OAK RIDGE
NATIONAL LABORATORY

MANAGED BY UT-BATTELLE
FOR THE DEPARTMENT OF ENERGY

ORNL/TM-2007/238

Scientific Application Requirements for Leadership Computing at the Exascale

December 2007

Prepared by
Computing Requirements Team
National Center for Computational Sciences

UT-BATTELLE

ORNL-27 (4-00)

ORNL/TM-2007/238

SCIENTIFIC APPLICATION REQUIREMENTS FOR LEADERSHIP COMPUTING AT THE EXASCALE

Computing Requirements Team
Sean Ahern
Sadaf Alam
Mark Fahey
Rebecca Hartman-Baker
Richard Barrett
Ricky Kendall
Douglas Kothe
O.E. Messer
Richard Mills
Ramanan Sankaran
Arnold Tharrington
James B. White III

Date Published: December 2007

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6283
managed by
UT-BATTELLE, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

Page

LIST OF TABLES

EXECUTIVE SUMMARY

The Department of Energy's Leadership Computing Facility, located at Oak Ridge National Laboratory's National Center for Computational Sciences, recently polled scientific teams that had large allocations at the center in 2007, asking them to identify computational science requirements for future exascale systems (capable of an exaflop, or 10^{18} floating point operations per second). These requirements are necessarily speculative, since an exascale system will not be realized until the 2015–2020 timeframe, and are expressed where possible relative to a recent petascale requirements analysis of similar science applications [1].

Our initial findings, which beg further data collection, validation, and analysis, did in fact align with many of our expectations and existing petascale requirements, yet they also contained some surprises, complete with new challenges and opportunities. First and foremost, the breadth and depth of science prospects and benefits on an exascale computing system are striking. Without a doubt, they justify a large investment, even with its inherent risks. The possibilities for return on investment (by any measure) are too large to let us ignore this opportunity.

The software opportunities and challenges are enormous. In fact, as one notable computational scientist put it, the scale of questions being asked at the exascale is tremendous and the hardware has gotten way ahead of the software. We are in grave danger of failing because of a software crisis unless concerted investments and coordinating activities are undertaken to reduce and close this hardware-software gap over the next decade. Key to success will be a rigorous requirement for natural mapping of algorithms to hardware in a way that complements (rather than competes with) compilers and runtime systems. The level of abstraction must be raised, and more attention must be paid to functionalities and capabilities that incorporate intent into data structures, are aware of memory hierarchy, possess fault tolerance, exploit asynchronism, and are power-consumption aware. On the other hand, we must also provide application scientists with the ability to develop software without having to become experts in the computer science components.

Numerical algorithms are scattered broadly across science domains, with no one particular algorithm being ubiquitous and no one algorithm going unused. Structured grids and dense linear algebra continue to dominate, but other algorithm categories will become more common. A significant increase is projected for Monte Carlo algorithms, unstructured grids, sparse linear algebra, and particle methods, and a relative decrease foreseen in fast Fourier transforms. These projections reflect the expectation of much higher architecture concurrency and the resulting need for very high scalability. The new algorithm categories that application scientists expect to be increasingly important in the next decade include adaptive mesh refinement, implicit nonlinear systems, data assimilation, agent-based methods, parameter continuation, and optimization.

The attributes of leadership computing systems expected to increase most in priority over the next decade are (in order of importance) interconnect bandwidth, memory bandwidth, mean time to interrupt, memory latency, and interconnect latency. The attributes expected to decrease most in relative priority are disk latency, archival storage capacity, disk bandwidth, wide area network bandwidth, and local storage capacity. These choices by application developers reflect the expected needs of applications or the expected reality of available hardware. One interpretation is that the increasing priorities reflect the desire to increase computational efficiency to take advantage of increasing peak flops [floating point operations per second], while the decreasing priorities reflect the expectation that computational efficiency will not increase. Per-core requirements appear to be relatively static, while aggregate requirements will grow with the system. This projection is consistent with a relatively small increase in performance per core with a dramatic increase in the number of cores.

Leadership system software must face and overcome issues that will undoubtedly be exacerbated at the exascale. The operating system (OS) must be as unobtrusive as possible and possess more stability, reliability, and fault tolerance during application execution. As applications will be more likely at the exascale to experience loss of resources during an execution, the OS must mitigate such a loss with a

range of responses. New fault tolerance paradigms must be developed and integrated into applications. Just as application input and output must not be an afterthought in hardware design, job management, too, must not be an afterthought in system software design. Efficient scheduling of those resources will be a major obstacle faced by leadership computing centers at the exascale.

Leadership systems must evolve in the next decade into more capable and productive science-producing systems by applying strong software engineering philosophies to systems capable of good end-to-end computing. Users must be able to ask “what if” questions to help drive science to find the needle in the haystack, which will require a paradigm shift.

Data analytics will empower scientists to ask these “what-if” questions, providing software and hardware infrastructure capable of answering these questions in a timely fashion. Strong data management will not just become important; it will become an absolute at the exascale. These trends are already evident on the internet when we look at tools like Google desktop, which have begun to revolutionize desktop computing by allowing users to find information in files that were previously untapped and unknown. These technologies must move into leadership computing and must be encouraged to work on the largest analysis machines.

Just like computing hardware requires disruptive technologies for acceleration of natural evolutionary paths, so too will algorithm, software, and physical model development efforts need disruptive technologies to ensure accelerated science application development and readiness for the exascale. This document identifies several areas that need focused and enhanced research and development in order to provide disruptive technologies for science application. These include automated diagnostics, hardware latency, hierarchical algorithms, parallel programming models, solver technology and innovative solution techniques, accelerated time integration, model coupling, and middleware library performance.

After a brief introduction and summary of key science drivers, this document summarizes exascale application requirements and presents key requirements for application models and algorithms, application software, system software and hardware, and data analytics. It concludes with recommendations for those areas in need of increased and focused research and development investment to ensure the readiness of exascale applications.

1. INTRODUCTION AND SCIENCE DRIVERS

Requirements are conditions or capabilities needed by users to solve problems and achieve objectives. They are also conditions or capabilities that must be met or possessed by a system to satisfy a standard or specification. Both definitions apply for computational science requirements that are deemed essential for the design, procurement, deployment, and operation of leadership computing systems at the Department of Energy (DOE’s) Leadership Computing Facility (LCF), located at Oak Ridge National Laboratory’s National Center for Computational Sciences (NCCS). Eliciting, analyzing, validating, and managing these requirements is crucial for success, especially for leadership computing systems beyond the horizon (>3 years). Exascale computing systems (capable of an exaflop, or 10^{18} floating point operations per second) are more than three years away, but they expected to be available and deployed in the 2015–2020 timeframe. With a speculative eye toward exascale computing systems, the NCCS Scientific Computing Group has collected and performed an initial discerning analysis of leadership computational science requirements. Analyzed in this document are the following:

- The science objectives and impacts possible on exascale systems;
- The possibilities and plans for physical models and numerical algorithms possessed by the scientific applications;
- The software opportunities and challenges;
- Desirable attributes for system hardware and system software; and
- Key characteristics of data analytics and work flow required to ensure high scientific productivity.

