are therefore invisible to current programming models. However, the cost of moving data off-chip is so substantial, that virtualizing data location in this manner wastes energy and substantially reduces performance. Therefore, there has been increasing interest in explicit software management of memory, such as the Local-stores used by the STI Cell processor and by GPUs. Over the next decade, explicitly managed on-chip memory will become mainstream in conventional CPU designs as well.

However, we have not found the right abstraction for exposing software-controlled memories in our existing programming models. To support an incremental porting path for existing applications, these explicitly managed memory hierarchies will need to exist side-by-side with conventional automatically managed caches. These software-managed caches may depend on being able to switch dynamically from automatically managed caches to software-managed caches (convert ways in set associative cache). Switchable caches are already demonstrated in the Fermi GPUs, but will likely be seen in conventional multicore architectures as well.

When data is placed into an explicitly controlled cache, it can be globally visible to other processors on the chip, but cannot be visible to the cache-coherence protocol. Therefore, if the path to higher performance involves keeping more data in these explicitly managed caches, then it means cache-coherence (and the notion of an SMP with it) cannot be part of the high-performance path. Programming language designers must consider how to enable expression of on-chip parallelism without SMP/cache-coherent model.

Intra-node Communication

The primary area of growth in parallelism is explicit parallelism on-chip. Whereas the number of nodes in an exascale system is expected to grow by a factor of 10x over the next decade, the parallelism on-chip is expected to grow by a factor of 100x. This requires reconsideration of on-chip organization of CPU cores, and the semantics of inter-processor communication.

Cache Coherence (or lack thereof): It is likely that cache-coherence strategies can scale to dozens of processing elements, but the cost and latency of data movement on chip would make cache-coherence an inefficient method for interprocessor communication for future chip designs. In all likelihood cache-coherence could be used effectively in clusters or sub-domains of the chip, but is unlikely be effective if extended across a chip containing thousands of cores. See Figure 3.8. It is more likely that global memory addressing without cache-coherence will be supported with synchronization primitives to explicitly manage memory consistency.

Global addressing: PGAS programming models, including the HPCS programming languages benefit from Global Addressing to ensure a compact way to reference remote memory. PGAS models are willing to accept global addressing without SMP cache-coherence on the node. Therefore, there will likely be support for incoherent global addressing for small-scale systems, but will require hardware investment to scale this to larger systems. There was substantial disagreement on how many address bits will be supported in mainstream implementation. From a technology standpoint, it is entirely feasible to support global addressing within context of Exascale. However, larger scale global addressing schemes will not naturally occur without

investment. Global addressing only makes sense with hardware support for sync, which is also investment dependent.

Fine Grained Synchronization Features: The programming models group requested much finer-grained synchronization features that could directly map to programming language primitives. These features could greatly improve the efficiency of fine-grained on-chip parallelism.

One option discussed involved moving atomic memory operations (Amos) to memory controllers and full empty bits on-chip. Moving atomics as close to memory as makes sense from a power and performance standpoint, but would force us to give up some temporal recurrences since the data operated on by the atomics would not pass through the cache hierarchy.

An alternative approach to supporting these atomics is to use an intermediate level of the memory hierarchy where synchronization constructs get enforced/resolved. For example, you could imagine an L2 cache on-chip that is specifically dedicated to fine-grained inter-processor synchronization and atomic memory operations. This approach would potentially encode synchronization state information or other coordinating state using the ECC words of the memory system, because cannot hold it in proc. All of these options seemed feasible, but would require close interaction with application developers and programming model design to determine which approach would be most effective.

Hardware Support for Fault Tolerance

The terminology for describing error sources and mitigations is described in more detail in the Resilience section of this document. Here we will focus on what can be done in the node architecture to assist with keeping error rates under control and to support practical strategies for error recovery. The described features are not sufficient to solve all resilience issues, but provide some notion of what techniques hardware architects can bring to the table to mitigate these errors.

Redundancy and SoC to mitigate Permanent Errors: Permanent (hard) errors depend on a different mitigation strategy than transient errors. Permanent hardware errors might be partly accommodated by incorporating redundant or spare components. For example, it is straightforward for system architects to build extra cores into a processor chip that can be pressed into service to replace any failed processors on chip. This is already done for the 188-core Cisco Metro chip, which contains eight additional cores for redundancy. Likewise, the consumer version of the Cell chip only exposes seven cores, and keeps the eighth core as a spare to tolerate manufacturing defects that result in permanent errors.

System on Chip designs, described in the Node Architecture section above, can greatly reduce the hard-error rate by reducing the number of discrete chips in the system. Both sockets and solder-joints are a large source of hard-failures – both of which are minimized if all peripheral components are integrated onto a single chip. This approach has been employed successfully on IBM Blue Gene systems to achieve a 10-15x lower hard-error rate than conventional clusters.

Node Localized Checkpointing for Tolerance of Transient Errors: It is clear that current checkpointing approaches that save the complete memory image of a job to the shared filesystem are not going to scale to exascale. This led to consideration of localized/buddy-system checkpointing approaches, such as LLNL's Scalable Checkpoint Restart (SCR), that checkpoint

state to local nonvolatile storage and to the storage of a neighboring node to enable fault recovery. Localized checkpointing to node-integrated non-volatile storage can accommodate a 2.5-hour mean time before application failure (ATBAF), but failure characteristics of nonvolatile node-localized storage must be far better than current commodity parts would support. Using increased redundancy and extensions to Reed-Solomon error correction encodings could make high-volume commodity NVRAM components suitable for node-localized checkpointing.

3.6 Power Management Strategies

Thermally limited designs forces 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 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 underutilized 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.

