Breakout Group 1:

Technical challenges and needs of academic and industrial software infrastructure research and development

Breakout Session Chair: Satoshi Matsuoka (Tokyo Tech./NII), Kathy Yelick (NERSC/LBL, UC-Berkeley)
Breakout Session Secretaries: Michael Heroux (Sandia NL), David Skinner (NERSC/LBL)

Original Slides: Satoshi Matsuoka and Jack Dongarra (UTK/ORNL)

IESP Workshop 2, June 28-29, Paris, France

Subgroup members

Intra-node Discussions: Kathy Yelick, Mike Heroux, Barbara Chapman, Mitsuhisha Sato, Mateo Valero, Jesus Labarta, Luc Giraud, Serge Petiton, John Shalf, Thomas Sterling, Jack Dongarra, Pete Beckman, Bernd Mohr, Jeff Vetter, Bill Gropp, Anne Trefethen, Vladimir Voevodin

Inter-node discussions: Satoshi Matsuoka, David Skinner, Franck Cappello, Barbara Chapman, Alok Choudhary, Sudip Dosanjh, Yutaka Ishikawa, Barney MacCabe, Bob Lucas, Mateo Valero

Objectives of Group1 at This Meeting

- Roadmaps for open software R&D towards exascale
  - From 2009 to 2020
- Important Software Component Identifications
  - Existing Components – how far do they scale?
  - Missing Components – how do we start R&D?
- (Collaboration scenarios)
- (Vendor relations – acceptance, support, etc.)
  - Sufficient "exascale" market?
  - Or, more widespread demand and community forces leveraged?

2009/05/21

Dimensions of petascale software roadmaps

- Time – Petascale now 2009 to Exascale 2018-2020
  - Yearly timeline
- System and other software components/concerns
  - E.g., programming, HA/FT, libraries, I/O & filesystems
- Architectural Diversities
  - Homogeneous vs. heterogeneous cores/ISA
  - Multithreaded shared memory vs. distributed memory
- Target system sizes---peta to exascale
  - # of compute cores
  - # of nodes
  - Other parameters, such as memory, NW, and I/O

Issues facing the software components

- Extreme parallelism and Scale
- Tightening memory/bandwidth bottleneck
- High Availability despite continuous faults
- Huge power / facility requirements
- Heterogeneity and other complexities

Tightening memory/bandwidth bottleneck

Future Architecture Trends, or

the "n^2 (component density) vs. n (I/O BW) problem "

- Very Dense computation
  - Vector/SIMD/Multithreading arch.
  - Power consumption and programming the issue

- Good absolute local memory BW
  - 1TB/s per chip soon, fast/opto signaling, 3-D packaging
  - but deepening memory hierarchy

- Relatively poor node I/O channel and NW BW
  - (only) 40Gb/100Gb soon, long distance signaling hard
  - There might be breakthrus, (e.g. planar laser diode emisson), but…

- Very poor Disk Storage BW
  - SSDs are just boosts, no exception to the laws of physics
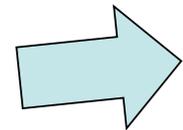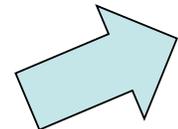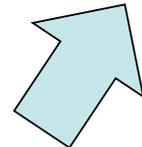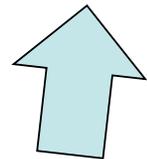
Architecture analysis and strawman targets

- **Target system size and market**
  - 1999/6: 2 systems Rpeak > 1TF, 70 machines > 100GF, 1 rack ~= 64GFlops (80 PentiumIII-S's), x20
  - 2009/6: 3 systems Rpeak > 1PF, 44 machines > 100TF, 1 rack ~= 10TF (200 Nehalem EPs), x100
  - 2019/6: a few exaflops (Rpeak > 1EF) ~28 machines > 100PF, 1 rack 1PF? , x500~1000
- Strawman Architectures circa 2018-20 (need to add memory)
  - Assume scaling down 45nm => 13~15nm, x10 transistors
  - Homogeneous arch: 100 cores/chip (x10), 16 FP issues/core (x4), 3.5 Ghz clock (x1.3) => 5TF/Chip, 10TF/node, 100 nodes/rack
    - 1 rack: 1 PetaF, 20,000 cores, 100 nodes
    - 1 Exa system: 20,000,000 cores (x100 BG)/ 100,000 nodes (x10 BG) / 1,000 racks (x4 ES)
  - Heterogeneous arch: 2500 (simple SIMD-Vector) cores/chip, 4 FP issues/core (x2), 2Ghz (x1.3) => 20TF/chip, 40TF node, 50 nodes/rack
    - 1 rack: 2 PetaF, 250,000 cores / 50 nodes
    - 1Exa system: 125 million cores / 25,000 nodes / 500 racks
  - May want to consider memory, power and other parameters

Software Components

- High Availability/Fault Management
  - Prevention, Tolerance, Detection, Recovery e.g., checkpointing
- Programming Languages and Models
  - Traditional: OpenMP + MPI
  - PGAS languages and variants
  - Accelerator languuages: CUDA, OpenCL
  - Others? (Locally-synchronous languages
- Compilers and Runtime Systems
  - Support for speculative computing, transactional memories, high asynchrony
  - Performance monitoring and feedback, auto-tuning
  - Debugging, verification, etc.
- IO and Filesystems (or perhaps more generally persistent storage models)

- Low level OS and Systems issues
  - virtualization, fault mgmt, memory mgmt, power mgmt, jitter/timing, …
- Numerical Libraries
  - (Lots of issues here, but should be prioritized by applications requirements)
- Systems Management and Configuration
  - (Goal should be to make future systems easier to manage than current Petascale systems)
- Networking and Integration with Broader Infrastructures
  - (Making these systems integrate with other things in the environment … whether its clouds, global filesystems, real-time data streams, etc.)