A similar analysis was recently performed in 2006 for computational science requirements at the petascale (capable of a petaflop, or 10^{15} floating point operations per second) [1], where the focus was on the 1–3-year horizon and leadership systems in the 0.10–1.0 petaflop range. The baseline for the collection and analysis of exascale requirements contained herein started from the existing petascale requirements [1], but there are differences between the petascale and exascale requirements, opportunities, and challenges. These have been identified as a result of recent surveys, interactions, and interviews with key computational scientists (see Appendix A) and a series of town hall meetings organized by DOE [2]. The salient features of the resulting requirements for exascale systems are contained in this document.

By articulating these requirements and using them to manage and arbitrate decisions, the NCCS will better align LCF systems with the needs and goals of the science projects using these resources. LCF requirements for the NCCS apply to the entire end-to-end analysis process that scientists follow when using the NCCS facilities. This process comprises system hardware, system software, the integrated development environment, and the problem-solving environment that includes data analysis, management, and visualization (i.e., data analytics). We expect that effective requirements development, management, and planning will positively influence the design, procurement, deployment, and operation of an NCCS system by providing a measurable improvement in the quality, quantity, or fidelity of breakthrough science simulation applications. For requirements to be useful to the NCCS, they must be actionable and as quantitative as possible without being solutions themselves. In reality, requirements flow in both directions: Applications impose requirements on the LCF systems, and the LCF systems in turn impose requirements upon the applications.

To start, consider the science drivers, objectives, and impacts possible with an exascale computing resource (Table 1). Science prospects and benefits exist in material science, earth science, energy assurance, and fundamental science, to name a few (for more detail, see [3]). Because large-scale computing is at the brink of another revolution (computing capability growth is accelerating), and because all key science domains are now developing, using, and relying on computational science tools for exploration and discovery, never before has computational science been presented with such an exciting opportunity . We are indeed at a tipping point for computational science and its role in accelerating and enabling scientific discovery. Examples of the possibilities outlined in Table 1 include:

- Synthesizing nanoparticles for a variety of designed tasks (e.g., energy storage);
- Understanding and designing high-temperature superconductors;
- Sequestering carbon in a predictable and manageable way;
- Predicting decadal climate outcomes to aid in preparing for climate change;
- Characterizing and bounding the coupled earth system, including its socio-economic linkages and dependencies;
- Designing catalysts for a wide variety of industrial processes;
- Using reliable, whole-device simulation of the multinational ITER fusion reactor [4] for efficient design and operation; and
- Deciphering the chemical evolution of the galaxy and the place of supernovae.

These and many other science drivers are detailed in Table 1 and references 1–3. Based on these drivers, their possibility of success, and their impact, it is clear that an exascale computing system will have an enormous societal return on investment when objectives associated with these science drivers are achieved.

Table 1. Select science drivers for leadership computing at the exascale

Opportunity	Application area	Science driver	Science objective	Impact
Material science	Nanoscale science	Understand synthesis of alloy nanoparticles with potential impact for design of new catalysts	Define the thermodynamics of compositions of alloy nanoparticles	Magnetic data storage; economically viable ethanol production; energy storage via structural transitions in nanoparticles
		Physics of interacting fermions on a lattice	Explain the fundamental mechanism of high-temperature superconductivity within a minimal model of interacting lattice fermions	Macroscopic quantum effect at elevated temperatures; power transmission and electronics
Earth Science	Environment	Carbon sequestration in geologic formations	Model dissipation of supercritical CO_2 injected in the subsurface accounting for fingering phenomena in kilometer-scale basin simulations	Carbon management via active capture and storage
	Climate	Decadal climate prediction	Cloud-resolving (1–5 km) atmosphere; longer time integration (100–300 years, 1000-year spinups); larger ensembles (5–20)	Understand and prepare for committed climate change
		Characterize and bound the coupled earth system	Maintain tolerable time-integration rates while increasing model resolution and complexity; integrate models and observations; model at the process level biogeochemical cycles and coupled physical and biogeochemical system	Understand and predict stability and sustainability of rain forests, polar ice and ice sheets, agricultural ecosystems, precipitation, methane hydrates, and extreme weather; quantify mitigation strategies
		Dynamical linking of socio-economic and climate responses	Couple infrastructure, climate, demographic, informational, and energy economic models to predict adaptation as communities react to stresses on infrastructure systems and propose potential policies	Identify future energy infrastructure needs
Chemistry	Chemistry	Systematic, large-scale exploration of optimal materials for catalysis or nuclear material separation agents	Combine density functional theory with evolutionary search for complex materials or an accurate combinatorial approach to screen the best separation material out of $O(10^3)$ compounds	Virtual design of catalysts and separating agents
Energy assurance	Bioenergy	Biomass recalcitrance	Understand the complexity of plant cell wall structure and its relationship to recalcitrance via large-scale (e.g., microbial and plant cell wall structures and cellulosomes) simulations of 10-100M atoms over ms timeframes	The most important current barrier to the emergence of a cellulosic biofuels industry

Table 1 (continued)

Opportunity	Application area	Science driver	Science objective	Impact
Energy assurance (continued)	Combustion	Understanding "flameless" combustion of diverse fuels at high pressure in a turbulent environment relevant to advanced fuel-efficient, low-emission engine concepts	Direct simulation of nonconventional mixed-mode, turbulent combustion of biofuels under compression ignition aero-thermo-chemical regimes accounting for emission using statistical moments and models for particulate matter	New combustion systems designed to use alternative fuels with high efficiency while meeting stringent requirements on emissions
	Fusion	Reliable, whole-device modeling of ITER	Coupling of auxiliary heating, MHD dynamics, and plasma core and edge codes	ITER design and operation
	Nuclear energy	A virtual simulator for facilities within an operating closed fuel cycle	An integrated set of models and simulations of the complete set of physical processes and facilities within an operating fuel cycle	A decision-making tool to help predict the outcome of changes made to the system as it operates
Fundamental science	Astrophysics	Detailed simulations of core-collapse supernovae, including nucleosynthesis, gravitational waves, and neutrino signatures	Perform core-collapse simulations with Bolztmann neutrino transport and nuclear kinetics capable of isotopic evolution for a wide range of stars	Understand the chemical evolution of the galaxy and the place of supernovae
	Nuclear physics	Decipher the evolution in time of fission and fusion processes	Use time-dependent coupled-cluster theory to investigate the time evolution of "below the barrier" events to deduce fragment mass and energy distribution	Nuclear energy and fusion reactors
	Accelerator physics	Optimize and design future particle accelerators for better efficiency at lower costs and develop advanced accelerator concepts	Include electromagnetic, thermal and mechanical effects for the ILC RF unit to determine optimal linear accelerator design	Increased return on investment of large DOE accelerator facilities

ILC = International Linear Collider
MHD = Magnetohydrodynamics
RF = Radio frequency

2. MODEL AND ALGORITHM REQUIREMENTS

Application models drive the need for specific numerical algorithms and software implementations. Often, these requirements are predetermined by the features and specifications found in specific systems, with system attributes such as peak speed or node memory capacity constraining the functional attributes available to a model. As a model is implemented on a future system, attributes such as the following are all influenced by the choice of system:

- Model state variables (how many planned);

- Model characteristics (partial differential equations [PDEs] vs. ordinary differential equation [ODEs]; deterministic vs. stochastic; formulation of equations; etc.);
- The presence of multiple, simultaneous phenomena and the required degree of coupling;
- The domain of dependence (local with specific patterns, global, etc.);
- Data dependency (degree of parallelizability); and
- Resolution, complexity, and duration and/or ensemble size.

