System Software Working Group

Frontiers of Extreme Computing

Ron Brightwell (c), Jeffrey Vetter (vc), Almadena Chtchelkanova, Guang Gao, Patrick Geoffrey, Mary Hall, Fred Johnson, Jim Kasdorf, Ron Minnich, Jose Munoz, David Probst, Neil Pundit, Steve Scott, Thomas Sterling, Kathy Yelick

Issues

Strategic Issues – System Software and Programming Environments WG

- Advanced execution models
- Parallel programming models, methods, and tools
- Resource management, allocation, and scheduling
- Mass storage and I/O management

9

CENTER FOR COMPUTATION & TECHNOLOGY AT LOUISIANA STATE UNIVERSITY

Questions

Questions – System Software and Programming Environments WG

- What new semantic constructs and execution models will be needed by Exascale applications and algorithms? What will the programming languages of the future look like?
- How will programmers contend with billion-way parallelism? What will the new languages look like?
- What operating system organizing strategy will effectively manage 100 million or more processing cores efficiently and reliably?
- How will compilers and runtime systems support the new classes of applications dominated by dynamic meta data structures?
- What will be the balance in the future between user direct control of resources and system automation for ease of use?
- Will programmers continue to program with arithmetic statements, or may a different paradigm become prevalent? Examples, neural networks, fuzzy logic, graph algorithms, image processing, real time?

Multiple Perspectives

- **Strategic Issues**
  - Advanced execution models
  - Parallel programming models, methods, tools
  - Resource management, allocation, scheduling
  - Mass storage and IO
- **Software Hierarchy**
  - Operating system
  - Runtime
  - Programming models, Languages, Compilers
  - IO, Storage, Mass storage
  - *Programming Tools*
    - *Correctness*
    - *Performance*
- **Cross-cutting Issues**
  - Power
  - Parallelism
  - Performance
  - Execution Model
  - Efficiency
  - Cost
  - Reliability
  - *Productivity*
- **Application**
- **Architecture**

Assumptions

- **Applications**
  - Use new programming models to exploit new architectural features
  - Legacy applications must work
    - Operate at performance the application designed for
- **Architecture**
  - Billions of threads
  - Globally address space
  - Multiple levels of memory with different characteristics
  - Dramatic improvements in system balance

Discussion Summary

Challenges

- Current OS Trends
- Traditional resource allocation mechanisms are at least 30 years old
- Lack of co-design
  - Need testbeds, simulators
- Support for fault-oblivious applications
- Radically new architectures
  - New ISAs, memory models
- Support for billion-way parallelism
- Little/no interaction among OS, compiler, runtime
- Programming models have not changed
- Software business models

Opportunities

- Large scale concurrency is here (100k) and growing
- Heterogeneous computing is quickly approaching
- Move beyond legacy application support
- Device technologies may inject dramatic improvements in architecture balance
- Major changes in architectures are forcing reexamination of trends
- New programming models that expose architecture's performance features
- Leverage community efforts to improve parallel programming for masses

Recommendations (1)

- **New resource management strategies for processor, memory, bandwidth**
  - Current methods are insufficient
  - Scheduling
  - Flops are free; bandwidth is precious
  - ‘New’ system balances
- **New abstractions for managing heterogeneous systems (e.g., multiple types of processors, memory, memory models)**
  - Need access to new simulators, architectures
- **Hardware and system software co-design**
  - Use VMs to test OS at scale before system arrives
  - Need access to new simulators, architectures during design
  - Expose hardware features that allow improved performance, reliability
    - E.g., Transactional memory
      - need features in hardware for operation and performance analysis

Recommendations (2)

- **Expeditions into new software systems for architectures that are 10^4 larger than current systems**
  - OS
  - Programming systems
  - ‘Programs writing programs’
  - Consider system software to be an application (‘eat our own dogfood’)
  - We can start this today – no need to wait!
- **Reliability abstractions and methods**
  - Fault-oblivious applications, programming models, ...
  - Checkpointing your exascale application will be impractical
- **IO, storage**
  - Implicit IO – need new methods for managing the n-level storage hierarchy
- **Self-evident**
  - More funding, testbeds, programs
- **Education**