Group 1 plans

- Initially roughly agree on architectural roadmaps circa 2012 (10-30PF), 2015-6(100-300PF), 2018-20(1EF) (10-15 min)
- Split into two groups: (5 min => until 3:50 PM)
  - Intra-node issues: programming models, concurrency and fine-grained resource mgmt., languages, node OS
  - Inter-node issues: HA/resiliency/FT, Power, global config. Mgmt., I/O
  - Prioritarize discussions, identify cross-subgroup issues
- Reconvene at the end of the day and identify the cross-subgroup issues, to prepare for tomorrow (10 min)

Technology Assumptions

Year	Technology	#cores / socket	Issues /core	Nodes	Memory	Storage	Network
2012	32-28nm	8-16	4-8	10,000		HDD+SSD	100Gb
		500-1000	2	5000			
2015-16	18-15nm	24-48	8-16	30,000	3-D SOC?	SSD+HDD	SOC?
		1000-2000	2	15,000			
2018-2020	9-13nm?	100-200	8-16	100,000	3-D SOC	SSD	SOC
		2000-4000	2	50,000			

Roadmap Formulation Strategy (strawman) for IESP2

- Consider each software component / area, in operation at centers or close to deployment
- If standard / open source component exists
  - Then investigate status quo circa 2009 wrt scalability
  - If project exists to enhance scalability
    - Then identify roadmap until project termination
    - If need to continue then identify the timeline gap till 2018-20/exa
  - Else (R&D gap identification)
    - Identify research challenges envision project req.
    - Attempt to create scalability timeline to 2018-20 exa
- Else (component does not exist in open source)
  - Identify why the component does not exist
  - Conduct R&D gap identification as above

2009/05/21

Collaboration Scenarios

1. Almost no collaboration
   - Periodic workshops, status reports of regions
   - Voluntary and ad-hoc usage of products of various projects

2. Loosely coupled collaboration
   - Focused meetings & workshops covering respective components & concerns of the software stack
   - Comparison of technical milestones, esp. for similar developments and application usage
   - Cross pollinating deployments of

3. Collaboration with Standardization
   - Definition of standards, test suites, and benchmarks, and their public availability

4. Tightly Coupled collaboration
   - International governance & funding structure
   - Cross-continental development teams (e.g., LHC/EGEE)

Roadmap Requirements (by Jack)

- Specify ways to re-invigorate the computational science software community throughout the international community.

- Include the status of computational science software activities across industry, government, and academia.

- Be created and maintained via an open process that involves broad input from industry, academia and government.

- Identify quantitative and measurable milestones and timelines.

- Be evaluated and revised as needed at prescribed intervals.

- Roadmap should specify opportunities for cross-fertilization of various agency activities, successes and challenges

- Agency strategies for computational science should be shaped in response to the roadmap

- Strategic plans should recognize and address roadmap priorities and funding requirements.

Research Topics to consider (by Jack)

- Contributors
- Priorities
- Existing expertise
- SW sustainability
- Developing new programming models and tools that address extreme scale, multicore, heterogeneity and performance
- Develop a framework for organizing the software research community
- Encourage and facilitate collaboration in education and training

2009/05/21

Roadmap/Milestone

	2009	2010	2011	2012	2013	2014	2015	2016
Software/ Language Issues								
Sustainability								
Collaborative workshops								
Coordinated research								
Educational activities								
Standards activities								
Priorities								
Staffing								

Group 1: Software Infrastructure Breakout

Day 1

2009/06/28

Group 1 Initial Discussion

2009/06/28

Strawman Roadmap Discussion (see Tech Assumptions Slide)

- Memory requirements:
  - 3D layout in 2015.
  - Amount/core uncertain.
  - On-chip vs. off-chip performance difference grows.
- Migration from HD to SSD: How?
- Architecture support for GAS: Coming at some point.
- Node memory architecture:
  - SOC.
  - Deep memory hierarchy.
  - On-chip cache coherence, infeasible?
- Node PM: Will industry solve?
- Side comment: Bechtoldsheim: Up to half of industry might be in HPC.
- FP precision: Role in this discussion?

Software Breakout Group 1A: Intranode Programming Model Day 1

Kathy Yelick, Mike Heroux, Barbara Chapman, Mitsuhisha Sato, Mateo Valero, Jesus Labarta, Luc Giraud, Serge Petiton, John Shalf, Thomas Sterling, Jack Dongarra, Pete Beckman, Bernd Mohr, Jeff Vetter, Bill Gropp, Anne Trefethen
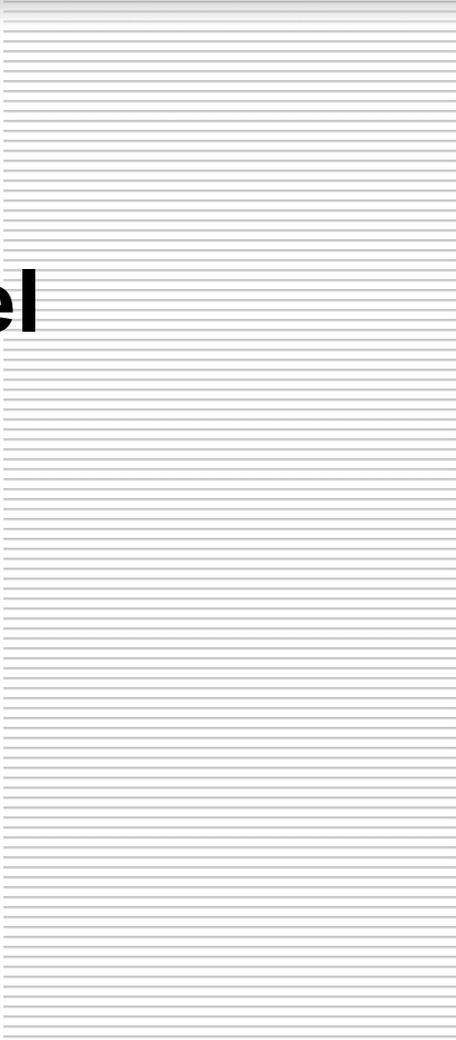
Agreement