Once a physical model has been postulated, the application developer must apply one or more algorithms to it in order to generate numerical solutions. For the applications we highlight here, the physical models tend to be sets of coupled linear and nonlinear PDEs and ODEs. Applications lead to algorithms, meaning application model and algorithm requirements are closely tied. A useful characterization of algorithms prevalent today is the so-called Seven Dwarfs of Colella [5]. Table 2 demonstrates this approach for the algorithms expected to play a key role within select scientific applications at the exascale.

Table 2. Algorithms expected to play a key role within select scientific applications at the exascale, characterized according to a seven dwarfs classification

Opportunity	Application area	Structured grids	Unstructured grids	FFT	Dense linear algebra	Sparse linear algebra	Particles	Monte Carlo
Material science	Molecular physics			X	X		X	X
	Nanoscale science	X			X		X	X
Earth science	Climate	X	X	X		X	X	X
	Environment	X	X			X	X	X
Energy assurance	Combustion	X			X		X	
	Fusion	X	X	X	X	X	X	X
	Nuclear energy		X		X	X		
Fundamental science	Astrophysics	X	X		X	X	X	
	Nuclear physics				X			
	Accelerator physics		X			X		
	QCD	X						X
Engineering design	Aerodynamics	X	X		X	X		

FFT = Fast Fourier Transform
QCD = Quantum chromodynamics

Several trends are noteworthy in the Seven Dwarfs categorization of codes in Table 1:

- The seven algorithm types are scattered broadly among science domains, with no one particular algorithm being ubiquitous and no one algorithm going unused.
- Structured grids and dense linear algebra continue to dominate, but other algorithm categories will become more common.
- Compared to the Seven Dwarfs for current applications [1], the table projects a significant increase in Monte Carlo and increases in unstructured grids, sparse linear algebra, and particle methods, as well as a relative decrease in FFTs. These projections reflect the expectation of much-greater parallelism

in architectures and the resulting need for very high scalability. Load balancing, scalable sparse solver, and random number generator algorithms will be more important.

- Some important algorithms are not captured explicitly in the Seven Dwarfs. Categories expected by application scientists to be of growing importance in 2010–2020 include adaptive mesh refinement, implicit nonlinear systems, data assimilation, agent-based methods, parameter continuation, and optimization.

As we approach the petascale and plan for the exascale, we will see several challenges arising from the sheer number of coupled equations and systems, as well as the apparent ability to significantly increase the spatial, temporal, and state variable resolution in these simulations. Some of these hurdles will resemble today’s challenges. Others will be fundamentally new, as one or more of the “seven dwarfs” kernels become relatively more important or prevalent. To deal with possible new challenges and opportunities provided by future architectures it is imperative that collaborations and close interactions occur between computer scientists, computational scientists, numerical mathematicians, and domain scientists to investigate new algorithms that might be more appropriate for the highly parallel hybrid HPC architectures anticipated for exascale systems, rather than the currently used algorithms that are still largely based on experience with serial machines (albeit with adaptations to parallel architectures.)

3. SOFTWARE REQUIREMENTS

The opportunities and challenges faced by computational scientists in an exascale computing environment are enormous. We already have a glimpse of the future of computing, with the proliferation of heterogeneous, functionality-specialized processors with hierarchical memory. The hardware also appears to be headed down that road, and the supercomputers of the next decade will no doubt incorporate this heterogeneous model into their architectures. Successful use of exascale resources promises great scientific advances, but enabling users to achieve this level of effectiveness will be difficult. In order to evolve with the systems, applications will have to follow multiple paths, with some being rewritten, others being augmented, and still others being newly developed.

The days of the “hero-developer” are long over. Not only does this model not scale to the enormous size and complexity of even today's production codes, no single person can be expected to adequately understand and address the depth and breadth of the issues associated with creating high-quality applications for these platforms. These goals can be accomplished only by computer scientists (including language specialists, compiler writers, runtime system developers, performance experts, etc.), algorithm developers, code developers, and end-user scientists working in a tightly integrated manner.

The role of the computer scientist will be to develop a means of interface between the heterogeneous computer and the developer and end-user scientists. Computer scientists must develop a set of software tools that allow a natural mapping of algorithms onto a diverse set of architecture capabilities. For example, these tools must include a useful and helpful debugger, because as machines grow in size, print-statement debugging loses its effectiveness. The manner in which developers can express ideas must complement the requirements of the compiler and runtime system. Currently, the prevailing approach is based on low-level constructs supported by current technologies. While this approach is expected to remain portable across emerging and future architectures (in the sense that code will run, although suboptimally), we already see constraints on performance. The fundamental problem is that low-level constructs overconstrain the compiler and runtime system, preventing use of architecture-specific capabilities.

It is important to recognize that this does not rule out these mechanisms; by raising the level of abstraction, they can become options beneath the layer of abstraction. Examples of this approach are well known in popular libraries: ScaLAPACK [6] moves data about the distributed memory environment through the BLACS [7], which may be configured to use an interprocess communication protocol. Zoltan

[8], a package for partitioning data among the processors, is based on a similar approach, allowing for experimentation of algorithms.

Abstraction enables incorporation of intent into the organization of the data structures. Memory could also be organized in a manner that exploits architecture capabilities. This has been done manually [9], but it is too labor-intensive for the code developer. Asynchronous capabilities for managing data flow should be developed, allowing for concurrent computation with disk operations (check-pointing, visualization, etc.). Previous attempts have been hindered by a lack of hardware support. Multiple processing elements in emerging architectures may be exploited to realize this capability.

This approach also enables the inclusion of fault tolerance with regard to both algorithmic and hardware issues [10]. For example, a process asks for data using a "gather" command, and that data may be moved using any available protocol.

These tools must also enable algorithmic experimentation at scale; an algorithm that works well at even the petascale may work poorly at the exascale. Power consumption is not constant, so power-aware algorithms are being explored. New approaches to existing and emerging questions must be developed. Current models require heroic software development projects, impeding this sort of experimentation.

Leadership computing application requirements, opportunities, and challenges are consistent with the observations laid out earlier this year in a series of town hall meetings [2]:

- Application development and maintenance tools and practices need to fundamentally change;
- Productivity improvement is an important metric and guide for tool and software choices;
- Fault tolerance and verification and validation (V&V) software components must be used to improve reliability and robustness of application software;
- Knowledge discovery techniques and tools should be explored to help with bug detection, simulation steering, and data feature extraction and correlation; and
- A holistic view of application data (from input to archival) is needed to most effectively deliver tools for the end-to-end workflow performed by scientists.

A single set of software tools will in general not be appropriate for all applications. Instead, functional views should provide mechanisms for different aspects of development and different types of algorithms. Coupling of models must be seamlessly enabled by the programming languages and models. It is imperative that the new software stacks have the capability to reuse legacy code where appropriate. The number of man-years invested by federal funding agencies in current software capabilities cannot be ignored. In addition, practical considerations (e.g., programmatic deliverables) will prevent development teams from investing the time that would be necessary to reinvent these applications with completely new technologies.

4. SYSTEM REQUIREMENTS

System requirements are requirements that specify system hardware and software needs. Application needs cover hardware footprints, from memory usage and memory patterns to communication usage and patterns to input and output (I/O) usage and all derived metrics in between (like memory and interconnect bandwidths and latencies). Managing the various hardware resources is the job of the system software, and thus the application hardware requirements imply requirements at various levels of system software. Of particular importance are requirements for the operating system and job-management system.