Currently, changes between power modes take many clock-cycles to take effect. In a practical application code that contains many solvers, the power modes cannot switch fast enough to be of use. Vendors present at the meeting felt that technology that would enable power management systems to switch to low-power modes within a single clock cycle. However, there is still a lot of work required to coordinate switching across a large-scale HPC system.

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 handcrafted (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 global 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.

Node-scale Power Management: Operating systems must support Quality-of-Service (QOS) 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 heterogeneous computing resources, and manage data movement and locality in memory hierarchy.

System-Scale Power 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.

Energy Aware Algorithms*:* There was an observation that algorithms must base order of complexity on energy cost of operations rather than temporal cost of operations. A good example of this approach is communication-avoiding algorithms, which trade-off FLOPS for communication to save energy. Modifying complexity theory to accommodate the cost of data movement presents its own set of challenges because FLOP/s are relatively invariant whereas modeling of distance dependent data movement is very dependent on ephemeral system architectural and physical characteristics.

Incorporating data movement cost into complexity theory would require a substantially more sophisticated framework. 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. There is much

opportunity to develop energy-aware algorithms, but the mathematics for expressing these costs has a long way to go.

Library Integration with Power Management Systems*:* 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 as well as supporting portability across different machines.

Compiler Assisted Power Management*:* 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.

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

System "Aging": Today's systems operate with clock rates and voltages in guard bands to account for chip "wear-out." By employing slight clock speed reduction over the lifetime of the system, can achieve 5% power savings instead of using guard bands to account for silicon aging effects.

Voltage Conversion and Cooling Management: Another key area for power reduction is to design hardware to minimize the losses in voltage regulation and power conversion components. For example, the D.E. Shaw system had 30% efficiency loss just from the power conversion stages going from 480V to lowest voltage level delivered to chips.

There are opportunities to use smart-grid strategies to reduce energy consumption. Improve data center efficiencies (5-10% savings in total power consumption) have been demonstrated using this approach. The smart grid technology can rapidly shift power distribution to balance power utilization across the system.

Exascale systems should be water cooled (some may be warm water cooled) because it is substantially more efficient that air-cooling.

3.7 Priority Technology Investments for Next Decade

Addressing the technology challenges discussed in this report and accelerating the pace of technology development will require focused investments to achieve an exascale system by 2018. Areas requiring early investments (long-lead times) are identified.

Architecture

Node software/hardware architecture (Long-lead time). Support for hierarchical memory structure, active messages, sync primitives, requires close interaction with software architects. Architectural emulators will be needed support language development.

System Level Simulation and Emulation Capability to enable Hardware/Software Co-Design (Long-lead time): Evaluating different hardware architecture options requires a robust and credible hardware/architectural simulation platform. Without such a platform, it will be

difficult to integrate application science groups into the design process for future node architectures without some way for them to run their codes to assess the benefits (and costs) of various architecture trade-offs. Verification and validation of these capabilities will be a critical issue.

Interprocessor Communication Primitives: We need to rigorously define the new architectural semantics for inter-processor communication to support future programming models. There is no hope of coming up with a common multi-platform programming environment if there is no agreement on the hardware-level support for a common set of semantic primitives.

Mechanisms for Dealing with 10^9 FPUs: Simply enumerating the communication with 10^9 partners would likely exhaust local memory on future node designs. Either hardware mechanisms that enable compact addressing (global address space) or some other compact approach to referencing remote communication partners will be required.

Tools for Application-Driven Hardware Design: The physical design constraints that lead to million-processor systems and constrained bandwidth constrain the space of viable system characteristics (speeds and feeds), but there are still many viable options in terms of computer architecture and organization to make these systems more usable. Given the exceeding complexity of any viable hardware solution in the future, the architecture of future systems must be co-designed from programmers' point of view as well as from the hardware point of view.

Hardware

Architectural and Chip Level Circuitry Mechanisms for Dealing with Resiliency: The ability to detect errors is as important as the ability to recover from them. Novel software mechanisms for detecting silent errors and recovering from transient failures will not emerge without some hardware interfaces for software to exploit.

New checkpointing mechanisms (Long-lead time): Node-localized non-volatile memory will be essential to enable continued scaling of application-based checkpointing out to exascale class systems. Checkpointing directly to a shared disk is out of the question. It is viable to checkpoint to non-volatile memory (FLASH or its replacement) on node and to a partner node to enable rapid state preservation and restart. However, the NVRAM technology packaging, durability, and cost must be improved substantially to make this approach feasible. Software technology to manage node-local NVRAM will also be a requirement. Another area of interest is hierarchical checkpointing. Developing new checkpointing mechanisms is a long-lead time investment and will be essential to the success of the 2015 systems as well.

Advanced Low-Power Memory Technology (Long-lead time): Investment in new memory interface protocols could achieve a 5x power reduction from JEDEC memory roadmap by 2018. The biggest show-stopper for an exascale computing system is the power consumption of the memory subsystem, so advanced memory technology can enable much better balance between floating point performance and memory bandwidth, which in turn expands the scope and applicability of exascale-class systems. This is a long-lead time investment that requires tens of millions of dollars to get below 5 PJ/bit.

Optical Transceivers (Long-lead time): Interconnect architecture, signaling, optical (e.g., low cost lasers/modulators) are currently used for long-haul connections, but will be increasingly important for node-level and rack-level connections. Without investment the technology may

not be available at an appropriate cost-point or packaging to be useful for an exascale-class system. This is a long-lead time investment that requires tens of millions of dollars. Ring resonators and low power lasers are areas of interest.

Hardware/software mechanisms for power management (Long-lead time): One area of interest is over-provisioning FPUs and bandwidth (both at same time would not be possible due to other constraints so we need to quickly switch between the two). Another area is to use rapid power/performance changes to change the power-state within a single clock cycle.

