EXASCALE SOFTWARE CENTER★

TOOLS PERSPECTIVE

John Mellor-Crummey

Tools Planning Team:

Jeffrey S. Vetter (ORNL)
Bronis R. de Supinski (LLNL)
Nathan Tallent (Rice → PNNL)
Allen Malony (UO)
Sameer Shende (UO)

★RIP

Exascale Tools Scope

- Performance
  - Empirical
    - observation using both instrumentation and sampling
    - offline analysis including problem diagnosis and multi-experiment analysis
    - introspection for autotuning: support for optimization guided by online analysis
  - Analytical modeling
- Correctness and debugging
  - Online
    - execution control: stop, go, conditional breakpoints, data watchpoints
    - state introspection: registers, memory, language & communication runtimes
    - analysis: semantic, relative, behavioral equivalence classes, statistical
  - Formal methods: static analysis, model checking
- Presentation and insight
  - Render correctness and performance results in a scalable, actionable form
- Tools infrastructure and middleware

Exascale Landscape: Hostile for Tools

Tools face the same issues as applications AND must consider application evolution as well

- Extreme, multi-level, heterogeneous parallelism
  - Process, thread, instruction, vector, accelerator
- Dynamic execution environment
  - Dynamic threading
  - Adaptive HW and SW systems
  - Hardware failure
- Massive asynchrony
  - Computation, communication, I/O

<u>Application Evolution</u>

- New programming models
- Growing application complexity
  - Multi-faceted applications
  - Simulations coupled with in-situ data analysis

Exascale Tool Challenges – I

- Exascale tools: event-based reactive systems of immense scale
  - Must interact with all HW and SW components
    - often at an extremely detailed level
  - Dubbing this an "engineering challenge" is an understatement
- Observation of exascale execution dynamics and system state
  - Tools require access to not only application state but runtime as well
    - requires co-design with programming models, runtime systems, and OS
  - On-the-fly, problem-focused measurement, analysis, and data reduction
  - As always, tradeoff between precision and accuracy
- Fault tolerance
  - Tools must be aware of application checkpoints, faults, and recovery
  - Tools themselves must tolerate faults

Exascale Tool Challenges – II

- Data management
  - Different data organizations are appropriate for measurement, analysis and presentation
    - thread-centric, resource-centric, time-centric, code-centric, data-centric, …
  - Large data volumes will require careful design of persistent representations, e.g. dense vs. sparse; consider access patterns
  - All analysis, data transformations, and I/O must be parallel
- Analysis and modeling
  - Cope with the complexity, dynamism, and heterogeneity of exascale executions
  - Data deluge requires automation of problem identification and diagnosis
- Presentation
  - User interfaces must direct attention, not just provide access to information
  - Automatically scale and focus presentation to render phenomena (automatically determined) of interest

Key Tool Design Questions - I

- What hardware mechanisms are needed to support effective tools?
  - Support for measurement, attribution, and diagnosis of problems
    - both for correctness and performance
- What language and compiler support is necessary to provide information required by tools?
- What runtime and OS mechanisms and interfaces are necessary to support inquiry and control by tools?
- How will tools efficiently and effectively monitor massively parallel programs executing on heterogeneous hardware with multi-scale, hierarchical parallelism in which faults may occur?
- What measurements and analyses are necessary to diagnose root causes of performance bottlenecks and to recommend solutions?
  - Load imbalance, serialization, resource contention, exposed latency, ...?
- How will tools interact with dynamic and fault-tolerant run times?

Key Tool Design Questions - II

- How will tools analyze and mine GB/TB of data and attribute information to source code in a meaningful way?
- What new tool paradigms can overcome lack of insight from existing tools?
  - Today: data summary
  - Need: automated problem discovery, diagnosis, and recommendations

Evolution of Current Capabilities?

- Performance
  - Sampling-based methods for measurement and analysis can scale
  - Instrumentation for capturing semantic information is necessary
  - Code-centric, data-centric, time-centric presentation paradigms useful
  - Promising recent developments
    - emerging integration with parallel programming environments
    - emerging measurement infrastructure for accelerator cores
- Debugging / Correctness
  - Vendor tools are marginally usable at current system scales
  - Correctness tools for identifying runtime communication errors do not scale
  - Model-checking and other formal methods limited in scalability, robustness for mainstream languages and range of programming models
- Tool infrastructure
  - Scalable middleware demonstrated and under continued development
  - Increasing HW support for performance monitoring, watchpoints, ...