4.1 SYSTEM HARDWARE

A given LCF system has many attributes that uniquely characterize it relative to other systems, but 12 attributes in particular are useful to consider from an application's perspective and have been used for current LCF systems. These attributes are listed in Table 3. For each of these 12 attributes, certain behaviors and properties of a given application may highlight the need for one particular attribute relative to another. Table 3 summarizes application behaviors and properties that serve as drivers for system attributes.

Table 3. Science application behavioral and algorithmic drivers for leadership system attributes

Leadership computing system attribute	Application algorithms driving a need for this attribute	Application behaviors driving a need for this attribute
Node peak flops	Dense linear algebra, FFT, sparse linear algebra, Monte Carlo	Scalable and required spatial resolution low; would benefit from a doubling of clock speed; only a problem domain that has strong scaling, completely unscalable algorithms; embarrassingly parallel algorithms.
Mean time to interrupt	Particles, Monte Carlo	Naïve restart capability; large restart files; large restart R/W time.
WAN bandwidth	Long time evolution, multiphysics, multiscale	Community data/repositories; remote visualization and analysis; data analysis.
Node memory capacity	Dense linear algebra, sparse linear algebra, unstructured grids, particles	High DOFs per node, multi-component/multi-physics, volume visualization, data replication parallelism, restarted Krylov subspace with large bases, subgrid models.
Local storage capacity	Particles, out-of-core algorithms	High-frequency/large dumps, out-of-core state, debugging at scale.
Archival storage capacity	Long-time evolution, multiphysics, multiscale	Large data (relative to local storage) that must be preserved for future analysis, for comparison, for community data (e.g., EOS tables, wind surface, and ozone data); expensive to recreate; nowhere else to store.
Memory latency	Sparse linear algebra, unstructured grids	Random data-access patterns for small data.
Interconnect latency	Structured grids, particles, FFT, sparse linear algebra (global), Monte Carlo	Global reduction of scalars; explicit algorithms using nearest-neighbor or systolic communication; interactive visualization; iterative solvers; pipelined algorithms.
Disk latency	Out-of-core algorithms	Naïve out-of-core memory usage; many small I/O files; small record direct-access files.
Interconnect bandwidth	FFT and other spectral methods, coupled models	Large messages, global reductions of large data; implicit algorithms with large DOFs per grid point.
Memory bandwidth	Sparse linear algebra, unstructured grids	Large multidimensional data structures and indirect addressing; data copying; library calls, requiring data copies if algorithms require data retransformations; sparse matrix operations; efficient utilization of many-core processors.
Disk bandwidth	Out-of-core algorithms	Reads/writes large amounts of data at a relatively low frequency; read/writes large amounts of large intermediate temporary data; well-structured out-of-core memory usage.

DOF = Degree of freedom
EOS = Equation of state
R/W = Read/write
WAN = Wide area network

Qualitative prioritization of these system attributes for each domain science is shown in Table 4. Attributes that are likely to increase in priority in 2010–2020 relative to today are marked with “+”, and the ones that are most likely to decrease in relative priority are marked with “–”. Each application area is constrained to have four areas “+” and four “–”. The “Summary” column gives the sum of “+” and “–” across applications.

Table 4. Relative prioritization of twelve leadership system attributes for relevant science domains

System attribute	Climate	Nuclear theory	Fusion	Chemistry	Combustion	Accelerator physics	Biology	Materials science	Summary
Node peak flops	–	+		+	+	–	–	+	+1
MTTI		+				+		+	+3
WAN network bandwidth	–	–	+	+		+	–	–	–1
Node memory capacity	–	+			–	+			0
Local storage capacity		+	–		–				–1
Archival storage capacity			–			–		–	–3
Memory latency	+	–		–	+		+	+	+2
Interconnect latency	+	–		–	–	+	+	+	+1
Disk latency	–		–		–	–	–	–	–6
Interconnect bandwidth	+	+	+	+	+		+		+6
Memory bandwidth	+		+		+		+	+	+5
Disk bandwidth			–	+	–	–	–		–3

MTTI = Mean time to interrupt

The summary column shows that the attributes expected to increase most in priority are interconnect bandwidth, memory bandwidth, MTTI, memory latency, and interconnect latency, in that order. The attributes expected to decrease most in relative priority are disk latency, archival storage capacity, disk bandwidth, WAN network bandwidth, and local storage capacity. It is unclear whether these choices by application developers reflect the expected needs of applications or the expected reality of available hardware. One interpretation is that the increasing priorities reflect the desire to increase computational efficiency to take advantage of increasing peak flops, while the decreasing priorities reflect the expectation that computational efficiency will not increase. The relative future priority of local and archival storage capacity and bandwidth in Table 4, qualitatively estimated at lower relative priority (and/or less available), is not perfectly aligned with the detailed, demanding I/O and storage requirements spelled out in section 5, which is an area of obvious and increasing priority on exascale systems. Relative to Table 4, an alternative analysis for Table 3 that supports section 5 is to at least maintain the current I/O capacity and bandwidth (in a relative sense) in future exascale systems. These two leadership system attributes (storage capacity and bandwidth) need further scrutiny, planning, and quantification.

An example of moving from qualitative to more quantitative runtime requirements is the analysis of what currently constitutes a single simulation for selected application codes. Such analysis helps to validate the importance of system attributes for these codes. Table 5 presents current typical development characteristics and runtime requirements of a single simulation for selected application codes on the NCCS LCF systems [1].

Table 5. Typical development characteristics and runtime requirements of a single simulation for selected application codes on the NCCS LCF systems circa June 2006

Science domain	Code	Code attributes	Job size (nodes, time)	Storage capacity needs (local, archive)	Node memory capacity needs	Number of queue dwell times for full simulation
Accelerator design	Omega3D	9 years old, 173-K C++ LOC, 12 developers	128–256, 24 hours	1 TB, 12 TB	8 GB	3–4
Astrophysics	CHIMERA	Components 10–15 years old, 5 developers, F90	128–256, 24 hours	300 GB, 2 TB	≥2 GB	10–15
	Vulcan2D	9 developers, F90	48, 24 hours	300 GB, 5 TB	2 GB	30
Climate	CCSM	Components 20 years old, 690 K Fortran LOC, over 40 developers	250, 24 hours	5 TB, 10 TB	2 GB	10–30
Combustion	S3D	16 years old, 100 K Fortran LOC, 5 developers	4000, 24 hours	10–20 TB, 300 TB	1 GB	7–10
Fusion	GTC	7 years old, ~30 developers	4800, 24 hours	10 TB, 10 TB	2 GB	4–5
Nuclear physics	CCSD	3 years old, 10 developers, F90	200–1000, 4–8 hours	300 GB, 1 TB	2 GB	1

CCSD = Coupled-cluster singles and doubles
CCSM = Community Climate System Model
GB = Gigabytes
TB = Terabytes
GTC = Gyrokinetic Toroidal Code
LOC = Lines of code

The recent "Modeling and Simulation at the Exascale for Energy and the Environment" report [2] provides a summary of runtime requirements for 2015; a balanced system will require on the order of 150 GB/s of injection bandwidth, 1.5 petabytes (PB)/s global bandwidth, and 500 nanoseconds latency. This document also reports that applications will require 0.5–2 bytes/flop of memory bandwidth and at least 2–4 GB of memory per CPU [central processing unit] core. The memory-per-core requirement is close to the current application requirements reported in Table 5. The implication is that per-core requirements will be relatively static while the aggregate requirements will grow with the system. This projection is consistent with a relatively small increase in performance per core, with a dramatic increase in the number of cores.