3.8 Major Risk Factors and Strategies for Mitigation

The following is a brief list of the risk factors and mitigation strategies discussed:

- Will applications be able to expose sufficient concurrency to exploit an exascale system?
- Will resilience challenges limit effective use of an exascale system?
- Will the 2018 exascale systems have broad-enough market base to be commercially viable?
- Will the required optical technology be broadly available (affordable)?
- Applications need a stable abstract machine model (execution model) before they will begin to move. Applications won't develop exclusively for exascale machine. If an exascale node architecture diverges substantially from mainstream, it will have very limited impact and few applications.
- Failing to design to sufficiently large scale may result in systems that work well in the midrange, but are impractical to integrate and scale-up to target scale.
- Final system does not any meet science objectives due to overly constrained bandwidth or overly complicated programming environment.
- Inadequate time for applications to adapt to changes in programming models because early prototypes will not be available in time.
- Can't afford system in 2018 if > $200-300M/system.
- If we don't make the investment, we will run the risk of stagnating simulation based science development in the US.

Systems	2009	2018 swim lane 1	2018 swim lane 2
System peak	**2 Peta**	**≥1 Exa**	
Power	**6 MW**	**≤20 MW**	
System memory	0.3 PB	50 PB	
Node performance	125 GF	1TF	10TF
Interconnect Latency (for longest path)	1-5usec	0.5usec (speed of light)	
Memory Latency	150-250 clock cycles (~70-100ns)	100 clock cycles (~50ns)	
Node memory BW (consistent with 0.4 B/F)	25 GB/s	400 GB/s	4TB/s
Node concurrency (FPUs)	12	1.000	10,000
Node Interconnect BW (consistent with 0.1 B/F)	3.5 GB/s	100 GB/s	1 TB/s
System size (nodes)	18,700	1,000,000	100,000
Total concurrency	225,000	1B*10 for latency hiding	1B *100 for latency hiding
Storage	15 PB	1000 PB (>10x system memory is min)	
IO	0.2 TB	60 TB/s	
MTBAI (mean time between application interrupt)	Days	24 Hours	

Figure 3.8 Overview of technology scaling for exascale systems. Swim lane 1 represents an extrapolation of manycore system design point whereas swim lane 2 represents scaling of a GPU design point.

Chapter 4: Scalability and Concurrency

The discussions here are centered on scalability and concurrency driven by characteristics within the node as well as scalability and concurrency driven by the interconnect. There was general agreement on what two types of nodes might be available in this timeframe, their scalability ramifications, and programming approaches that would maximize their utilization. There was also general agreement on what the interconnect characteristics would be in this timeframe and the associated ramifications to code scalability. We discuss the need for co-design to address application scalability from both I/O and OS perspectives, and identify the need for testbeds, early prototypes, simulators and other resources that will help the community make sustained progress. Finally, we identify major risk factors associated with scalability and concurrency and risk mitigation strategies; these include where major investments could be prioritized and where long lead-time R&D could be identified to reduce risk.

A compute node is the end point of the interconnect network (with a NIC) and smallest replicable unit on that network. It is assumed here that all compute nodes (per job step) are the same, that a node will most likely have persistent storage such as FLASH or storage class memory, and that compute nodes are likely to be heterogeneous (e.g., CPUs and GPUs). Also assumed here is that I/O nodes can be different, that is, direct connections to I/O infrastructure and some have more memory; and Login and Service nodes can be different, that is, some have differences in OS.

Our application view of the memory system inside the node includes stacked memory and non-volatile memory. Options here are:

- All CPU/GPUs have separate memory that is explicitly managed
- Current model: cache hierarchy
- Private local scratchpad
- Future: shared memory space for the node (non-coherent)

Our application view of the memory system from inside the node looking out includes a view to the global (off-node) address space. Options here are:

- MPI view with one sided communications
- PGAS options: e.g., UPC, CAF, Global Arrays
- RDMA
- Coherence Model – answered by the programming models group, informed by the scalability and concurrency group

4.1 Node Concurrency

Inside a node there will be arithmetic logic units (ALU), a modest number of compute thread slots, a massive number of data parallel thread slots and that these slots will be schedulable. The OS image for the node must also be understood (discussed later). It is unclear whether there will be multiple coherence domains in the node versus only one. Data parallel concurrency seems to imply that we need to think about differences in thread definitions and their characteristics: compute threads versus data threads. Concurrency per node will be driven by the number of FPUs per chip, the number of threads per ALU, and the instruction bundle size. Total system concurrency will be driven by the number of nodes needed to reach an exaflop running at the expected frequency times the concurrency per node. Two example paradigms (or “swim lanes”) emerged. See Figure 3.8.

4.2 Interconnect Scalability

If we assume 0.1 B/F in each direction, then aggregate network requirement at the node boundaries has been set to 100 PB/s for the exascale machine in 2018. For a direct network this will be 100GB/s in each direction for a 1M node configuration. For the purposes of this discussion it is assumed that half of the bandwidth will be carried in optics and the other half is carried in copper. To first order it really doesn't matter how this bandwidth is divided in terms of node count. 1M nodes with 100 GB/s or 100K nodes with 1000 GB/s are likely to have the same amount of overall optics cost if there is a fixed bandwidth per optical connection. (If some multi-stage network is assumed for this size system the amount of optics will grow by O(N) for N stages.) Given today's projections of the cost of optics in 2018 of $.65-.85/Gb/s just the 50PB/s (plus encoding and 20% redundancy for reliability) would cost $400-500M. It is this overwhelming cost that drives the conclusion that extensive long-term investment in optics technology is required. The requirements for intra-machine optics are likely to be different enough from the telecommunications industry requirements that we cannot count solely on the industry growth and we must augment it with HPC specific investment.