- Thousands of functional units on a socket
  - 100s of cores with 10s of FPUs per core (data parallel hardware)
  - OR, 1000s of cores
  - OR, Lightweight PIM cores; heavy weight cores with data parallel instructions and temporal locality needs
- DRAM Memory per FPU capacity will drop
  - Memory per socket will go up, but will not keep up with FPU growth
- "Core" is not well-defined
  - PC / FPU >> 1 → multithreaded architecture
  - PC / FPU << 1 → vector/SIMD architecture
- Cache coherence across a 2020 socket is not feasible

Agreement

- ## Memory bandwidth "wall" will be solved
  - Yes, technology will exists and market forces will cause this to happen (even outside HPC)
    - Photonics, 3D stacking,…

- ## Memory latency problems will be solved by hardware and algorithms
  - Need massive concurrency, lightweight threading combined vectors
  - 1B way concurrency including threads, vectors, and/or prefetching

Programming Model Requirements

- ## Sustainability (13)
  - Backward compatibility through interoperability
    - Mixed language.
  - Incremental adoption support through interoperability
  - Composable
  - Portable and standardized
  - Develop in collaboration with industry
  - Familiarity with old models (no Haskel, remember that we are Fortran/C/C++ programming community).

Programming Model Requirements

- Address load imbalance (from the hardware, faults, and the applications/algorithms) (12)
  - Dynamic concurrency generation (Do you need it and can you afford it?)
  - Express more parallelism than you need (e.g., hierarchical task graph)
  - Dynamic runtime
  - Less synchronous than fork/join or SPMD
  - Dynamic runtime that relaxes to efficiency of a static model when dynamic is not needed.

Programming Model Requirements

- Ability to manage locality and data movement (9)
  - Make efficient use of bandwidth.
  - Scheduling for latency hiding
  - Ability to specify data scope and locality
  - Support for scratch pad memory
- Performance transparency and feedback (6)
  - Including runtime adaptation
  - Performance aware computing.

Programming Model Requirements (re-vote these?)

- Multiple precision support: ½, full, extended (3)
- Fault detection and ability to respond (1) Leave to inter-node sub-subgroup?.
  - E.g., transient memory errors
- 1B way concurrency (1)
  - Assumed true as part of memory wall solution.
- Support for strong scaling
  - Global address space, but it is a challenge at exascale (1)
- Avoiding synchronization
  - Message-driven computation for hiding communication and synchronization latency (1)
  - One sided communication and active messaging, dataflow tokens
  - Lightweight synchronization objects

Programming Model Requirements (re-vote these?)

- User-defined distributed data structures
- Energy management support (queries)
- Overlapping communication and computation

Issues for a Programming Model Program

- Some of these problems will be solved by industry
- Support for multiple hardware solutions
- If we did nothing, where would we get (MPI + CUDA?)
- If we did a lot, are there things we could not solve?

Why is 2009 unlike 1999?

- Transition from Gigascale (vectors) to Terascale (clusters) required software revolution
- The transition from Terascale to Petascale did not
- The transition from Petascale to Exascale will require another (harder) software revolution:
  - No clock speed scaling; all performance from concurrency
  - Energy bill for large machine (for 5 years) ~= hardware cost
  - Linpack is now a reliability benchmark
  - Even homogeneous machines appear heterogeneous
  - Ubiquitous code change (vs. MPI which required only new infrastructure).
- Separate
  1) The on-chip concurrency problems
  2) Between chip availability problems
  3) Other software model and ecosystem problems:
     multi-physics/scale/institutions/languages and sustainability

Breakout Group 1B:
Technical challenges and needs of academic and industrial software infrastructure research and development; software scaling

Breakout Session Chair: Satoshi Matsuoka (Tokyo Tech./NII)
Breakout Session Secretary: David Skinner (Berkeley NL)

Members: : Satoshi Matsuoka, David Skinner, Franck Cappello, Barbara Chapman, Alok Choudhary, Sudip Dosanjh, Yutaka Ishikawa, Barney MacCabe, Bob Lucas, Mateo Valero, Vladimir Voevodin

IESP Workshop 2, June 28-29, Paris, France, Day 1

Inter-node topics

- I/O, I/O, I/O
- Resiliency (RAS,HA, Fault prevention, detection, recovery, management)
- Checkpointing
- Virtualization, Dynamic Provisioning, Partitioning
- Performance monitoring, feedback, parallel autotuning
- Resource provisioning (including heterogenous nodes, workflow scheduling)
- Parallel programming models (distributed memory and PGAS)
- Parallel debugging
- Systems integration (SW bringup, factory testing)
- Systems management (realtime, config management,)
- Interaction with external resources : Clouds, Global FS (or non-FS), archiving, real-time data streams
- Power and Facilities (SW only)
- Security

2009/05/21

Resiliency

- Faults everyday in PetaScale systems
  - Need to cope with continuous stream of failures
  - SW errors & HW errors, memory errors may dominate
  - MTTF < MTTC (mean time to checkpoint)
- Existing CPR falls short (full MPI CPR scales to 1k cpus)
  - Proactive actions (RAS analysis, Fault prediction)
  - Silent Errors (faults not monitored)
  - Lack Hardware Support (Diagnostics/Interfaces, SSDs?)
- Actions
  - Reduce number of errors, describe errors better
  - Make applications more resilient
  - Make systems situationally aware
- Roadmap & Research (immediate needs in 2012, figure out 2018 in 2012)
  - Fault oblivious algorithms, Error tolerant algorithms
  - Tunable advisement about key data/computations/communications to protect/check.
  - Relaxed model of correctness (from consistency)
  - Need C-PAPI for faults, make errors less silent
  - If faults are reproducible, inject them to test resiliency layers
  - Advanced diagnostics (monitor well-being of system, autonomous correction)

Software Infrastructure Breakout: Day 2

2009/06/29

Group 1 plans

- Initially roughly agree on architectural roadmaps circa 2012 (10-30PF), 2015-6(100-300PF), 2018-20(1EF) (10-15 min)

- Split into two groups:
  - Intra-node issues: programming models, concurrency and fine-grained resource mgmt., languages, node OS
  - Inter-node issues: HA/resiliency/FT, Power, global config. Mgmt., I/O
  - Identify topics
  - Prioritarize the topics and discussions
  - Identify cross-subgroup issues