The continuing growth of potential computation rate will continue to strain I/O requirements. One rule of thumb for application I/O is that it consumes <5% of total run time, including time for checkpoints that may dump 10–50% of the physical memory. At the exascale, such a checkpoint is on the order of petabytes and requires terabytes per second of bandwidth. The resulting scratch file system could require

250,000 disks [2] to provide such bandwidth. It is unclear if I/O bandwidth and volume can feasibly keep up with increases in computation rate. The priority expectations from application developers, in Table 4 above, may indicate that I/O does not need to keep up, or just that it is not expected to. Options exist for easing memory and I/O requirements, including recomputing instead of storing, in-memory checkpointing, performing data analysis during computation, and overlapping I/O with computation.

I/O capability may lose further ground because of disruptive technologies that could accelerate the availability of exascale computation. Potentially disruptive hardware technologies include three-dimensional chips and memory, optical processor connections, optical networks, customized processors, and more optimal packaging on chip dies, node boards, and within cabinets.

4.2 SYSTEM SOFTWARE

The applications interact with the underlying hardware through the glue provided by the system software. As the time and memory consumed by the system software are unavailable to the application, the continuing trend to higher parallelism at multiple levels (increase in the number of processing nodes, multiple levels of memory and cache, and the number of compute cores per node) requires that the system software be as unobtrusive as possible. The execution of daemons and background tasks can introduce variability between different processes, which can severely impact parallel performance. Even a small amount of "jitter" can cause a dramatic increase in operations that require global communication [11, 12].

A second issue with system software that will increase in importance for highly parallel systems and applications is general stability, reliability, and fault tolerance during application execution. Applications may be more likely to experience loss of resources during a run. The system software can mitigate these losses with a range of responses from redistributing that task (most likely with aid from the application) to simply notifying the application of the failure. As stated in [2]: "With the potential that exascale systems will be having constant failures somewhere across the system, application software isn't going to be able to rely on checkpoint/restart to cope with faults. … For exascale systems, new fault tolerance paradigms will need to be developed and integrated into both existing and new applications."

Beyond resource availability, efficient scheduling of those resources is a major obstacle faced by leadership-computing centers. Existing schedulers may have been designed for contexts other than leadership computing and may not support the priorities, work flow, or scale of leadership computing. Design mismatches can appear as a lack of robustness of the schedulers (e.g. slow response time, intrusive daemons) and in problematic configuration issues. The growing need to manage expanded work flows and schedule multiple phases of those workflows complicates the situation. Just as I/O must not be an afterthought in hardware design, job management must not be an afterthought in the design of system software.

5. DATA ANALYTICS REQUIREMENTS

Data analytics are required to extract scientific results from simulations performed on leadership-class systems. This can be achieved by combining strong software engineering philosophies with systems capable of end-to-end computing, where "end-to-end" refers to the entire process, from computing to I/O (both disk and tape) to analytics systems to scientific results. The users must be able to ask the "what-if" questions that drive their scientific inquiry. Getting answers to these questions from massive simulation data is indeed akin to finding a needle in the haystack.

An excellent way to understand long-term data analytics requirements is to first examine the leading scientific applications currently running on leadership systems, understand their short-term requirements, and extend into future.

Most large-scale simulation application development and usage follow a common process. They start with a thought and use mathematics to translate this thought into equations. The equations are then transcribed into code. The user then runs the code and writes out information to disk. After going back and forth verifying the code, the user can begin to validate the code. Once the validation process is complete (or at least mature), the user can finally run predictive simulations. As the science evolves, better models are produced, allowing the scientific community to run good approximations to these first-principles calculations, which in turn aids in the process of augmenting the codes with additional equations or parameters, which begins the process again.

Usually the focus is on flops and network bandwidth, but in science-driven analytics for the next 20 years we must seriously consider methods to shield scientists from this level of detail. We must push for a closer integration of database technologies with parallel and out-of-core techniques. We advocate for a new computing environment in which scientists can ask, "What if I increase the pressure by a factor of ten," and have the analytics run the appropriate codes to examine the effects of such a change.

We also expect within the next two decades for the paradigm to change, from one in which we must load all data onto a large machine to perform analytics to one in which we stream the data from disk/tape and apply the data analysis and visualization routines. This change is necessary because datasets are growing exponentially large. For example, simulations by fusion scientists (using the GTC code) will start producing data at a rate of about 60 GB every minute. Although this rate may not appear overwhelming at first glance, it amounts to more than 600 TB of data produced in less than a week. If the researchers knew exactly what to look at within the data, this requirement could easily be reduced by perhaps a factor of 1,000. The problem is that if scientists knew exactly what they were looking for, they could build a much smaller model of the code and then run this on a much smaller number of processors. Science is about discovery, and we need to enable this discovery in exascale computers.

In many fields, the data management challenges will heighten as we begin coupling codes. These multiscale, multiphysics codes will begin to dominate the high-end computing platforms as we move toward exascale computing. The analytics and data management problems are compounded by multiple authorship (e.g., the CCSM climate model) and the explosion in dataset size. Fortunately, the actual number of features in the data does not increase exponentially, but rather increases linearly with the quantity of physical processes in the simulation.

We must be proactive in designing an analytics system directly coupled to our computing platform. Scientists need to be able to extract subsets of data before it initially hits disk, and perhaps also after it is on tape. Workflow automation techniques can apply on smaller systems, behind the scenes, continuously streaming data from tape to disk and from disk to analytic systems.

One final piece of the puzzle is that we need to enable the science community to examine its data for validation and predictive capabilities in these codes. This means that we need scientists to be able to work with prototyping systems (e.g., IDL, Matlab, R) to prototype their analysis. Using the streaming techniques mentioned above, scientists can then deploy their software on these advanced computing resources and do their science. Higher-level users will then want to look at the suite of software created by the science community to ask "what-if" questions, which can then make use of the software infrastructure on the complex computing resources. Allowing users to interact with the data—without requiring an understanding of arcane technical concepts such as parallel file systems or parallel algorithms—will enable us to reach new levels of science. This can be possible if we think about how to make these environments more conducive for doing science.

In order to make this vision into a reality, we must look at the entire pipeline of data analysis, from application data writing to analysis activities. This process may itself be analyzed by collecting requirements from each current application, determining the following:

- The desired analysis tools and algorithms (visualization, data mining, etc.),
- File system storage needs per simulation,
- The maximum desired write time as a percentage of simulation time, and
- Archival storage needs per simulation.

Given the above, several important application-specific data analysis requirements directly follow—for example, output bandwidth (gigabytes per second) from the leadership computing system to local storage. The most demanding data management requirements often come from applications that incorporate multiphysics and multiscale models. This kind of coupling leads to high dimensionality in evolved quantities (e.g., radiation fields, chemical and nuclear species, and particle phase spaces). Many applications also tend to involve long-time evolutions. Therefore, large multidimensional datasets are output at many regular intervals to allow for analysis of time-dependent correlations and the overall evolution of the modeled systems. The particular science objectives for these types of applications are often directly related to a set of resolution requirements—in time, space, ensemble, and multivariate—which in turn determines the overall quantity and size of datasets output. This scaling also holds for the I/O requirements for checkpoint (restarts). The infrastructure requirements for the LCF stemming from these application requirements ultimately set the scale of the scientific simulations that can be performed on the system. It also constrains the set of knowledge discovery and data analysis activities that can be performed on the resultant simulation data.