New Capabilities Needed

- On-line: measurement, monitoring, control, data reduction
  - Scalable problem-focused measurement to support effective diagnosis
    - resource consumption, inefficiency, power consumption
  - Techniques for extreme parallelism, dynamic threading, and heterogeneity
  - APIs to support introspection & lightweight analysis for adaptation
    - performance, fault-tolerance
  - Programmable thread-based agents for correctness introspection
  - Framework for survivable tools
- Analysis
  - Diagnosing bottlenecks with massive dynamic threading
  - Integration of semantic correctness checking in new programming models
  - Data mining for diagnosis of performance/correctness
- Modeling and prediction for diagnosis
- Scalable presentation
  - Visualize application data, system state, activity & their evolution over time
- Paradigms that drastically reduce the optimization and debugging effort

Essential Component Technologies

- HW, OS support for measurement, especially sampling
- Programmable thread-based agents for scalable online analysis of data and execution state
- Automatic identification of interesting phenomena within data
- Tool infrastructure API for applications to control, inform, & inquire
- Stateless protocols for fault-tolerant interactions between tool components
- Idioms for scalable presentation
- Binary analysis to support modeling and instrumentation
- Binary and wrapper instrumentation (measurement, correctness, control)
- Stack unwinding for attributing costs
- Tool middleware and use of system/library support (e.g., Parallel I/O)

Hardware Co-Design Opportunities

Insight from tools will be limited without HW support

- Need HW measurement interfaces to monitor & attribute
  - Communication, computation, power, data movement, latency, I/O, …
- HW should support both calipers and sampling
- Need efficient access to application state
  - Program counters, thread stacks, …
  - Data state: memory watchpoints, association of memory events with program counters, etc.
- Design of new HW technologies must consider tool support required to understand correctness and efficiency
- HW tool assists to improve tool efficiency?

A Few Words About GPUs

- **NVIDIA Profiling Roadmap (what they are thinking about)**
- Measurement
  - Finer granularity profiling: node → kernel → instruction (pipeline, memory subsystem)
  - Existing hardware limitations are not fundamental, e.g. the following are possible:
    - PC and event-based sampling
    - increased type, size, and number of hardware counters
    - tracing (though time and space overhead can be high)
  - Power, power state profiling
  - Increased profiling scope
    - remote profiling, e.g. node in a cluster
    - multi-process profiling
- Attribution of performance problems and opportunities at source level
- Analysis
  - Automatic identification of common, actionable performance problems
    - e.g. loads with poor memory subsystem behavior
  - Tools to identify algorithms, functions, loops, etc. that are good candidates for GPU acceleration
- Profiling ecosystem enablement

David Goodwin, NVIDIA Tools Group

Software Co-Design Opportunities

- Exascale tool development must interact with programming models, compilers, runtime/system software
  - Exascale machine models will be basis for tool design, validation, and use
  - Identify necessary runtime and OS support for enabling tools
  - Need tool capabilities to meet needs at all levels of software stack
  - Identify points of tool interaction for providing feedback and controlling tuning knobs
- Exascale software advances could be leveraged in tool development
  - e.g., data management, visualization

OS and Runtime Co-Design

Usable tools require OS and runtime systems to provide:

- Interfaces for intercepting and modifying operations on key abstractions
  - Threads, processes, locks, memory allocation, files, communication channels
  - Ability to run tool processes/threads of control
  - Exporting of hardware-measurement interfaces
  - Scalable access to executables and shared objects for online analysis
  - Accurate and complete unwind and line map information
  - Program timer and PMU threshold-based interrupts in repeat mode
- Support for thread-specific asynchronous signals
- Summarization of (thread-specific) signals during system calls vs. system-wide sampling
- Interface for mapping machine-level to application-level state
  - e.g., recovering application call paths in the context of work stealing