Software Breakout Group 1A: Intranode Programming Model

Kathy Yelick, Mike Heroux, Barbara Chapman, Mitsuhisha Sato, Mateo Valero, Jesus Labarta, Luc Giraud, Serge Petiton, John Shalf, Thomas Sterling, Jack Dongarra, Pete Beckman, Bernd Mohr, Jeff Vetter, Bill Gropp, Anne Trefethen, Vladimir Voevodin

Solved Problem, Unsolved Problems, and Unsolvable Problems in
Exascale Software

With a focus on problems within a socket
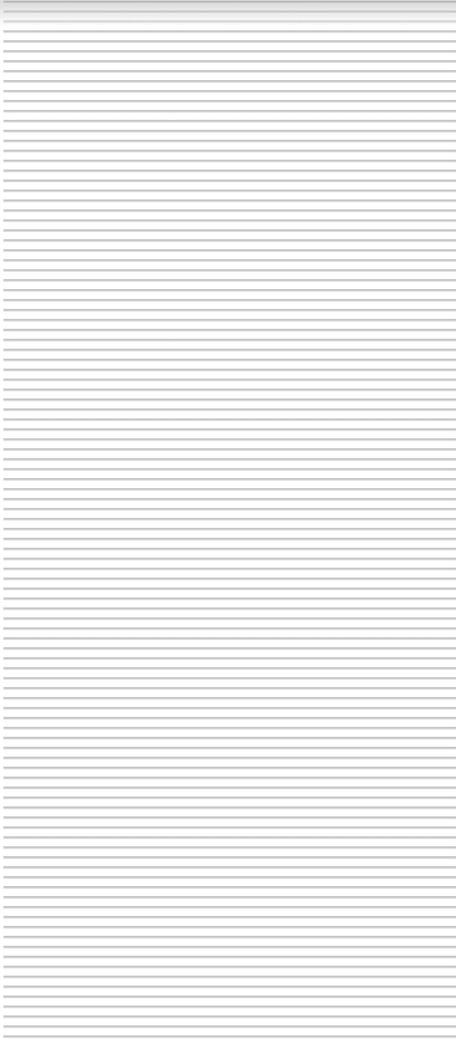
High level topics

- 1B concurrency and load balance (Thomas Sterling, Jesus Labarta)
- Locality and distributed data structures (Barbara Chapman, Vladimir Voevodin, Mitsuhisa Sato)
- Sustainability, especially interoperability (Bill Gropp, Mike Heroux)
- Operating systems (Thomas Sterling, John Shalf)
- Algorithms (Jack Dongarra, Anne Trefethen, Serge Petiton, Luc Giraud)
- Misc: performance.. (Bernd Mohr, Jeff Vetter, David Keyes)
- Fault tolerance (Kathy Yelick)

2009/05/21

B way parallelism and load balance (Thomas Sterling, Jesus Labarta)

- Situation:
    - Need parallelism to feed the foreseable B cores hardware
    - Need further paralleism to let them tolerate latency
    - Dynamic scheduling for load balancing to tolerate not only algorithmic imbalances but the variance we are going to observe in platforms
    - There will be hierarchy both in algorithms and platforms (will they match?)
    - Need Low Overhead for synchronization and dispatch
- State of the art:
    - Generally static, compile/submission time specified
    - Based on preconception of knowledge of problem and machine
    - 100K processes and heterogeneous accelerators
    - Fork-join / spawn –wait / point to point synch + globally synchronizing (Zoltan, J-Machine,..)
- Needed Actions:
    - Develop advanced models of parallel programming models to expose dynamic parallelism
    - Develop advanced flow control model including advanced synchronization semantics and dependence handling mechanisms.
    - Runtime adaptive mechanisms and policies that converge to static if possible resource allocation
    - Methods for self aware resource allocation for dynamic load balancing
- Roadmap & Research: (immediate needs in 2012, figure out 2018 in 2012)
    - APIs for exposing fine grain/dynamic parallelism and enabling lightweight synchronization
    - Policies resource allocation and load balancing
    - Prototyping barebones parallel translation and runtime on "some" heterogeneous multicore

Managing Data Locality and Distributed Data Structures
(Barbara Chapman, Vladimir Voevodin, Mitsuhisa Sato)

- Situation: Locality essential for programming and performance in PetaScale systems, will be more complex in Exascale
  - Extreme number of threads and memory distributed among nodes, cores and devices
  - Complex cache hierarchy and resource sharing among cores
  - New memory technologies are coming e.g. 3d stacking
  - Explicit data transfer essential for use of accelerators
- State of the art: Programming models have different approaches to supporting locality
  - Implicit and explicit distribution of data structures (e.g. MPI, PGAS languages, HPCS languages)
  - Alignment of work and data (e.g. loop iteration mapping, Locale/place)
  - Explicit data transfer between devices
  - Use of manual program transformations to increase locality e.g. cache blocking
- Needed Actions:
  - Provide a model for the expression of scope and locality at both algorithmic and application code levels, especially for global view programming
  - Develop techniques and features to enable efficient use of bandwidth and to support latency hiding
  - Develop techniques for automatic optimization of data motion where possible, but user control for performance-aware programming
  - Explore both implicit and explicit models for accomplishing locality including analyzable code
- Roadmap & Research: (immediate needs in 2012, figure out 2018 in 2012)
  - Features and notation for describing locality, e.g. algorithmic locality, user-defined distributed data structures, alignment of work and data, task mappings
  - Develop support for data migration through non-conventional memory hierarchies (accelerators, scratchpads)
  - Create tools for measuring, detecting, improving and exploiting locality
  - Improve system-level approaches to managing locality
  - Implicit locality models and automated locality support in long term
  - Integrate with novel models for achieving concurrency and fault-tolerance for fine-grained state preservation recovery

Managing Data Locality and Distributed Data Structures (cont.)