The exact topology of the interconnection is an excellent topic for co-design since communications patterns depend on algorithmic decisions. There are classes of communications that are nearest neighbors and classes of communications that are more global. Given the increased scale of the system the balances in the system may change how the various algorithmic structures are actually coded. Some of these balance changes were topics of the workgroup as discussed below.

It appears that the single thread performance is not likely to be very different than today, so if nothing dramatic is done to the software communication paths the latency in the network is likely to be the same as today. Given the O(1000) increase in threads global synchronization costs will grow substantially, and will limit the overall scalability of algorithms structured the way they are today. These scalability concerns, among other things, lead us to recommend a shift to more asynchronous algorithm structuring in another section.

It is obvious that there will be usages of systems of this scale that do not dedicate the entire machine to one job, but will involve several independent jobs running concurrently. For this usage pattern it is very beneficial to have the capability to partition the network and the traffic such that interference is avoided. This applies not just to the algorithm communications but also to I/O, and to a lesser extent control information.

With the extreme number of endpoints in the system, and the massive amount of communications traffic, it is clear that some form of end-to-end reliability must be included. This can be a combination of hardware and software but should not require the application software itself to be the source of the reliability. Part of the investment in the optics recommended in the first paragraph should be to reduce as much as practical the raw bit error rates in the network itself.

4.3 Co-Design Opportunities

For most of the last two decades, the high-end computing platforms used by computational scientists presented them with the same model of execution, communicating sequential processes. Standard programming languages and libraries provided stability and portability. While exceptions abound, the principal strategy scientists used for exploiting the ever larger and more powerful systems was scaled-speedup. That is, as the machines got larger, so did the problems they were asked to solve. However, aspects of the execution model such as the balance

of operations and memory per process remained largely unchanged, and this enabled an increasingly broad set of high-end parallel applications to come into being.

The end of Dennard scaling[4] at the turn of the century has brought this period of stability to an end. Clock rates have stopped increasing, which implies that increased performance can only be achieved through increased concurrency or specialized architecture. The mainstream microprocessors that are leveraged to create most of today's high-end systems have multiple processor cores per die, and the number of such cores is expected to increase exponentially. New architectures are being developed for other niches such as gaming and graphics. These are being incorporated into high-end systems, creating heterogeneous processing nodes. The rate of change in DRAM density is falling off of Moore's Law, and it soon may stop altogether. As the number of compute cores continues to increase, this will cause a dramatic change in the traditional balance between operations and memory density. NAND Flash development now paces commercial solid-state memory technology, and these devices are beginning to appear in systems, both as solid-state disk (SSD) and non-volatile memory. It's not at all clear how systems and applications will exploit them.

The changes touched on above and elaborated on elsewhere in this document will make it increasingly difficult for a broad range of computational scientists to use the most powerful computing systems. This is not a new phenomenon. Many applications have had difficulty adapting to the distributed memory, messaging passing systems that predominate HPC today. This will only get worse as the rate of growth in concurrency will be greater in the next decade than it was in the last two. By the time we reach exascale, applications may have to have O(10^10) independent threads to fully exploit the largest systems. Amdahl's Law teaches us that even the smallest unnecessary overhead, be it in our algorithms or in the systems, will have devastating consequences for the throughput of these codes and hence the pace of discovery in their respective fields. The sluggish growth in memory volume will also be troublesome. Scaled-speedup will be increasingly difficult to achieve, and familiar techniques like ghost cells to buffer data exchange between processors may no longer be feasible for many problems.

All of the above emphasizes the risk that high-end systems may be less and less general purpose in the future, that such systems may be tailored to meet the needs of one or more related scientific disciplines. On the other hand, it is also clear that the exascale node will in large part be the same technology as that in transport container-scale systems, in rack-scale systems, in desktop systems, in laptops and hand-held devices.

Therefore, it is quite likely that a decade from now, high-end systems will have evolved significantly from today's systems, and applications will have to follow suit. For example, we may have to redesign or abandon Krylov-space iterative methods, such as conjugate gradients, since they involve inner-products that introduce a logarithmic computational bottleneck. Scientists and mathematicians must know this as soon as possible, as developing new algorithms can be a high risk and time consuming endeavor. Conversely, where new algorithms are not anticipated, systems will have to accommodate them, perhaps with specialized combining networks to process inner-products. This suggests that computational scientists and computer architects must sit down together to understand the complete range of tradeoffs possible for each of them, and then co-design their codes and systems to maximize scientific throughput.

The co-design of future systems and applications will be an expensive process. To manage this, system designers will minimize unique designs, and wherever possible leverage components

[4] The realization that as transistors get smaller, they can switch faster and use less energy (1974)

produced for other purposes. Therefore, even if the integrated circuit dies are unique to a family of HEC systems, they will be integrated from off the shelf IP modules. Software development is also a time consuming and labor intensive process. To the extent possible, it is critical for exascale systems to be backward compatible, allowing existing software to be reused where possible. New features of HEC systems, such as global address spaces must evolve into the execution model with as little disruption as possible. A new generation of tools (compilers, performance analysis, etc.) must be provided to help users generate many orders-of-magnitude more concurrency than in today's calculations, and to express this t the system so it can exploit them.

As DOE charts its path from today to exascale, it should do so in concert with other organizations, both foreign and domestic, that are trying to do the same. The best such example will likely be DARPA's Ubiquitous High Performance Computing (UHPC) program, which will focus on many of the same core technology issues such as power density, memory hierarchy, communication latency and bandwidth, resiliency, etc. DOE should manage its investments so as leverage DARPA's and those of other organizations that also have HEC research activities.

In this vein we discuss two areas where we believe co-design will play an important role, application scalability with file system support and application scalability with operating system support.