The I/O requirements for LCF systems can be broken down into two distinct categories, namely, those required to

- Output results in the form of restart dumps and other analysis files, and
- Postprocess data files for analysis and visualization.

The output portion of I/O requirements can be estimated for the LCF by examining the needs of the largest data-producing codes on the current systems, currently CHIMERA, GTC, S3D, T3P/Omega3P, and POP [1]. Output of the codes can be in two categories: restarts and analysis. On the 250 TF systems some of these codes, GTC for example, will be producing 1 GB/minute (analysis data), and this amount is projected to double. The need to verify these simulations against experiments had engendered this data explosion. Researchers using codes such as S3D want the ability to write out 3 PB, which is 100 times the amount of data they are currently writing, but they are currently limited by the I/O bandwidth, their analysis pipeline, and the lack of storage. Scientists would also like an environment in which they can easily switch from synchronous I/O techniques to asynchronous techniques and in situ visualization techniques. Ideally, they want to maintain a certain quality of service to maintain <5–10% overhead for I/O for the life of their simulation. These demands are usually made by scientist who desire lots of metadata-rich output, but because of fluctuations on the I/O system due to other users, I/O can sometimes eat up 25% of their compute time. This limits the amount of science they can run on the machine because of the limited availability of leadership computing resources. As a result, they are left with less time for additional simulations, or for running the current simulation further in time.

For scientists to process this large amount of data on a machine of reasonable cost, we must focus on out-of-core routines coupled with data-streaming techniques. We need fast access to tape such that scientists can ingest the raw data from a day of simulation runs in a day of analytics. Understanding the end-to-end processes that scientists currently undertake can inform our hardware requirements for the future.

In order to give a more quantitative example of data analytics requirements, consider an exaflop machine with 100 PB of main memory, with typical application restart outputs approaching 10% of the available memory every hour, or roughly 10 PB/hour. The analysis output from the simulations will approach similar levels. Together, the checkpoint and analysis output will amount to 20 PB/hour. Typical execution time per week from these very large data-producing codes will most likely never exceed one full week per month on the entire machine. This allows the requirements to be estimated as follows:

- **Disk Bandwidth.** To handle the burst from 20 PB of output per hour with <10% overhead, we need to write out 20 PB/360 s, or about 50 TB/s. Asynchronous I/O has the potential to reduce the sustained bandwidth required to 20 PB/3600 s, or about 5 TB/s while the application still experiences less than 10% overhead. We must have quality of service to ensure this during the lifetime of the run.
- **Disk Capacity.** The scratch has to be able to hold the data produced by a simulation for at least

3 weeks since the time it was performed. The volume of data that will be produced by a typical simulation will be about 20 hours/day for 5 days at 20 PB/hour, or 2 exabytes (EB). We will need approximately 6 exabytes of scratch space to hold simulation data for 3 weeks.

- **Tape Bandwidth.** As mentioned earlier, the data generation rate is 5 TB/s. However, not all the checkpoint data will need to be archived. Furthermore, not all applications will be performing I/O at the peak level predicted earlier. Therefore, it is estimated that a sustained tape-write bandwidth of 1 TB/s will be required to match the data generated by the simulations. A read bandwidth that is higher than write bandwidth is desirable, since it expedites the analysis workflow when it is time to revisit the archived data. For that reason, a read bandwidth of 2 TB/s is recommended.
- **Analytics Machine Memory.** Given that one needs to read in multiple time slices and multiple variables, one would assume that the analytics machine will have to contain about 1/100 of the total memory of the entire dataset. We will assume that with out-of-core and streaming capabilities introduced in the software stack we will have to handle a few correlation lengths and a correlation time. We must approximate this amount of memory as approximately 1/100 of the total amount of data/timestep and about 1/100 the total amount of data (for all timesteps). This gives us a reduction of four orders of magnitude. This gives us an estimate of 200 TB, or about 1/500 of the memory on a 100 PB exascale computer. Another way to examine this is to figure that we need approximately the same amount of memory for one full timestep of analysis data. Since some codes will actually use the majority of their restart files as analysis data (e.g., the S3D code), this says that we will need about 10 PB of data. Luckily, the checkpoint data produced by such codes is only about 1/40 of total memory on the production platform. This reduces our memory constraint to about 2.5 PB of memory. Thus, the memory requirement can vary from 200 TB to 2.5 PB of memory.
- **Analysis Machine, Type of Memory.** For fast prototyping of analysis routines, users have frequently requested shared memory machines. This allows them to quickly prototype machines without worrying about living in the memory constraints of massively parallel processing. This can be accomplished in two ways: fat nodes with “large-enough” memory, and fully shared memory machines. If shared-memory machines with 200 TB of memory are too costly, then minimally it would be reasonable to assume we would like to split this amount of memory between at most 8–10 machines, so that one could look at a “large-enough” region of degrees of freedom (spatial degrees, degrees in variables, etc.). This puts the minimal amount of shared memory necessary at about 20 TB of memory.
- **Latency, bandwidth.** Analysis routines are generally more sensitive to latency, since these routines usually have not been optimized as much as the large simulations. The bandwidth needs to be “fast enough.” It is reasonable to assume that the injection bandwidth needs to be equal to the exaflop machine, and the latency needs to be at least 2 times better.

Clearly a paradigm shift must occur for researchers to find the needle in the haystack. Analytics is the place where LCFs truly need blue-collar computing. We need to empower scientists to ask “what-if” questions and have the software and hardware infrastructure capable of answering these questions in a timely fashion. Strong analysis, visualization, and workflow are important. Strong data management will not just become important—it will become an absolute must as we move into the age of exascale computing. We can already see these trends in the computing world by looking at companies like Google. Google desktop has revolutionized desktop computing by allowing us to find information that might have otherwise gone undetected. These types of technologies are moving into leadership class computing and must be made to work on the largest analysis machines.

6. ACCELERATING DEVELOPMENT AND READINESS

The anticipated requirements described in the previous sections motivate research and development (R&D) in a wide range of areas. The following section suggests cross-cutting R&D themes designed to meet the most prominent and critical unresolved challenges resulting from these requirements.

6.1 AUTOMATED DIAGNOSTICS

Improved diagnostics are a growing need across the spectrum of high-performance computing (HPC) activities, including but not limited to scientific inquiry, application development, system administration, network administration, hardware maintenance, cybersecurity, center management, and future hardware and software design. The required growth in size and complexity of software and systems may only be feasible through aggressive automation of diagnostic instrumentation, collection, and analysis. Drivers for automated diagnostics include performance analysis, application verification, software debugging, hardware-fault detection and correction, failure prediction and avoidance, system tuning, and requirements analysis.

6.2 HARDWARE LATENCY

Whereas aggregate computation rate, parallelism, and bandwidth should improve significantly over the coming years, hardware latencies are unlikely to see similar improvement. In addition to software strategies to mitigate high latencies, hardware improvements could reduce latencies for targeted operations or improve the prospects for latencies to be hidden through overlap with other operations. Possibilities include fast synchronization mechanisms on chip, in memory, or over networks, along with smart networks that can accelerate or offload higher-level latency-sensitive operations, like global floating-point reductions.

