Activities in the ASC WG on Tools

Martin Schulz, LLNL (lead)
Atinuke Arowojolu, DOE
Sean Blanchard, LANL
James Brandt, SNLs
Scott Futral, LLNL
John Mellor-Crummey, Rice University
Barton Miller, University of Wisconsin
David Montoya, LANL
Mahesh Rajan, SNLs
Kenneth Roche, PNNL
Allan Snavely, UCSD/SDSC
Mary Zosel, LLNL
History: ASC exascale planning efforts

- NNSA formed five working groups in June 2010
  - Architectures, System SW, Programming Models & Tools, Viz/Data, I/O
  - Planning meeting in Washington DC
  - Exercised showed that tools and PM needed to be separate groups
- Subsequent planning meeting in September 2010 in Albuquerque
  - Added working groups for application side
  - Later refined to: applications & Solvers/Libraries, total of eight WGs
  - Outbriefs on challenges and gaps
- Working group leads represented NNSA in March 2011 ASCR meeting
- ASC exascale meeting in San Francisco, March 2011
  - Added members from ASCR labs and academia chosen by ASCR
  - Joint working group discussions on cross-cutting issues
  - Started with September outbriefs
  - Outbriefs for each working group with ASCR input
    Recommendations for next steps/PathForward investments
Scope of the Tools WG

• Major Software Stack Elements the Group is Responsible for:
  – Tools for application development (debugging, correctness, performance)
    • Wide spectrum: memory, power, locality, resilience, …
    • Static analysis tools for code evaluation
  – Tools for SSW to evaluate the exascale stack itself
    • SSW, I/O, Network, File systems, Scheduler, …
    • Need to get away from ad-hoc tools, need whole system solution
  – Shared infrastructure for measurement, data gathering and presentation
    • Online analysis, data aggregation, shared across the system stack
    • Post-mortem, online, in site and batch tools
  – HW and SW APIs / information exchange with other WGs
    • APIs that we want to wrap and monitor
    • Introspection APIs (HW and SW)
    • Guidance for other system components (targeted, information isolation)
    • APIs exposing semantic information from the users to tools
  – Resources for testing/validation of the system (incl. tools)

• Not in scope: compilers (vendors!), resiliency techniques, runtimes
• Some successful tools all the way to Petascale class machines
  – Many successes with brute force scaling
  – Still evolving and often brittle
  – Mostly focused on single paradigm codes

• BUT: traditional paradigms are starting to break down
  – Applications are turning towards hybrid models
  – Traditional debuggers don’t scale
  – Performance analysis has to deal with flood of data
  – Full tracing at Petascale is not feasible anymore
  – Fragmented runtime systems and environments

• New approaches most include the following principles
  – Data reduction and on-line analysis
  – Flexible infrastructures for prototype tools
  – Integration and sharing across topic areas and WGs
  – Integrated runtimes avoiding stove pipes
Exascale Challenges for/around Tools

• Challenges in providing new capabilities
  – Scalability of measurement, analysis, and presentation
    • Incl. new metrics: memory, power, …
  – Turning information into insight
    • Despite flood and complexity of data from billions of threads
  – Dealing with new programming methodologies
    • Heterogeneous systems/architectures (HW and SW)
    • Coupled systems and applications
  – “What if” tools for Co-Design

• Challenges for tool implementations
  – Quick design of prototype tools for new scenarios
    • Agile development to keep up with PMs
    • Need them early, enable specialized tools in this and other areas
  – Getting right interfaces with the right abstractions
    • To SSW, HWA, Apps, Libraries, Runtimes, Compilers, …
  – Resiliency for tools and tool infrastructures
Technical Goals to Provide Efficient Tools

Gaps that other groups look for the Tools WG to fill:

• Understand and evaluate node level resources
  – Memory and threading
  – Global understanding of node local data
  – Scalable analysis algorithm (on-line/in-situ)

• Support new high-level abstractions in new PM approaches
  – Understand the performance impact of their abstractions
  – Match performance <-> PM abstractions
  – Code refactoring/translation support

• Ability to correlate HW, SW, System, App Events/Data
  – Understand/distinguish impact of system events
  – Errors/faults incl. silent errors
  – Map it to common domains

• Root cause analysis for performance and correctness
  – Construct and understand dependency chains
  – Track data flowing through the system
Technical Gaps for Building Tools

Gaps that need to be filled to provide the requested tools:

• Access to the necessary data from across the system
  – Standardized interfaces to HWA & SSW & PMs
  – New hardware features to get more data on memory
  – Low overhead is essential

• Scalable data collection and processing
  – Online and/or in-situ analysis
  – Requirements for scripting languages (?)