4.4 Application Scalability with File System Support

Traditionally, parallel file systems play a critical role in leadership class machines for simulation science because these machines protect computation progress in the face of frequent machine component failure by periodically recording computation state in disk-based checkpoint files. After a failed component is isolated out of the machine a computation can be restarted from the stored checkpoint file. For the machine to effectively advance science, rather than simply defend itself against failures, the fraction of time spent capturing checkpoints should be keep low, typically at or under 10%.

The size of a checkpoint file scales with machine memory, so each will be up to O(PB). Our target is 24 hours mean time to application failure. Thus, a parallel file system must capture checkpoints very fast, estimated at on the order of 60 TB/sec, or far too much of the machine's valuable time will be spent taking these checkpoints. This is an increase in parallel file system data rate of O(300X). But magnetic disk bandwidth grows slowly and is only expected to increase by 4X in the time until Exascale machines appear, so exascale systems would need O(80X) increase in the number of disks. Disk prices are not expected to drop significantly, although their capacity is expected to increase by O(16X), so meeting exascale bandwidth demands might increase storage system costs by O(80X), an unacceptable growth in cost.

Finding a cost effective fault tolerance strategy is a critical challenge for exascale systems. The size of checkpoint storage should be at least 30X main memory size to accommodate a sufficient number of checkpoints. This makes use of main memory technology like DRAM unacceptable for checkpoint storage, because even a single copy of main memory is expected to cost about half of the total cost of the system. Solid state memory technologies such as NAND flash will be perhaps an order of magnitude less expensive per byte than DRAM, but even this will be far too expensive to provide a checkpoint storage 30X the size of main memory.

Solid state memory, however, is expected to be cost effective for bandwidth, and for one copy of main memory. And disk is expected to remain cost effective for capacity. So the leading proposal for fault tolerance in exascale systems is to evolve parallel file systems to include a tier of solid state storage; checkpoints would be written at exascale speeds out of the compute system

into the solid state storage tier and during the time before the next checkpoint is taken, copied into the disk storage. Because the time between checkpoints is at least 10X the time to take a checkpoint, the checkpoint bandwidth to disk storage can be 10X lower than that into solid state disk, or O(6 TB/s).

This group is cautiously optimistic that this hybrid model for secondary storage will have appropriate cost structure, and that parallel file systems software can be evolved to integrate this double copy mechanism for checkpoint/restart. Because of the criticality of checkpoint/restart to effective use of exascale systems, however, we recommend aggressive exploration of hybrid secondary storage and the file system software changes needed to exploit it.

This fault tolerance strategy applies only to applications running on the exascale computer. It is not a solution to faults in the storage system, because it relies on the storage system to be constantly available and reliable. However, the storage system component count is expected to rise by at least O(10X), and with disk capacity rising by about 4X, the failed disk rebuild work will grow by O(40X), so the internal fault tolerance of the storage system, today dependent on RAID hardware in the components, will have to be revised.

I/O stream concurrency, today on O(10K), could increase to a stream per core, or O(1B), with Exascale systems. Synchronization and concurrency management in the parallel file system is likely to strangle bandwidth scaling. Systems that coalesce streams will greatly reduce the stress on parallel file system concurrency control. Coalescing at the level of the node might limit stream concurrency to O(1M), however even that is very challenging for parallel file systems. System call forwarding, with I/O nodes integrating streams from multiple compute nodes, has the potential to bring parallel file system concurrency down to current levels. This technology is being pursued today, and its timely success is important to exascale systems.

Coalescing can deal with stream management; however, if all nodes are concurrently writing small strided ranges to shared files, even coalescing will not overcome the false sharing collisions of O(1M) streams on file blocking structures. Various current projects are exploring delayed integration of concurrent write structures, such as log-structuring, to minimize synchronization bottlenecks. Techniques such as these must be tested and deployed at petascale in order to avoid aggravating these concurrency challenges at exascale.

Beyond the critical defensive I/O workload, exascale systems will be increasingly applying new workload patterns to parallel storage systems.

First, data analysis will increasingly be in-situ, or co-located with simulation. Data analysis gathers subsets of information from the simulation results or a single or a series of time steps, inducing either a very large read bandwidth load, comparable to defensive I/O bandwidths, if all output is read to find the subset of data, or apparently random read access patterns, if only the needed data is fetched. Parallel file systems on exascale systems will increasingly be required to support significantly heavier read and random read workloads. One especially challenging format for data analysis will be data capture workloads generating large numbers of small files. Estimates put the number of small files in exascale storage systems at O(1B) to O(1T), with corresponding high random access workloads. Current parallel file systems have been designed for high bandwidth on large objects, often at the expensive of small file and random access performance. This design simplification is going away as we approach exascale, and parallel file systems software structures will need to be revised extensively.

Second, the increasingly importance of uncertainty quantification (UQ), both through ensemble and imbedded model simulations, will place special emphasis on the data analysis implicit in UQ's response model generation. Concurrent analysis of simulation output, model

generation and adaptive generation of additional simulation runs will cause the parallel file system workload to see concurrent read and write accesses to an array of files. Concurrent read and write at high bandwidth and high concurrency has been a relatively non-critical workload in pre-petascale systems, and parallel file systems will need to be improved for this workload. Metadata embedded in stored objects may become a workload of importance as well, as persistent memory structures has been put forward as a tool for UQ systems.

4.5 Application Scalability with Operating System Support

Functional partitions, e.g., a compute partition, a login partition, and an I/O partition have been used since the earliest MPP systems and we believe that this partitioned strategy will continue to be used for the foreseeable future. We anticipate that an Exascale system will have at least four partitions: 1) a compute partition with approximately one million nodes, 2) a service and I/O partition with 500 to 5000 nodes, 3) a login partition (used to launch applications) with about 10 nodes, and 4) a system management partition that provides access to the RAS sub-system and has about 10 nodes. With the exception of the compute partition, all of the partitions will run full-featured operating systems (e.g., Linux). Even though these partitions will run the same basic OS, the OS will likely be configured and tuned to the specific needs of the partition. The nodes in the compute partition will run a "lightweight" OS which provides the minimal functionality needed to support computation.

