Programming Environments

• Identify elements of the programming environment (e.g., debuggers, performance analysis, storage + viz, ...) and how are they integrated
• Discuss issues of **performance portability** across systems of different type, scales, and generations
• Discuss the role of **performance models and productivity models** to motivate approaches, challenges and opportunities
• Discuss **debugging strategies for correctness and performance**
• Understand the needs for **storage abstractions** in future programming models and environments
• Discuss **interoperability and composability** issues
• Determine requirements and interconnections of programming environments with the **rest of the system stack**
Correctness Debugging Today

- Debugging Tools
  - Printf is still a popular tool
  - Applications have consistency checks
  - Totalview, Alinea
  - Debuggers often come late; well after a system is in place
  - Race conditions: not so common
  - Numerical errors: very common
  - Checkpoint restart essential
  - Visualization can be important in debugging large data sets

- Debugging at Scale
  - Track a representative process
  - GDB with 1K states that are merged
Correctness Debugging Future

• In addition to those current tools future programming will require:
  • Better support for race conditions
  • Task graph visualization
• Reproducibility tools
  • Ability to get bit-wise reproducible results on same machine/size/input
  • Currently for reductions, also for “atomic updates”, etc.
• Interactions with the system errors
  • Ability to detect and analyze
  • Closely related to resilience tools
• Performance tools used more than correctness tools
• Especially for perf portability
• Valgrind, Vampire, etc. used
• TAU, HPCToolkit for parallel analysis
• Tools for performance experts vs. performance novices?
• Performance regression testing in systems software (apps too?)

General Requirements:
• Composability across components
• Ability to measure success metrics
Performance Debugging

Future (from PModel)
- Need to reverse engineer runtime
- More automatic decisions $\rightarrow$ more tools to understanding
- Debugging optimized code hard (DSL compilers)
- Communication leaks in PGAS
- Task graph visualizer with time
- Future (system interaction)

Interactions with the system, E.g., Power and energy debugging:
- Will system allocations be based on energy? At least get energy statement
- Control done by hardware (throttling); measure / interpret dynamic behavior
- If programmer manages energy, tools essential
Reproducibility, Resilience, QOS

**Reproducibility**
- At some cost to be able to get this at runtime, even with significant slowdown
- Markers in the program that must be run in a reproducible way;
- Force order to validate code
- Flight data recorder to rerun a particular execution
- Unit testing: can we record data in/out to rerun
- Idea: Same order and therefore same results (ordered collectives used for validation); need to separate instability from bad hw/sw

**Resilience infrastructure:**
- Need to be able to simulate faults
- Monitoring state of systems (heartbeats)

**QOS metrics**
- Willing to take longer for lower power
- Willing to run with less reliable hardware (and save power)
Performance portability

Several approaches to performance portability

Categories: encapsulation, translation, autotuning

- Partitioning of the application code between:
  - Application + Math
  - Computer science
- Create user-friendly abstractions (Kokkos and RAJA)
- Separate algorithms from schedules
  - Need refactoring tools
- Highly optimized Libraries
  - DSLs (EDSLs) with translators
  - DS Libraries
  - Autotuners
  - Algorithmic choice
  - Compilers that understand autotuners (generate many versions)

Does difference in NNSA and SC Application model affect approach?

Broad vs. focused impact of program?
Approaches to Autotuning (Personal comments)

**Autotuners are code generators plus search**

- Avoids two unsolved compiler problems: dependence analysis and accurate performance models
- Popular in libraries: Atlas, FFTW,…
- Some things don’t “libraryize”
  - Sparse / irregular harder (OSKI)
  - Parameterized of functions (stencils)

**How do we produce all of these (correct) versions?**

- Using scripts (Python, perl, ML, C,..)
- Compiling a domain-specific language (D-TEC)
- Compiling a general-purpose language (X-Tune)
- Dynamic compilation of a domain-specific (SEJITS)
Performance Models

- **Three related types of performance models**
  - Benchmarks to drive compilers and tools
  - Simulation tools
  - Analytical models

- **Use analytical models to**
  - Understand performance limits and power
  - Predict benefits/pitfalls;
    (with some skepticism expressed)

- **Hardware trends harder to see**
  - Understand performance cliffs;
  - Generalized roofline

- **Understand failures in future practice**
  - Collective database of failure modes and predictability (mining of logs)
  - Are uniform distribution of failures (probably not)
  - Failure vs. fault rate; domain of failure
Productivity Models

• Too important to not measure
  • We should measure and model it
  • High performance but not dealing with complexity of system
• Don’t want to punt because it is too hard
• Is adoption rate the best measure?
• We want a metric to show why the first exaflop machine is not necessarily productive exascale systems
  • How much science is done?
  • Utilization of the investment
  • Number of programs that run on the technology (use the model)
  • Trickle down to petascale in a rack (vs. the cloud, in the cloud?)
Storage Abstractions

• Understand the needs for storage abstractions in future programming models and environments
  • Transient storage for workflows important, e.g., NVRAM for life of workflow
  • There is a storage systems and I/O report to be posted by March 30; gives needs for programming environments
• Expand value of systems
  • Accommodate applications from different environments
• Compiler and programmer need to given information to stage data movement throughout the 10 levels of memory hierarchy
Requirements and Interconnections with the Rest of the System Stack

- Traceability of the mapping
- Understand performance behavior with behavior of system; communication infrastructure can affect
- Need to be hooks to instrument /debug the entire system
- Need a meta-interface to
  - Pass information up to user: understand the semantic loss in the implementations
  - Pass information down to runtime
- Learn from successive execution of the program, incremental database of the program execution; or across ssystem
Productivity Models

• Is there value to productivity models?
• We don’t know how to measure it
• Some companies do measure productivity of programmers
• The community is too small and the work and programmers are too heterogeneous
• Social factors and administrative constraints
• We (NNSA labs) are doing comparisons for roadmaps
  • Productivity is single biggest factor
  • It would be useful to agree on a metric
  • Cost includes both development and maintenance
• One way to get at this is for different code teams to report on:
  • Are they using DSLs? Autotuners?
  • Collecting data from the user community; even libraries / frameworks
Within the HPC field, where we value performance
- More effort produces better performance (up to a point)
- We invest heavily to achieve that
- Productivity represent a relationships between achievable performance and what is observed
- A value of productivity is that it can guide us to improve performance and reduce time for develop codes
- Adoption of programming models on smaller petascale systems