Closing Thoughts

- Exascale tools challenges reflect full range of complexity found in exascale software/hardware
- A prescription for tools development
  - Interact with all exascale software groups
    - establishes requirements and decision metrics
    - co-design with programming models, OS/runtime, I/O, Viz
  - Identify and select critical technology
    - select tool capabilities to enhance and extend
    - identify necessary new capabilities for investment
  - Research and development/engineering effort focus
    - refining and scaling appropriate existing technologies
    - developing new technologies to address new exascale concerns

Strawman Plan

Performance Tools Strawman Plan - E

- Performance data sources
  - Hardware counters for monitoring and attributing time, power, processor core/uncore activity/idleness, network traffic, data movement, synchronization
- Performance measurement underpinnings
  - Kernel interfaces for programming & accessing HW counters
  - Kernel support for delivering and handling signals for asynchronous events
  - Instrumentation hooks for key interfaces in libraries, OS in full software stack
    - e.g. MPIT, one-sided communication, synchronization, I/O, …
  - Compiler-based instrumentation (e.g. function entry/exit)
  - Binary instrumentation
  - Link-time and/or launch-time library wrapping tools
    - e.g. hpclink, Launchmon, P^nMPI, hpcrun
  - Source instrumentation?

Performance Tools Strawman Plan - E

- Performance measurement software
  - High-level interface for programming performance counters
    - e.g., PAPI, CUPTI
  - Introspection API to be used by tools and autotuners
    - e.g., PAPI
  - Attribute metrics to calling context based on synchronous and asynchronous events using call stack unwinding
    - Components: accurate & complete compiler line maps (vendor buy-in)
    - Run-time library support for unwinding continuations
  - Instrumentation for (problem-focused) measurement and (sometimes) online analysis, and (sometimes) attribution for key library interfaces such as communication, I/O, synchronization etc.
  - Profiling infrastructure

Performance Tools Strawman Plan - E

- Data management
  - Logging measurement data to files using parallel I/O
    - e.g., SIONLIB, MPI/IO, custom data formats
  - Scalable multi-experiment parallel profile database, e.g. TAU
- Analysis
  - Parallel data analysis
  - Online-analysis to support introspection, e.g. performance assertions
  - Pinpointing scalability bottlenecks; differential profiling, e.g. HPCToolkit
  - Identifying rate-limiting resources for code regions, e.g. Roofline
  - Binary analysis for attribution
  - Data-centric diagnosis, e.g. HPCToolkit
  - Parallel data mining, regression analysis
  - Heterogeneous performance analysis
- Presentation
  - Code-centric, data centric, and time-centric performance metrics

Performance Tools Strawman Plan - R

- Performance data sources
  - HW counter support for sampling accelerator performance
- Performance measurement software
  - Measurement approaches for workflows, e.g. coupled codes
  - Support for tool fault tolerance
  - Integrated performance monitoring with feedback support
- Analysis
  - Data mining for automatic bottleneck detection and diagnosis
    - scalable diagnosis of temporal workflow bottlenecks: provisioning, critical path
    - diagnosing node throughput bottlenecks
    - assessing application fault tolerance
  - Analytical and empirical modeling
- Presentation
  - Automatically identify, autoscale & present relevant data
  - Multidimensional or temporal data

Correctness Tools Strawman Plan - E

- Correctness data sources
  - Hardware for breakpoints, watchpoints
- Correctness monitoring & control underpinnings
  - Kernel interfaces process control, e.g. ptrace, Topaz teledebugging
- Correctness measurement software
  - Binary instrumentation for monitoring accesses & computation
    - e.g. valgrind, Dyninst
  - Instrumentation library for checking communication, e.g. MARMOT, MUST
- Online analysis & control
  - Data access errors, e.g. valgrind
  - Online data analysis, e.g. relative debugging
  - Online data reduction and control, e.g. MRNet, STAT
  - Scalable breakpoint debugging
- Presentation
  - Code-centric presentation of correctness metrics, e.g. HPCToolkit, STAT

Correctness Tools Strawman Plan - R

- Correctness measurement software
  - High performance race detection
- Online analysis & control
  - Better techniques for command, control, and feedback at scale for debugging
- Online presentation of data
- Offline analysis
  - Statistical techniques; cooperative bug isolation
  - Static analysis for proving correctness; e.g. MPI checkers