• Management and allocation of extra resources
  – Application launching
  – Launching and controlling tool/support daemons
  – Hide system differences

• Common service daemon architecture that is shared and reused
  – Tool component frameworks
Tool Needs: Modular Infrastructure

• Common infrastructure across WGs
  – Distributed/Cross-node architecture
  – Gather/Aggregate data
  – Online/In-situ analysis
  – Wiring up infrastructure
  – Easy to deploy and maintain
  – Easily reusable modules

• Use cases for tools (+related issues)
  – Performance information
  – Process/Debugging state
  – Status/Health monitoring
  – Dynamic resource management
  – Fault detection and mitigation
  – Online steering
Key Dependencies with Other 7 WGs (1)

- **HWA**
  - Measures of resource consumption: power, network, memory bandwidth, issue slots, …
  - Raw measures of inefficiency (exposed latency, lack of memory parallelism)
  - Identification of resources (e.g., for heterogeneous nodes, GPU versions)
  - Hardware instrumentation to emulate 2018 machine costs with 2015 machine

- **SSW**
  - Right APIs incl. RAS and debugger interfaces (incl. testing)
  - Expose all hardware features, don’t hide anything incl. counters, power, resiliency, faults, HW topology
  - Timely reporting and precise attribution of asynchronous events
  - Interfaces to scheduler, scheduling of tool resources
  - SSW runtime monitoring, runtime must expose right abstractions

- **I/O & I/O Networks**
  - For tools: interfaces to capture and measure performance (MPI_T like)
  - Capture network and storage topologies
  - Tool needs: load balancing and striping, detect link contention
  - Modeling vs. measurement to find bottlenecks
  - Tracing data movements and separate between system and user traffic
  - Provide building blocks to enable specialized I/O tools (generic tracers/profilers)
  - More discussion needed: storage approaches and formats for tools (SQL DBs?)
Key Dependencies with Other 7 WG (2)

- **Visualization and Data Analysis (VDA)**
  - Common needs, requirements on SSW (online analysis and data storage)
  - Exploit application knowledge available in Viz tools (data layout, …)
  - Provide building blocks to enable specialized VDA tools (e.g., in situ analysis)
  - Need VDA techniques for performance data analytics and visualization
    (outlier detection, equivalence groups, compression/data reduction, feature detection, …)

- **Programming Models (PMs)**
  - Compiler and runtime must provide information for tools to map costs back to PM abstractions
  - Translators/PMs/Compilers must expose abstractions to tools
  - PM runtime monitoring, runtimes must expose right abstractions

- **Applications, Solvers, Algorithms, Libraries (Apps, SAL)**
  - We are treating libraries as apps (exception: potential API interception)
  - List of expectations on tools – information that Apps/SAL people want to see
    - Data centric profiling – away from flop centric tools to memory centric tools
    - Memory locality and consumption
    - Data structures and access patterns
    - Opportunity analysis (concurrency, offload to accelerators, compiler feedback)
    - Delivering information on power and resiliency
  - Mini-Apps for testing of tools (for performance, complexity, SSW, …)
  - Application internal monitoring interfaces to capture semantic and performance data
Suggested PathForward Projects

• Memory Tools
  – New generation of tools to explore memory related metrics

• Tool Building Blocks / Infrastructure
  – Modular and Separable Tool Components

• Application-Tool Interfaces
  – Interfaces to exchange performance and semantic information

• Mini/Skeleton Applications
  – Aid in the definition of the collection of Mini-Apps

• Power Tools
  – Inclusion of power metrics into application oriented tools

• Correctness Tools
  – Verification of correct usage of PM abstractions

• Support for New Models
  – Investigation of support for new programming models
Big Picture Issues

• Coordination – must be a continuous, agile process
  – Among tool developers
    • Coordinate on common interfaces and components
    • Maintenance models
  – With Apps/SAL teams
    • Ensure their needs are met
    • Establish interfaces
  – With SSW, I/O, VDA
    • Share infrastructures
    • Avoid ad-hoc tools
  – With vendors
    • Need interfaces and documentation
    • Co-Design interactions on getting the right system hooks

• Test beds
  – Essential, need sufficient access for tools research
  – Work around security concerns (e.g., for power sensors)