Because of its size (~one million nodes), support for the compute partition will require the development of hierarchical infrastructure that provides support for collective OS operations involving all of the nodes in the compute partition, e.g., job launch, dynamic loading of libraries, performance monitoring and debugging. It is apparent that future runtime systems will want to take advantage of the information that is being collected by the RAS sub-system (e.g., availability of other compute nodes or availability of communication links). Moreover, runtime systems would likely benefit from an integration of the RAS sub-system and the hierarchical infrastructure.

The lightweight OS running on the compute nodes must include mechanisms that facilitate access to the extended functionality provided by other partitions. System call forwarding and/or proxy processes running on another partition can be used to provide the needed functionality while minimizing the additional OS state that needs to be maintained on the computes nodes. The compute node OS needs to support the aggregation of I/O streams in the compute partition based on packages like IOFSL (I/O Forwarding Scalability Layer). The compute node OS also needs to provide hooks for the hierarchical infrastructure used to manage the compute partition with an emphasis on the hooks needed to support scalable debuggers and performance monitoring tools.

Given the size of the compute partition, hardware and software jitter will have a significant impact on the time needed for system wide synchronization. This implies that block-synchronous programming models will not perform well on the entire system and programmers will opt for asynchronous programming models whenever possible. The compute node OS needs to provide good support for asynchronous programming models.

In the context f communication, the compute node OS needs to support programming models that require a communication endpoint for every other compute node in the system (~one million endpoints). This will require scalable support for communication startup, endpoint management, resource provisioning, and teardown. Beyond providing support for explicit programming models, the compute node OS needs to provide support for programming models that require

implicit communication between nodes, e.g., loads and stores over a (noncoherent) global address space.

The following resources will aid the community in making sustained progress:

1. Network simulators and network models for topology, routing and resiliency
2. Extended I/O testbeds
3. Early access to high-level system characteristics
4. Early access to high-level system models to explore trade-offs in the design space
    a. Advanced designs for node (with prototypes) for non-volatile memory
    b. S/W development
5. Early delivered machines to address asynchronous algorithm development and scalability

4.6. Major Risk Factors and Strategies for Risk Mitigation

The identified risks and risk mitigations (Risk/Risk Mitigation) are listed in priority order:

1. Cost of optics & transducers/investments
2. Communications latency/new paradigms
3. Numerical algorithms/co-development
4. HW & SW jitter/ asynchronous algorithms and communications
5. Interconnect BER/more BW to support higher order error correction
6. New storage paradigm/align with cloud computing community and testbeds
7. Random system characteristics impacts current validation techniques/statistical validation
8. Maintaining the pyramid investment model (trained workforce, S&T base)/education and outreach

Top priority should be emphasized with investment associated with scalability and concurrency. The top priority investments along with approximate level of investment are listed below in priority order:

1. Optics and transducers
2. Asynchronous communications and algorithms
3. Low overhead communication and latency HW/SW stack (combined with (1) above)
4. New non-volatile storage paradigm
5. New hierarchal system HW/SW architecture

Long lead time items requiring R&D investment:

- Optics and transducers
- Scalable numerical algorithms
    - Asynchronous, load-imbalance and fault tolerant
    - Larger surface/volume ratio
- New storage paradigm
    - Application defined objects, non-volatile, robust, reliable efficient

APPENDIX A: Workshop Agenda

Architectures and Technology for Extreme Scale Computing

December 8th - 10th, 2009 · San Diego, CA

Monday, December 7th, 2009

Time	Session	Lead	Room
6:30 - 8:00pm	Pre-Workshop Dinner (Organizers, Panel Leads, & Speakers)		Synergy
8:00pm	Adjourn		

Tuesday, December 8th, 2009

Time	Session	Lead	Room
7:30-8:30am	Working Breakfast: Registration		Foyer
Welcome and Plenary Talks:			
8:30-8:45am	Welcome, Logistics	Rick Stevens & Andrew White,	Inspire Ballroom
8:45 - 9:00am	Opening remarks from ASCR		
9:00 - 9:15am	Opening remarks from ASC		
9:15 - 10:15am	Overview of Scientific & Technical Applications	Rick Stevens & Andrew White	
10:15 am	General Discussion		Foyer
10:30 - 11:45am	Roadmap Presentation & Q&A session	Horst Simon or Sudip Dosanj	Inspire Ballroom
11:45 - 12:00pm	Stage setting: Purpose of panel sessions, review of agenda, plans for report	Rick Stevens & Andrew White	
12:00 - 1:15pm	Working Lunch: Preparation for breakout sessions		
1:15-3:30pm	**Breakout Sessions**		
	Node Architecture and Power Reduction Strategies	John Morrison, John Shalf, & Horst Simon	Synergy
	Programming models and environments	Gorda & Yelick	Convene 3
	Fault management and resiliency	Dosanjh & Geist	Convene 4
	Scalability and concurrency	Nichols & Seager	Convene 5
3:30pm	General Discussion		Foyer
3:45-6:00pm	**Continue Breakout Sessions (return to breakout rooms)**		
6:30 - 8:00pm	Working Dinner: Organizational meetings		Synergy
8:00pm	Adjourn		