Bonus Slides

Operating Systems

- **Efficiency, Performance**
- **Reliability,**
- **System software resource overheads in terms of memory, time, power, heat, dollars overwhelms applications software**
  – can’t afford resources to make it possible

Programming Models / Languages

- **Synergy between os, runtime, and compile time**
- **Appropriate abstractions for levels of software stacks**

Runtime

IO, Storage, Mass Storage

Execution models

- **Abstract framework encompassing architecture, programming model, runtime/os, a set of governing principles interrelating them**

Cross-Cutting Issues

- Power
- Parallelism
- Performance
- Execution Model
- Efficiency
- Cost
- Reliability

Power

- **Needs to be managed as a precious resource (anything that consumes power)**
- **System software (OS, compiler, …) needs to expose and manage power resources to applications**
  - Cache management, bandwidth management, etc.
- **How to model power**
- **Get processor developers to provide simple mechanisms**
- **Minimizing data movement helps**

Parallelism

- Billion-way parallelism is a (the?) significant challenge
- How to express parallelism (TLP, PGAS, Data Parallel, etc.)
- OS should support arbitrary resource management policies
- System software for large-scale heterogeneous processing systems
- Ease of programming

Performance

- Reduce overhead so it is no longer a lower bound on granularity
- Don’t slow the apps down
- Abstraction for exposing architectural performance features
- Reduce operating system call overhead to the level of a procedure call
- Current approach for performance tools will not scale

Execution Models

- **Must exploit synergy between OS, run-time and compile-time**
- **Linking the memory model with the execution model**
- **Lack of I/O (streaming, secondary storage) support inherent in the execution model**
- **How to get the "right" protocol interaction between the compiler, run-time, and OS**
- **Appropriate abstractions**
  - For the machine and for the app developer
- **Support for numerous, various architectures and applications**

Efficiency

- **Need new approaches to resource allocation and scheduling that do not degrade efficiency and predictability**
- **Shared address space machine memory model**
  - Cache-coherency (has not) and will not scale
  - An API for exposing such parallelism
  - Pre-fetchers will not scale
- **Continuous load balancing (adaptivity)**
- **Devise a programming language for a trans-exaflops machine**
  - Language support for avoiding evil data races
  - Balance responsibilities between user and system
  - Marxist distribution of responsibilities (who's good at what)

Productivity

- **Support for legacy applications – new machines have to produce useful results early**
- **Support for application portability**
- **Dealing with application developers' inertia**
- **Leveraging smart applications' people**
  - Can't please all of the people all of the time
  - Need to work with app developers to "do the right thing"
- **Provide a sane environment for application development**
- **Need new program development environments**
  - Debuggers for billions of threads

Cost

- **Lack of appropriate OS testbed resources**
- **Where does the money come from for system software development**
  – Software may no longer be free
- **Cost from uniqueness of systems?**
  – Radically different system software from machine to machine

Reliability

- System software and tools need to provide an environment for the development of more robust, failure-tolerant applications
- Managing resources in the presence of failures at scale – dynamic reconfiguration
- Fault oblivious programming model
- Tools to insure system software correctness
- Invariant violation application debuggers

Design and Implementation

- **Current trends in OS development are not addressing fundamental issues required for trans-exaflops computing**
- **Current OS's are not structured to enable trans-exaflops computing**
- **Expectations of "develop on the desktop and run efficiently on the exaflop" need to be managed**
- **System software people are not getting it right either (automake, configure are part of the problem)**
- **The customer may not always be right**
- **System software verification on large-scale systems**
  - Need real applications, real problems, and lots of time
- **Leveraging disruptive technology smoothly**
- **Non-fixed OS (composabililty)**
- **What should be virtualized?**