6.3 HIERARCHICAL ALGORITHMS

One reaction to latency stagnation has been to increase the depth of the memory hierarchy, a trend that is likely to continue at wide-ranging levels of the hierarchy. Heterogeneous computing is beginning to create a process hierarchy, in addition to deepening the memory hierarchy further still. As systems continue to grow in complexity, the need to tolerate failures could bring redundancy from the archive and filesystem level (RAID) to the memory level. Applications will require algorithms that are aware of the system hierarchy and can adjust to it. In addition to the now-common strategies of cache blocking, hybrid data parallelism, and file-based checkpointing, algorithms may need to include dynamic decisions between recomputing and storing, fine-scale task-data hybrid parallelism, and in-memory checkpointing.

6.4 PARALLEL PROGRAMMING MODELS

The requirements in this document motivate specific improvements for parallel programming models and libraries: arbitrary task/data parallel hierarchy and minimized synchronization. As described above, the memory hierarchy of HPC systems continues to deepen, but current programming models still target one level of the memory hierarchy at a time. For example, a hybrid-parallel application uses the source language for instruction-level parallelism, OpenMP for multi-processor intra-node parallelism, and message-passing interface (MPI) for inter-node parallelism. New levels of the memory hierarchy are mapped to one of the existing levels of parallelism or ignored. The levels of parallelism within the application are mapped to a specific, distinct level of the programming model and are difficult to modify. An improved programming model would allow the application developer to identify an arbitrary number of levels of parallelism within an application and map them onto hardware hierarchies at runtime, perhaps dynamically. Models continue to be coupled into larger models, driving the need for arbitrary hierarchies of task and data parallelism. Finally, latency stagnation drives the need for minimized synchronization, or alternately maximized asynchrony.

6.5 SOLVER TECHNOLOGY AND INNOVATIVE SOLUTION TECHNIQUES

It is safe to make two broad assumptions:

- Scientists will continue to desire to solve more ambitious problems, with simulations run at higher resolutions and on longer time scales. Models will continue to incorporate an increasing number of coupled phenomena.
- The machines will look quite different than they do today; for example, the number of processor cores will likely be in the millions and the number of cores per socket could be in the hundreds. Given current projections about future architectures, contention for memory bandwidth among the CPU cores within a socket will become a major performance hurdle.

In terms of what these might mean in practical terms for the solver community, many scientists will take advantage of massive increases in computing power by adding more coupled processes to their models. As many of these processes are strongly and nonlinearly coupled, operator-splitting schemes will become problematic, and in many cases fully implicit formulations will become necessary. For many such problems, one of the most promising approaches is Newton-Krylov-type methods [13] that utilize physics-based preconditioning.

Physics-based preconditioners algebraically decouple certain phenomena that we know—based on our knowledge of the physics—to be somewhat loosely coupled. One of the attractions of these preconditioners is that they can enable us to use efficient operator-split solvers from existing highly efficient codes without introducing errors due to decoupling into our solutions (because we decouple only in the preconditioner). This code reuse aspect is important because many scientific teams must minimize the time spent reengineering their existing code infrastructure for the exascale. In addition to this benefit, physics-based preconditioners can enable use of Jacobian-free methods that save memory and computational cost by avoiding explicit calculation of the Jacobian, use of scalable multilevel solvers on subsets of the problem, and many other techniques. There are many settings in which physics-based preconditioners have not been thoroughly explored; hence significant research effort is warranted.

Since it is likely that many of the simulations at the exascale will utilize implicit schemes, and because solver costs tend to increase with increasing simulation fidelity (which scientists will surely increase as we move to the exascale), it seems a given that solvers for the linear systems that arise will assume even greater importance. Based on projections about the type of architecture that an exascale machine might have, we see two major concerns that are significantly more worrisome than at the petascale:

- First, because global communication operations across millions of processors will be prohibitively expensive, solvers will have to eliminate global communication where feasible and mitigate its effects where it cannot be avoided. Research on more effective local preconditioners will become a very high priority.
- Second, if increases in memory bandwidth continue to lag the number of cores being added to each CPU socket, further research into ways to effectively trade flops for memory loads/stores is warranted. An example of such a technique is storing sparse matrices as collections of dense blocks (elements of which may be zero) rather than of individual elements. This has been done to a limited extent, but further research on, for example, means to automatically select the optimal block size for performance with a given problem/architecture combination is desirable. Another technique of more theoretical interest is the use of block Krylov methods. Traditional Krylov methods solve a linear system $Ax = b$ by building a space one basis vector at a time from which increasingly better approximate solutions are extracted. A block Krylov method solves a block linear system $AX = B$, expanding the space by k vectors (where k is the block size) at each iteration. Block methods

typically require fewer iterations than traditional Krylov methods but tend to require more total flops; they require fewer memory accesses, however, because they allow multiple matrix-vector products to be computed for each access of the matrix A. Block Krylov methods have enjoyed some success, but their widespread adoption has been limited by some theoretical issues. First, because solutions for the different right-hand sides generally converge at different rates, there is a need for deflation to remove the converged right-hand side from the block system. However, our understanding of the numerical difficulties involved is not complete. Second, although block Krylov methods are a natural choice when presented with a linear system with several right-hand sides, in many simulation codes we only have one right-hand side. A block Krylov method can still be helpful in such cases if appropriate additional right-hands are chosen, but methods to choose these vectors are still a matter of open research.

In short, the need for scalable linear and nonlinear solvers and eigensolvers will continue as the size and complexity of simulations grow. On future architectures, algorithms must be considered that might be significantly flops-inefficient in return for efficiency in memory loads and stores and global communication operations. The requirements include multilevel methods, preconditioners, adaptive mesh refinement, irregular meshes, Newton-Krylov methods, and complex-mesh generation. The various solver technologies need development and tuning within the context of the memory and process hierarchies and latency stagnation described above. Novel techniques should also be investigated. An example is the easily parallelized Monte Carlo method (popular for radiation transport as well as in various ab initio approaches), which could prove useful in a many-core environment as a linear or nonlinear solver or even as a deterministic solver of partial differential equations (e.g., Navier-Stokes equations).

6.6 ACCELERATED TIME INTEGRATION

Many applications require dramatic increases in the length of simulated time, but the exponential increase in single-process performance over the last few decades has stalled, and increasing parallelism does little for the serialized time dimension. Increasing resolution can exacerbate the problem by requiring shorter time steps for numerical stability. Complimentary strategies for acceleration of time integration include fully implicit methods (aided by solver technology), pipelined-in-time and parallel-in-time algorithms, and compact shape-preserving bases. The latter areas are relatively undeveloped and offer unknown potential. The urgency of progress grows as single-process performance stagnates and even declines.

6.7 MODEL COUPLING

Models continue to be combined and coupled into more complete and complex models. These coupled models require effective methods to implement, verify, and validate the couplings, which can occur across wide spatial and temporal scales. The coupling requirements drive the need for robust methods for downscaling, upscaling, and coupled nonlinear solving. Data assimilation is of growing importance, with the rapidly expanding volume of high-quality data from satellites, sensors, and new experiments. Evaluation of the accuracy and importance of couplings drives the need for methods for validation, uncertainty analysis, and sensitivity analysis of these complex models. The scientific payoff from future simulations depends on such evaluation.

6.8 MAINTAINING CURRENT LIBRARIES

Most current HPC applications rely upon libraries, particularly MPI and often BLAS, LAPACK, FFTW, ScaLAPACK, PETSc, or Trillinos. Such reliance is likely to continue and grow, so these libraries must continue to perform as HPC systems grow in parallelism and complexity. Promising new architectures will need these libraries, particular the baseline of MPI and the BLAS, to be viable targets. The libraries must be tuned and updated to achieve the performance potential of such new architectures.