Architectures and Technology for Extreme Scale Computing

December 8th - 10th, 2009 · San Diego, CA

Wednesday, December 9th, 2009

Time	Session	Lead	Room
7:30-8:30am	Working Breakfast: Preparation for breakouts		Foyer
8:30am-10:00am	**Breakout Sessions**		
	Node Architecture and Power Reduction Strategies	John Morrison, John Shalf, & Horst Simon	Synergy
	Programming models and environments	Gorda & Yelick	Convene 3
	Fault management and resiliency	Dosanjh & Geist	Convene 4
	Scalability and concurrency	Nichols & Seager	Convene 5
10:00am	General Discussion		Foyer
10:15 - 11:45am	**Continue Breakout Sessions** (return to breakout rooms)		
11:45 - 1:00pm	Working lunch: Organizational meetings		Foyer
Plenary Session:			
1:00 – 2:45pm	Presentations from Breakout Session Leaders		Inspire Ballroom
2:45 pm	General Discussion		Foyer
3:00 - 5:30pm	**Continue Breakout Sessions** (return to breakout rooms)		
6:00 - 7:30pm	Working Dinner: Organizational meetings		Synergy
7:30pm	Adjourn		

Thursday, December 10th, 2009

Time	Session	Lead	Room
7:30-8:30am	Working Breakfast: Preparation for Plenary		Foyer
Plenary Session:			
8:30 - 10:00am	Workshop Summary from each panel		Inspire Ballroom
10:00 am	General Discussion		Foyer
10:20 - 12:00pm	Report Writing Session for Chairs, Panel Leads, and Writers		Synergy
12:00 - 1:00pm	Working lunch: Report Writing Session Continued		Inspire Ballroom
1:00 - 4:00pm	Report Writing Session Continued		Inspire Ballroom
4:00pm	Adjourn		

APPENDIX B: Workshop Attendees

Architectures and Technology for Extreme Scale Computing

December 8 - 10, 2009 · San Diego, CA

Attendee List

First Name	Last Name	Email	Affiliation
Jim	Ang	jaang@sandia.gov	Sandia National Laboratories
Raymond	Bair	bair@mcs.anl.gov	Argonne National Laboratory
John	Baron	jbaron@sgi.com	Silicon Graphics Incorporated
Mark	Barrenechea	mark@sgi.com	Silicon Graphics Incorporated
David	Bernholdt	bernholdtde@ornl.gov	Oak Ridge National Laboratory
Mike	Black	mblack@micron.com	Micron Technology, Inc.
George	Bosilca	bosilca@eecs.utk.edu	University of Tennessee
Ron	Brightwell	rbbrigh@sandia.gov	Sandia National Laboratories
Greg	Bronevetsky	greg@bronevetsky.com	Lawrence Livermore National Laboratory
David	Brown	dlb@llnl.gov	Lawrence Livermore National Laboratory
William	Camp	william.j.camp@intel.com	Intel Corporation
Franck	Cappello	fci@lri.fr	INRIA & University of Illinois
William	Carlson	wwc@super.org	IDA Center for Computing Sciences
Brad	Chamberlain	bradc@cray.com	Cray, Inc.
Daniel	Chavarría	daniel.chavarria@pnl.gov	Pacific Northwest National Laboratory
Andrew	Chien	andrew.chien@intel.com	Intel Corporation
Alok	Choudhary	a-choudhary@northwestern.edu	Northwestern University
Giri	Chukkapalli	giri.chukkapalli@sun.com	Sun Microsystems
Paul	Cook	plc@sgi.com	Silicon Graphics Incorporated
Paul	Coteus	coteus@us.ibm.com	International Business Machines
William	Dally	dally@stanford.edu	Stanford University and NVIDIA
Tomas Diaz	De La Rubia	diazdelarubia1@llnl.gov	Lawrence Livermore National Laboratory
Marty	Deneroff	marty.deneroff@gmail.com	D.E. Shaw Research
Lori	Diachin	Diachin2@llnl.gov	Lawrence Livermore National Laboratory
David	Domyancic	domyancic1@llnl.gov	Lawrence Livermore National Laboratory
Sudip	Dosanj	ssdosan@sandia.gov	Sandia National Laboratories
Anshu	Dubey	adubey1@cs.uchicago.edu	University of Chicago
Efstratios	Efstathiadis	stratos@bnl.gov	Brookhaven National Laboratory
Elmootazbellah	Elnozahy	mootaz@us.ibm.com	International Business Machines
Christian	Engelmann	engelmannc@ornl.gov	Oak Ridge National Laboratory
Stephane	Ethier	ethier@pppl.gov	Princeton Plasma Physics Laboratory
John	Feo	john.feo@pnl.gov	Pacific Northwest National Laboratory
Guang	Gao	ggao@capsl.udel.edu	University of Delaware
Alan	Gara	alangara@us.ibm.com	International Business Machines
Al	Geist	gst@ornl.gov	Oak Ridge National Laboratory
Garth	Gibson	garth.gibson@cs.cmu.edu	Carnegie Mellon University
Roscoe	Giles	roscoe@bu.edu	Boston University
Roger	Golliver	roger.a.golliver@intel.com	Intel Corporation
Brent	Gorda	bgorda@llnl.gov	Lawrence Livermore National Laboratory
Steven	Gottlieb	sg@denali.physics.indiana.edu	Indiana University
Don	Grice	dgrice@us.ibm.com	International Business Machines
Gary	Grider	ggrider@lanl.gov	Los Alamos National Laboratory
Mary	Hall	mhall@cs.utah.edu	University of Utah
Robert	Harrison	harrisonrj@ornl.gov	Oak Ridge National Laboratory