- Benefits
  - For prefetch (identify data ahead of time)
  - For software controlled memory (know what data needs to be copied in so you can set up your DMA transfers). Prefetch is just a different implementation
  - For layout on-chip to reduce contention for shared resources: (because even on-chip there will be locality constraints will affect performance location of topological neighbors in a chip multiprocessor)
  - For fault resilience (explicitly identify what data is changed by unit of computation so you know when it needs to be preserved)
  - When dependencies are analyzed at even coarse-grained level, more freedom to reorder program units to increase slack for communication latency hiding and reorder for reuse between program units (not just) can restructure/reschedule at a program level
  - Also enables functional partitioning to express more concurrency (makes it easier to create feed-forward pipelined parallelism when domain-decomposition reaches its limits

2009/05/21

Algorithms & Software Libraries (Jack Dongarra, Anne Trefethen, Serge Petiton, Luc Giraud)

- Situation: Algorithmic problems everyday in PetaScale systems, Exascale will be worse
  - Accumulation of round-off errors
  - Adaptivity for architectural environment
  - Fault resistant algorithms – bit flipping and loosing data (due to failures). Algorithms that detect and carry on or detect and correct and carry on (for one or more)
  - Scalability : need algorithms with minimal amount of communication
  - Coupling of multi-scale and multi-physics codes
  - Amounts of data will increase (pre an post processing)

- Roadmap & Research: (immediate needs in 2012, figure out 2018 in 2012)
  - Fault oblivious algorithms, Error tolerant algorithms
  - Hybrid and hierarchical based algorithms (eg linear algebra split across multi-core and gpu, self-adapting)
  - Mixed arithmetic
  - Energy efficient algorithms
  - Algorithms that minimize communications
  - Autotuning based on historical information
  - Architectural aware algorithms/libraries
  - Error propagation and sensitivity across mathematical and simulation models
  - For application drivers identify key algorithmic areas

Sustainability (Bill Gropp, Mike Heroux)

- Situation
  - Huge software base
    - Need to evolve legacy apps
  - Need to introduce new models/approaches to address unique Exascale issues
    - Need adoption strategy
  - Industry recognition of multicore crisis in programming models
- State of the Art
  - MPI + C/C++/Fortran + OpenMP and/or pthreads etc.
  - UPC, CAF; HPCS languages (research efforts); etc.
  - Much of computational science primarily in serial parts of code
  - No guarantee of interoperability between programming models
  - Scalability demonstrated upto 100K nodes
- Needed Actions
  - Need effective multi/many core programming model(s)
  - Need standards and/or mechanisms for efficient interoperability (no copies)
  - Need interoperability with tool chain (debuggers, performance, OS, I/O)
  - Determine division between industry and Exascale community
- Roadmap and Research
  - Research: Find commonality between models; common framework for describing interactions
    - Memory model, synchronization model, etc.
  - Research: Enhance (not replace) commodity multicore model for Exascale requirements (e.g., fault handling)
  - Research: Tools to migrate legacy code to new models (e.g., exploit heterogeneous arch)
  - Roadmap (short): Identify features to add to commodity programming models (incl MPI) and mechanism
  - Roadmap (short): Identify issues in interoperability and composition
  - Roadmap (long): Define and construct prototypes implementing interop and composibility in select pgm models
  - Roadmap (long): Define and construct prototypes implementing tool chain/development environment

Operating Systems (Thomas Sterling, John Shalf)

- Situation: Operating systems were designed with single processor or SMP node model
  - Do not cope with heterogeneous hardware and nonconventional memory structures
  - Poor scaling efficiency as cores/hardware added
    - Serial path for exception handling (does not scale)
    - Global locks for shared resources
  - Weak notion of locality and performance isolation
  - Requires cache-coherence and homogeneous ISA to work
  - Unrecoverable fault result in kernel panic (reboot to recover from CPU error)
  - Applications and runtime have very limited control of scheduling policy and resource management (OS interposes self with context switch for each protected/hardware resource request)
- State of the art: Linux of various flavors assumes homogeneous shared memory system
  - Hierarchical OS (offload OS calls to "service handlers"): e.g. Plan 9
  - Lightweight Kernels (limited functionality, controls OS noise, mem footprint) e.g. CNK or CNL
  - Full kernels (Full functionality, but complex, large memory footprint, OS noise) e.g. Linux
- Needed Actions:
  - Need to provide applications and runtime more control of the policy (scheduling & resource)
  - Remove OS from critical path for access to resources (grant protected bare-metal access path and then get out of the way)
  - Develop global namespace management
  - Interoperability between local functionality and global functionality (e.g. make TLB be integrated with global memory model, global resource discovery and namespace management)
  - Need to support for managing heterogeneous computational resources and non-cache-coherent memory hierarchies
  - Expose mechanisms for finer reporting and control of power management (provide to app and runtime)
  - Scalable parallel, locality-aware, interrupt dispatch mechanism
  - Develop QoS mechanisms for managing access to shared resources (on-chip networks, memory bandwidth, caches)
  - Scalable mechanisms for fault isolation, protection, and information propagation to application and runtime on-chip (for transient hardware errors and software errors)
- Roadmap & Research: (immediate needs in 2012, figure out 2018 in 2012)
  - Establish performance metrics and models to quantify scaling opportunities and robustness for global OS model
  - Remove OS from critical path for hardware policy access
  - Define asynchronous API for system calls with abstract service location for satisfying calls and global namespace approach.
  - Derive an experimental platform (simplified OS) with strawman functional elements and interrelationships to facilitate exploration and quantification of competing mechanisms for managing OS concurrency. Quantify benefits, complexity, and hardware support requirements for competing approaches.
  - Implement test X-OS experimental platform on medium/large scale testbed to integrated with global OS namespace and management features

Performance (Bernd Mohr, Jeff Vetter, David Keyes)

