Breakout Group 1A: Programming Models at Exascale

With a focus on problems within a socket

Breakout Session Chair: Kathy Yelick (Berkeley LBL)
Breakout Session Secretary: Mike Heroux (Sandia SNL)

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

High level topics

- 1B concurrency and load balance (Thomas & Jesus)
  - heterogeneity
- Locality and distributed data structures (Barbara & Vladimir & Sato)
- Sustainability, especially interoperability (Bill & Mike)
- Fault tolerance (Kathy)
- Operating systems (Thomas & John)
- Algorithms (Jack & Anne & Serge & Luc)
- Misc: performance.. (Bernd & Jeff & David)

B way parallelism and load balance

- 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

- 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

- 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

Algorithms & Software Libraries

- 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

- 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

- 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

- Situation:
  - Functionality, Correctness, and only then, performance is taken care of
  - 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

Programming Model Support for Fault tolerance

- 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: Issues in Scalable Parallel Software at Exascale

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

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

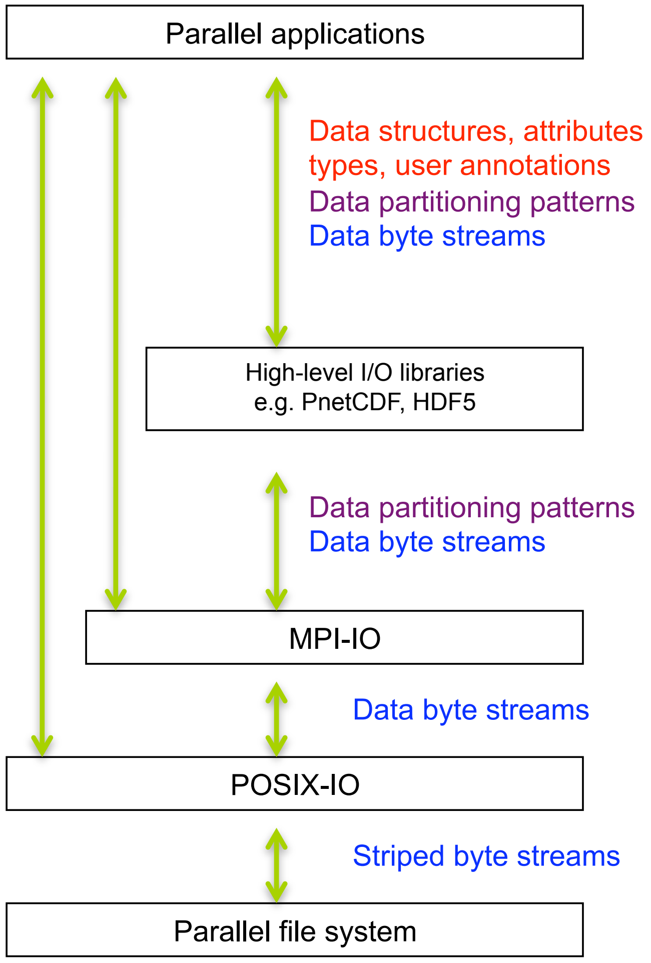
Inter-node topics

- 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 (choudahry, ishkawa)

- 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 $\longrightarrow$
- 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)

- 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 (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)

Virtualization, Dynamic Provisioning, Partitioning (mccaabe)

- 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

Power and Management (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

- 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

Removing Bottleneck to Strong (&Weak) Scaling (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

Resiliency (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)