7. SUMMARY

An important (and in retrospect, obvious) lesson learned in this process is simple: Application requirements must be elicited for periods extending at least two leadership system deployments in the future to allow for more creative, unconstrained thinking and planning. Current leadership computing facilities are focused tactically on the next system upgrade (usually one to two years away) as well as the next system delivery (usually two to three year away). The time period beyond the next system delivery is therefore three years or greater in the future. By exploring system and software requirements in this timeframe, we are able to pose questions without being constrained by the impending arrival of specific systems.

It is this more speculative thinking and planning that leads to new and innovative strategic plans and solutions beyond the horizon. To give a specific example, this process drives researchers to think about how to formulate and implement algorithms and software for totally new math libraries rather than how to tweak the performance of existing math libraries.

Leadership class facilities must engage in a regular and evolving applications requirements process that is rigorous and quantitative. This process is difficult and time-consuming, yet necessary. High-consequence decisions about current and future systems informed by this process will help to deliver systems best suited for accelerating scientific discovery and understanding.

The establishment of a formal, rigorous, and useful requirements management process is very challenging when applied to breakthrough science applications for leadership computing. At this level, the research is by its very nature exploratory and high risk. The requirements process must always evolve, continuing to improve as guided by lessons learned, just as this document must be a living document, ever changing to keep up with the applications themselves. Computational science requirements for leadership computing flow both ways—LCF systems set requirements for the science applications just as the science applications must set requirements for the LCF systems.

ACKNOWLEDGMENTS

The staff in the NCCS, particularly members of the Scientific Computing Group, would like to thank the computational scientists involved in the 2007 INCITE projects that received allocations on the NCCS LCF. Without their applications, science achievements, time, and support of the NCCS LCF, this facility would not have a reason to exist. This work has been supported by DOE's Office of Advanced Scientific Computing Research.

REFERENCES

1. Computational Science Requirements for Leadership Computing (Oak Ridge National Laboratory National Center for Computational Sciences, July 2007), available at http://nccs.gov/media-center/nccs-reports/.
2. Modeling and Simulation at the Exascale for Energy and the Environment (Office of Science, U.S. Department of Energy, October 2007), available at http://www.mcs.anl.gov/~insley/E3/E3-draft-2007-08-09.pdf.

3. Science Prospects and Benefits with Exascale Computing (Oak Ridge National Laboratory National Center for Computational Sciences, December 2007). This document is in final editing and will soon be available at http://nccs.gov/media-center/nccs-reports/.
4. See http://www.iter.org.
5. P. Colella, "Defining Software Requirements for Scientific Computing," DARPA HPCS presentation, 2004.
6. See http://www.netlib.org/scalapack/.
7. See http://www.netlib.org/blacs/
8. See http://www.cs.sandia.gov/Zoltan/.
9. R. S. Baker and K. R. Koch, "An SN Algorithm for the Massively Parallel CM-200 Computer," Nuclear Science and Engineering, 128, pages 312–320, 1997.
10. T. R. Jones, L. B. Brenner, and J. M. Fier, "Impacts of Operating Systems on the Scalability of Parallel Applications," Tech. Rep. UCRL-MI-202629, Lawrence Livermore National Laboratory, Mar. 2003.
11. F. Petrini, D. J. Kerbyson, and S. Pakin, "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q," Proceedings of the ACM/IEEE SC2003 Conference (SC'03), 2003.
12. Ron Brightwell, Rolf Riesen, Keith Underwood, Trammell Hudson, Patrick Bridges, and Arthur B. Maccabe, "A Performance Comparison of Linux and a Lightweight Kernel," Proceedings of the IEEE International Conference on Cluster Computing (Cluster2003), December 2003.
13. D. A. Knoll , D. E. Keyes, "Jacobian-Free Newton-Krylov Methods: A Survey of Approaches and Applications," Journal of Computational Physics, volume 193, pages 357-397, 2004.

APPENDIX

QUESTIONNAIRE ON SCIENTIFIC APPLICATION REQUIREMENTS FOR LEADERSHIP COMPUTING IN THE NEXT DECADE

Members of the Scientific Computing Group in the National Center for Computational Sciences (NCCS) Leadership Computing Facility (LCF) at Oak Ridge National Laboratory (ORNL) surveyed numerous computational scientists in a broad range of scientific domains and asked them to speculate on requirements for their scientific application(s) on Leadership Computing platforms in the next decade (2010–2020). A large fraction of the information, guidance, and plans outlined in this document is derived from the answers provided in these surveys from this expert community of leading computational scientists. Without their insight, knowledge, and experience, the application requirements outlined in this document would not have nearly the fidelity or significance.

Time constraints did not permit all 2007 ORNL LCF INCITE Projects to respond to this survey, but the following list of scientists were solicited and able to participate: Pratul Agarwal, Valmor de Almeida, Jeff Candy, Jackie Chen, David Dean, John Drake, Tom Evans, Robert Harrison, Lei-Quan Lee, Peter Lichtner, Tommaso Roscilde, Benoit Roux, Thomas Schulthess, Ed Uberbacher, Phil Locascio, Patrick Worley, Fred Jaeger, Anthony Mezzacappa, William Tang, Wei-li Lee, and Don Batchelor (omission of any names on this list is unfortunate and unintensional).

The survey questions are itemized below.

- What are some possible science drivers and urgent problems that would require Leadership Computing in 2010–2020? Please provide bullet items.

- What are some looming computational challenges that will need resolution in 2010–2020? Please provide bullet items.

- What are some sample science objectives and outcomes that Leadership Computing could enable in 2010–2020? Please provide bullet items.

- What are some improvement goals for science-simulation fidelity that Leadership Computing could enable in 2010–2020? Please provide bullet items, in terms of absolute resolution, relative increases, additional physical processes, etc.

- What are some possible changes in physical model attributes for Leadership-Computing applications in 2010–2020? Please provide bullet items, in terms of numbers of dimensions, numbers of state variables, numbers of diagnostic variables, etc.

- What major software-development projects could occur in your application area in 2010–2020? Please provide short descriptions and implications for Leadership Computing.

- What major algorithm changes could occur for your applications in 2010–2020? Using the “seven dwarfs” categorization, indicate which dwarfs might be added or eliminated: structured grids, unstructured grids, fast Fourier transforms (FFTs), dense linear algebra, sparse linear algebra, particles. Monte Carlo, and other:

- What libraries and development tools may need to be developed or significantly improved for Leadership Computing in 2010–2020? Please provide bullets summarizing the new capabilities needed.

- How might system-attribute priorities change for Leadership Computing for your application? For the following system attributes for Leadership Computers, please select four that are most likely to increase (+) in priority in 2010–2020 and four that are most likely to decrease (–), relative to today: node peak flops; mean time to interrupt (MTTI), wide-area network (WAN) bandwidth, node memory capacity, local storage capacity, archival storage capacity, memory latency, interconnect latency, disk latency, interconnect bandwidth, memory bandwidth, and disk bandwidth.

- In what ways might or should your workflow in 2010–2020 be different from today? Please provide a short description or bullets.

- Are there any “disruptive technologies,” “game changers,” or “revolutions” that might affect your Leadersip Computing applications? Please provide short descriptions or bullets.