- Situation:
  - Functionality, Correctness, and only then, performance is taken care of (Telescoping)
  - Too manual and labor-intensive
  - Limited applicability and usage of performance models
- State of the art:
  - Simple statistical summaries, at best snapshots over time
    - Can handle 25K events/s per thread => 5min, 64k threads => 2-10TB, mainly only thread count will increase
  - Emphasis on data presentation rather than on analysis and necessary optimization
- Needed Actions:
  - Performance-aware design, development and deployment
  - Integration with compilers and runtime systems
  - Support for performance observability in HW and SW (runtime)
  - Need more intelligence in raw data processing and analysis
  - Support for heterogeneous hardware and mixed programming models
- Roadmap & Research: (immediate needs in 2012, figure out 2018 in 2012)
  - Make sure can handle envisioned number of threads in 2012, 2015, 2018
  - Integrate performance modeling, measurement, and analysis communities and agendas
  - Ensure performance-aware design of hardware, system software, and applications

2009/05/21

Programming Model Support for Fault tolerance (Kathy Yelick)

- Situation: Faults everyday in PetaScale systems, Exascale will be worse
  - Need to cope with continuous stream of failures
  - SW errors & HW errors, memory errors may dominate
  - MTTF < MTTC (mean time to checkpoint)
  - *Need to distinguish failures from full system interrupts*
  - *Detection problem alone will be major challenge*
- State of the art: *Programming models assume fault free*
  - *Research on fault tolerance MPI*
- Needed Actions:
  - *Programming model support for fault detection*
  - *Programming model support for recovery (transactions, retry,…)*
- Roadmap & Research: (immediate needs in 2012, figure out 2018 in 2012)
  - *2012: Model needed for classify faults and ability to tolerate (2012)*
  - *2015: Languages and compilers for hardware faults (2015)*
    - *Memory errors first*
  - *2018: Languages, compilers and tool support for software faults (2018)*
    - *E.g., retry for rarely found race conditions; Parallel debugging of 1B unsolved*

Breakout Group 1B:
Technical challenges and needs of academic and industrial software infrastructure research and development; software scaling

Breakout Session Chair: Satoshi Matsuoka (Tokyo Tech./NII)
Breakout Session Secretary: David Skinner (Berkeley NL)

Members: Satoshi Matsuoka, David Skinner, Franck Cappello, Barbara Chapman, Alok Choudhary, Sudip Dosanjh, Yutaka Ishikawa, Barney MacCabe, Bob Lucas, Mateo Valero

IESP Workshop 2, June 28-29, Paris, France, Day 2

Inter-node topics

- I/O, I/O, I/O (HD/SSD, local, global parallel FS, non-FS) (1) (choudhary, ishikawa)
- Resiliency (RAS,HA, Fault prevention, detection, recovery, management) (1) (cappello)
- Virtualization, Dynamic Provisioning, Partitioning (3) (maccabe)
- Performance monitoring, feedback, parallel autotuning (2) (skinner)
- Resource provisioning (including heterogeneous nodes, workflow scheduling) (2)
- Parallel programming models (distributed memory and PGAS) (1)
- Parallel debugging (?, what is the programming model) (sudip)
- Eliminate bottlenecks to strong scaling (hidden latencies in SW) (1) (lucas, nakashima)
- Systems integration (SW bringup, factory testing, transition to production SW state) (3) (skinner)
- Systems management (realtime, config management, SW change management) (3)
- Interaction with external resources : Clouds, archiving, real-time data streams (2)
- Power and Facilities (SW only, thermal process migration, energy scheduling/charging) (2) (matsuoka)

1) High priority research, need new ideas
2) Near term research gap, need new solutions
3) Implementation gap, need to adapt solutions/methods

I/O (Alok Choudhary, Yutaka Ishikawa)

- Situation: Scalable I/O is critical - Scalability problems
  - Programming and abstraction (how is I/O viewed from 100K+ processes), Is the file I/O abstraction necessary (e.g., what about high-level data persistence models including databases)
  - S/W Performance and optimizations (BW, latency)
- State of the Art
  - Applications use I/O at different levels, in formats, using different number of layers
- Needed Actions
  - Think differently? Purpose of I/O (e.g., checkpointing at OS/application level, persistent data storage, data analytics, use it and throw away?); customized configurations?
  - Define architecture hierarchy abstraction from S/W perspective
- Roadmap and Research (Immediate Need, Intermediate, Long Term)
  - Newer models of I/O (high level, DB, elimination of dependencies on number of nodes)?
  - Exploitation of new memory hierarchy (e.g., SSD) for S/W layers, optimizations
  - Power/performance optimizations in I/O
  - Intelligent and proactive caching mechanisms
  - Integration of data analytics, online analysis and data management
  - Data provenance/management
  - Derive I/O requirements from users or workload analysis

Parallel applications

Data structures, attributes types, user annotations
Data partitioning patterns
Data byte streams

High-level I/O libraries e.g. PnetCDF, HDF5

Data partitioning patterns
Data byte streams

MPI-IO

Data byte streams

POSIX-IO

Striped byte streams

Parallel file system
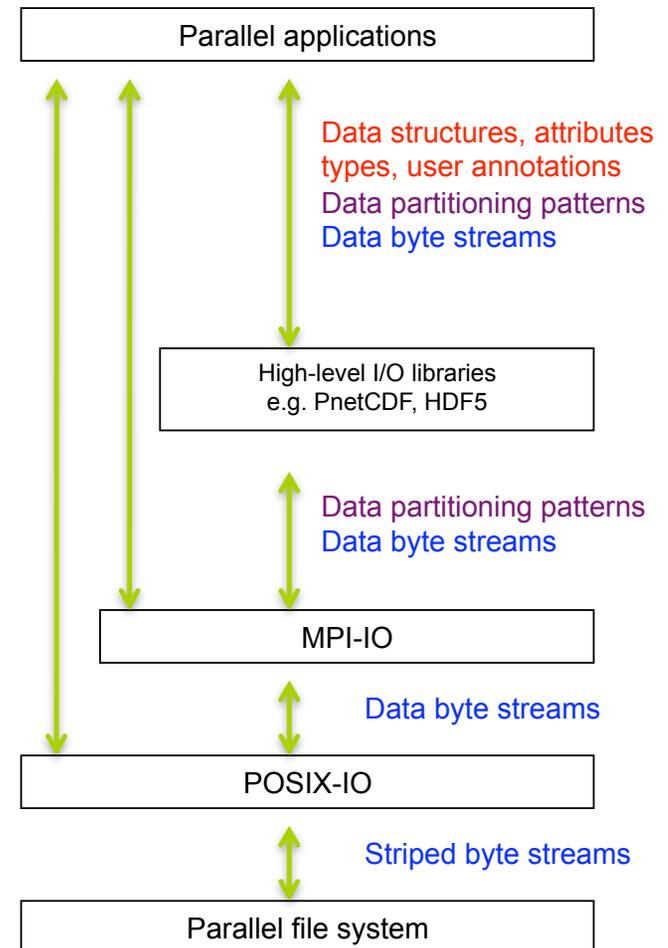
Parallel Debugging (Sudip Dosanjh)