Angela	Hatton	angela.hatton@pnl.gov	Pacific Northwest National Laboratory
Rajeeb	Hazra	rajeeb.hazra@intel.com	Intel Corporation
Barbara	Helland	barbara.helland@science.doe.gov	US Department of Energy, Office of Science
Daniel	Hitchcock	daniel.hitchcock@science.doe.gov	US Department of Energy, Office of Science
Thuc	Hoang	thuc.hoang@nnsa.doe.gov	National Nuclear Security Administration
Adolfy	Hoisie	hoisie@lanl.gov	Los Alamos National Laboratory
Paul	Hovland	hovland@mcs.anl.gov	Argonne National Laboratory
Rob	Jacob	jacob@mcs.anl.gov	Argonne National Laboratory
Fred	Johnson	fcj@acm.org	Science Applications International Corporation
Gary	Johnson	gmj@computationalsciencesolutions.com	Computational Science Solutions
Norm	Jouppi	norm.jouppi@hp.com	Hewlett Packard
Larry	Kaplan	lkaplan@cray.com	Cray, Inc.
Darren	Kerbyson	djk@lanl.gov	Los Alamos National Laboratory
Moe	Khaleel	moe.khaleel@pnl.gov	Pacific Northwest National Laboratory
Kenneth	Koch	krk@lanl.gov	Los Alamos National Laboratory
Donald	Lamb	lamb@oddjob.uchicago.edu	University of Chicago
Julien	Langou	julien.langou@ucdenver.edu	University of Colorado, Denver
Alan	Lee	alan.lee@amd.com	Advanced Micro Devices
Sander	Lee	sander.lee@nnsa.doe.gov	National Nuclear Security Administration
Bob	Lucas	rflucas@isi.edu	Information Sciences Institute, University of Southern California
Tina	Macaluso	antoinette.macaluso@saic.com	Science Applications International Corporation
Barney	Maccabe	maccabeab@ornl.gov	Oak Ridge National Laboratory
Chris	Maher	maherc@us.ibm.com	International Business Machines
Allen	Maloney	malony@cs.uoregon.edu	University of Oregon
Andrés	Marquez	andres.marquez@pnl.gov	Pacific Northwest National Laboratory
Michael	Mason	michael.mason@hp.com	Hewlett Packard
Michel	McCoy	mccoy2@llnl.gov	Lawrence Livermore National Laboratory
Bill	Menger	bmenger@fusiongeo.com	Fusion Petroleum Technologies, Inc.
Bronson	Messer	bronson@ornl.gov	Oak Ridge National Laboratory
Paul	Messina	messina@mcs.anl.gov	Argonne National Laboratory
George	Michaels	george.michaels@intel.com	Intel Corporation
Douglas	Miles	douglas.miles@pgroup.com	The Portland Group
Charles	Moore	chuck.moore@amd.com	Advanced Micro Devices
Gerald	Morris	gerald.r.morris@usace.army.mil	US Department of Defense
John	Morrison	jfm@lanl.gov	Los Alamos National Laboratory
Norman	Morse	normanmorse@gmail.com	Hewlett Packard
Richard	Murphy	rcmurph@sandia.gov	Sandia National Laboratories
Jeff	Nichols	nicholsja@ornl.gov	Oak Ridge National Laboratory
Kunle	Olukotun	kunle@csl.stanford.edu	Stanford University
Dhabaleswar	Panda	panda@cse.ohio-state.edu	Ohio State University
Greg	Papadopulos	greg@sun.com	Sun Microsystems
Philip M.	Papadopulos	phil@sdsc.edu	San Diego Supercomputer Center
Michael	Papka	papka@anl.gov	Argonne National Laboratory

Wilfred	Pinfold	wilfred.pinfold@intel.com	Intel Corporation
Steve	Poole	spoole@ornl.gov	Oak Ridge National Laboratory
Alex	Pothen	apothen@purdue.edu	Purdue University
Sridhar	Rajagopalan	sridhar.rajagopalan@hp.com	Hewlett Packard
Michael	Raymond	mraymond@sgi.com	Silicon Graphics Incorporated
Dave	Resnick	dresnick@micron.com	Micron Technology, Inc.
Katherine	Riley	riley@mcs.anl.gov	Argonne National Laboratory
Robert	Ross	rross@mcs.anl.gov	Argonne National Laboratory
Vivek	Sarkar	vsarkar@rice.edu	Rice University
Rob	Schreiber	rob.schreiber@hp.com	Hewlett Packard
Mark	Seager	seager@llnl.gov	Lawrence Livermore National Laboratory
Jim	Sexton	sextonj@us.ibm.com	International Business Machines
John	Shalf	jshalf@lbl.gov	Lawrence Berkeley National Laboratory
Allan	Snavely	allans@sdsc.edu	University of California, San Diego
Thomas	Sterling	tron@cct.lsu.edu	Louisiana State University
Rick	Stevens	stevens@anl.gov	Argonne National Laboratory
Fred	Streitz	streitz1@llnl.gov	Lawrence Livermore National Laboratory
Craig	Stunkel	stunkel@us.ibm.com	International Business Machines
Sriram	Swaminarayan	sriram@lanl.gov	Los Alamos National Laboratory
Rajeev	Thakur	thakur@mcs.anl.gov	Argonne National Laboratory
James	Tomkins	jltomkins@gmail.com	Tomkins Associates
Keith	Underwood	keith.d.underwood@intel.com	Intel Corporation
Lucy	Vannoy	lucy.vannoy@pnl.gov	Pacific Northwest National Laboratory
Jeffrey	Vetter	vetter@computer.org	Oak Ridge National Laboratory
Michael	Vildibill	mikev@sun.com	Sun Microsystems
Steve	Wallach	swallach@conveycomputer.com	Convey Computer
Cheryl	Wampler	clw@lanl.gov	Los Alamos National Laboratory
Nanbor	Wang	nanbor@txcorp.com	Tech-X Corporation
Andy	White	abw@lanl.gov	Los Alamos National Laboratory
John	Wu	kwu@lbl.gov	Lawrence Berkeley National Laboratory
Kathy	Yelick	kayelick@lbl.gov	Lawrence Berkeley National Laboratory
Dantong	Yu	dtyu@bnl.gov	Brookhaven National Laboratory