- Situation: Significant topic of research for many years
  - Tools are available for applications with 1,000 MPI tasks
  - Very few applications execute the first time on 10's of thousands of cores (even very mature, widely used codes)
  - Debugging usually requires lots of time by expert parallel programmers
- State of the Art:
  - TotalView, Allinea's Distributed Debugging Tool
  - Early work on a light-weight debugger
- Needed Actions:
  - Current methods will not scale to exascale. Most application programmers don't want to debug a code with 100,000 or 1,000,000 MPI tasks. A fundamentally new paradigm is needed.
  - Automated tools and formal methods
- Roadmap&Research
  - 2010: Suggestion for an applications readiness team workshop, plan for community building/ information sharing
  - 2012: Light-weight debuggers are needed that can supply limited information for 100,000 MPI tasks
  - 2012: Simulation/testing tools are needed for large task counts -- i.e., so programmers can test their codes on O(1M tasks) on smaller systems
  - 2015: Breakthrough needed for O(1M) tasks
  - 2018: Near-production Exascale tools

Performance Monitoring and Workload Analysis (David Skinner)

- Situation: At petascale application walltimes are variable and mysterious, exascale
  - Workloads are studied anecdotally based on narratives or very limited data
  - Performance often ascribed to an application as opposed to a series of runs on a specific machine
  - Performance engineering done ex-situ sometimes away from users and production setting
  - Good serial interfaces for perf data (PAPI), but with limited big picture view (no parallelism)
  - Inter-job contention almost unstudied, poorly understood, aggravating at petascale toxic at exascale

- State of the art:
  - Performance monitoring of HPC resources lags that seen in autos (a dashboard)
  - A spectrum of tools are needed (low overhead profiling, intelligent tracing, deep dive perf debug)
  - Low overhead (< 2%) application profiling available at terascale, barely working at petascale
  - Tool scaling varies from easy to use to heroic efforts, no one tool will meet all needs
  - Asymptotically poor performance (failure) often undiagnosable, trial and error approach

- Needed Actions:
  - Tools must become composable allowing for deep dive tool use as well as background monitoring
  - Continuous performance reporting required to maintain basic system operation + opt-in tools
  - Pre-emptive verification of requested resource performance prior to job launch
  - Shine light on app/system interactions, system status, contention weather, resource conflicts, wasted resources
  - HPC managers and users need easy to use methods to provide common basis for productive performance dialogue
  - Workload analysis will allows HPC facility managers to better procure, provision, schedule resources to mitigate contention

- Roadmap & Research: (immediate needs in 2012, figure out 2018 in 2012)
  - Extend performance counters to power, cooling, faults, interconnect, filesystem
  - Accurate descriptions of job lifecycle (how long to start, reason failed, resources consumed, actions needed)
  - Integrate multiple levels of monitoring into high level user and system contexts
  - Clustering, compression, stream sketching, and synopsis generation of performance data (web methods)
  - Data mining for performance prediction, workload patterns, and online analysis of system and applications
  - Synthesis of measurements to connect inputs and outputs to research goals (high level metrics)

2009/05/21

Virtualization, Dynamic Provisioning, Partitioning (Barney MacCabe)

- Situation:
  - Virtualization provides "a level of indirection" between the application and the systems
  - Operating systems can exploit this level of indirection to support properties that are not associated with the physical system (e.g., dynamic node allocation, migration after a fault, etc)
  - Isolation between applications is critical when multiple applications share a single system
  - Applications frequently have very subtle dependencies on specific library and/or OS versions, there is a need for applications to "bring their own OS" to the nodes
- State of the art:
  - Applications are allocated a static a virtual machine (partition) at load time
  - Systems like Blue Gene provide direct support for partitioning and isolation in the network, in other systems this partitioning and isolation is a key part of the system software dunning on the nodes
  - Node virtualization is rarely supported, although Blue Gene easily support rebooting nodes when applications are launched
- Needed Actions:
  - Clarify programming models needs
    - what is dynamically provisioned?
    - Do all virtual nodes make progress when the number of virtual nodes exceeds the number of physical nodes?
    - What is the overhead? Can this overhead be eliminated for applications that do not
  - Clarify other benefits that might accrue from node virtulaization
    - Dynamic migration after fault detection or to balance resource usage
- Roadmap & Research: (immediate needs in 2012, figure out 2018 in 2012)
  - Support for, light-weight, node virtualization that supports a common API (e.g., the Xen API)
  - Direct support for light-weight virtualization mechanisms (no node OS) based on small computational units (e.g., Charm++, ParalleX)
  - Better understanding of the needs of computational and programming models

2009/05/21

Power and Management (Satoshi Matsuoka)

- Situation: 100TF-1PF systems at 500KW-6MW today, 100MW or more envisioned for Exascale
  - Beyond Moore's law scaling is pushing power/energy requirements as systems grow larger
  - Power/Energy may become fundamental limiting factor---$10s millions , CO2 footprint
  - Need to drastically reduce energy consumption to be commercially feasible
- State of the art: leveraging some datacenter/notebook power saving features
  - DVFS (Dynamic Voltage & Frequency Scaling) within application
  - System-level resource management tied to scheduling and DVFS
  - Manual optimization of datacenter cooling to reach "reasonable" PUE ~= 1.5-2.0
- Needed Actions:
  - Altenative architectures and devices with fundamentally order(s)-of-magnitude better power-performance characteristics and their exploitation in SW, e.g., GPUs, phase change memory, SSDs, …
  - Measure/predict power&energy, based on underlying sensors and power-performance models
  - Aggressive cooling technologies (e.g., ambient cooling) coupled with machine oprations e.g., packing processes to achieve higher thermal gradient
  - Auto-tune/optimize the entire system for best energy-performance levels, achieving the necessary x10 improvement beyond x100 offered by Moore's law over 10 years.
- Roadmap & Research: (immediate needs in 2012, figure out 2018 in 2012)
  - 2012: various software artifacts to exploit alternative architectures/devices along with their power models, open source sensor/monitoring framework to measure power and thermals on a 10,000 node scale machine
  - 2015: comprehensive system simulation auto-tuning techniques to allow optimization, workload management tied to thermals/energy for energy optimization
  - 2018: scaling of the software artifacts above to reach 100,000 node / 100million core exascale system with sufficient response time for effective energy and thermal control

Exascale Software Integration (David Skinner)

- Situation: System software, middleware, and system configuration settings are brittle
  - During machine bring up finding proper software configuration is a complex expensive search
  - Maintaining an optimal or reasonable state over time, across upgrades is difficult
  - Terascale downtimes are expensive, Exascale brings both greater complexity and cost of outage
  - Users bear the brunt of poorly integrate/configured software
- State of the art:
  - Impact of multiple changes often unverifiable
  - Change management not in widespread use (plan for changes, review changes)
  - Cfengine is not a change management plan
  - Many centers operate without test machines

- Needed Actions:
  - Transition from "out of the box" approach to a "trust and verify"
  - Revise expectations about when integration and testing happens
  - Incremental testing as system is brought up
- Roadmap & Research:
  - Improved software engineering across the board (users, HPC centers, industry partners)
  - Redefinition of acceptance period to allow for software bringup
  - Alternately add a SW integration period
  - Automated SW testing at all levels, connection to SLA/expectations

2009/05/21

Removing Bottleneck to Strong (& Weak) Scaling (Bob Lucas)

- Situation:
  - many algorithms have some O(N) portions which should be severe bottleneck when N grows to 10^8-10^9
  - most scaling today is weak but memory/core will (should) decrease and even if exa-weak-scalable the problem should become TOO large
  - most systems provide communication means with high latency.
- State of the Art
  - MPI & PGAS programming models
  - algorithms (e.g. math kernels) with sequential bottleneck (e.g. dot product) and/or with global state
- Needed Actions = SLOW (thanks Tom)
  - Starvation: expose & manage concurrency
  - Latency: minimize & tolerate (hide)
  - Overhead: minimize & expose fine-grain concurrency
  - Waiting for Contention: remove global barriers
- Roadmap & Research (near, medium & long)
  - algorithms: force/allow app. guys to use nice math library
  - prog. models: global addr. spac, light weight sync., coarse-grain functional or dataflow
  - h/w mechanisms: active memory / messaging
  - near: remove barrier (& other global comm.) and replace with asynchronous global flow control which also is capable to hide global latency.
  - medium: algorithm research to eliminate any O(N) (or higher) portions
  - ultimate goal: to reach real speed-of-light (& physics) limits

2009/05/21

Resiliency (Franck Cappello)

- Situation (problem statement, why a relevant topic):
    - Faults everyday in PetaScale systems, Exascale will be worse
    - Need to cope with continuous stream of failures (applications will have to resist to several errors, of different kinds, during their execution)
    - SW errors & HW errors, Undetected Soft errors (Silent errors) are already a problem. SW errors may dominate
- State of the art (and limitations):
    - Checkpointing on remote file system, however: MTTI <= Checkpoint time for Exascale systems
    - Proactive actions (RAS analysis, Fault prediction), however: 1) how to manage predicted software errors? 2) we need more event traces to improve fault prediction algorithms
    - Silent Errors (faults not monitored) are suspected and sometimes detected afterwards, however their is a need of characterization (what, where, how frequent, etc.)
    - Fuzzy event logging: Some errors are silently corrected (and so not reported) + some errors are reported by humans and not well integrated
    - No coordination between software layers (and errors are not reported across layers)
    - Almost no hardware support for Resilience (except at the node level), however we need to detect more errors and ease the job of the software
    - No experimental platforms
- Needed Actions:
    - Investigate Checkpoint/Restart limitations
    - Make applications more resilient
    - Develop novel application level tunable resilience techniques
    - Develop coordination mechanisms from HW to applications (through all SW layers)
    - Make errors less silent
    - Improve interaction mechanisms between automatic error correction systems and humans
    - Develop experimental platform
- Roadmap & Research: (immediate needs in 2012, figure out 2018 in 2012)
    - HW, SW, Soft, Silent Error characterization in HPC systems (Immediate, but should ve revised periodically)
    - Investigate the current scalability and bottlenecks of MPI checkpoint/restart (immediate)
    - Investigate how to reduce these bottleneck (in-situ checkpoint, hardware support: SSDs, Networks) (immediate)
    - Fault oblivious algorithms, Error tolerant algorithms, ABFT like techniques (medium and long term)
    - Tunable advisement about key data/computations/communications to protect/check. (Relaxed model of correctness -similar to consistency for memory) (immediate and medium term)
    - Need uniform interface for faults, Improve hardware support (more sensors, detectors for Diagnostics/Interfaces) (medium term)
    - Improve error description and report, make systems situation aware, provide advanced diagnostics (monitor well-being of system, autonomous correction) (medium term)
    - Design and implement experimentation platform with sophisticated fault injectors (immediate)

Unresolved Concepts, Issues

- ## Node. What is a node?
  - Two categories: Some vs. many.
  - Important pre-req. Must define.
  - Others: Socket (UMA).
  - Node characteristics: OS focus.
  - One thing is certain: Within a node the PM *must* change. Inter-node PM might not.

- ## Classification of Algorithms:
  - Challenging.
  - Must be included somewhere in these discussions

- ## Intra-node and inter-node interplay.

- ## Evolution-Revolution Discussion.

Unresolved concepts, issues

- Global, global view.
- Connection of proposed topics to application drivers.