Challenges

- **OS trends are not helping**
- **Expectation of Linux desktop environment everywhere**
- **Just say "no"**
  - To non-scalable and/or non-predictable things
- **Our apps are not Google apps**
  - More strict requirements
  - But could they be more robust to failure(s)?
- **There's a right way – just do it**
  - Conflicts with the business model

Challenges

- **Runtime tightly coupled with compiler?**
  - Opportunity to explore more dynamic behavior
  - Execution model has to allow asynchronous threads
  - Predictable performance

Challenges

- **Sheer scale – number of things to manage**
  - Billion-way parallelism
  - Reduce overhead so it is no longer a lower bound on granularity
- **Current OS's are not structured to enable trans exaflops**
- **Synergy between OS, run-time and compile-time**
- **How to express parallelism (TLP, PGAS, Data Parallel, etc.) and the corresponding execution model**
- **Linking the memory model with the execution model**
- **Managing resources in the presence of failures at scale**
- **Multiple definitions of an "execution model"**
- **Shared address space machine memory model**
  - Cache-coherency (has not) and will not scale
  - An API for exposing such parallelism
  - Pre-fetchers will not scale
- **Strict scaling from teraflops to trans exaflops**
- **Support for legacy applications**
- **New approaches to resource allocation and scheduling that do not degrade efficiency and predictability**

Challenges

- **Support for application portability**
- **Dealing with application developers' inertia**
- **Lack of I/O (streaming, secondary storage) support inherent in the execution model**
- **Leveraging smart applications' people**
  - Can't please all of the people all of the time
  - Need to work with app developers to "do the right thing"
- **Lack of appropriate OS testbed resources**
- **OS should support arbitrary resource management policies**
- **Getting the OS out of the way**
- **Provide a sane environment for application development**
- **Fault oblivious programming model**
- **Need new program development environments**
  - Debuggers for billions of threads
- **System software verification on large-scale systems**
  - Need real applications, real problems, and lots of time

Challenges

- **How to get the “right” protocol interaction between the compiler, run-time, and OS**
- **Appropriate abstractions**
  - For the machine and for the app developer
- **Leveraging disruptive technology smoothly**
- **Continuous load balancing (adaptivity)**
- **Support for numerous, various architectures and applications**
- **Non-fixed OS (composabililty)**
- **System software for large-scale heterogeneous processing systems**
- **Devise a programming language for a trans-exaflops machine**
  - Language support for avoiding evil data races
  - Balance responsibilities between user and system
  - Marxist distribution of responsibilities (who’s good at what)

Opportunities

- **Decent programming models**
  - Expressiveness, generality, performance, productivity
- **Influence architectures – co-design**
  - E.g., FEB on network messages
- **Initiate OS expeditions to explore these new design spaces**
- **Neil – look at the largest systems we have and highlight successes**
- **Moving beyond legacy applications**
- **Give applications developers tools to manage parallelism and locality easily**
- **Develop massive parallel asynchronous fine grained execution model**
  - Tnt is not asynchronous
- **Reassigning responsibility throughout the software stack**

Opportunities

- **Funding for system software NRE**
- **Resilient computing – computation that continues to completion in spite of failures – fault oblivious computing**
- **User challenge → parallelism is not easy**
- **Leverage hardware developments in multicore and many-core**
  - HPC's problems are now the world's problems ☺
- **Photonics should ease traditional burdens on software**
- **Help apps developers manage locality**
  - Tools for doing so
- **Allow experts to manage software system explicitly and inject domain knowledge into the system**
  - Conservative defaults, heroic overrides
- **Feedback-driven or adaptive compilers**
  - Redefine role of compiler
  - Heterogeneous systems
  - Autotuning
  - Interactive optimization
- **Beefeaters is Thomas' favorite drink**
- **Opportunity to establish a new relationship between runtime and OS; where the compiler is a conduit from the programming model to the runtime**
  - Hardware <-> Runtime Systems <->
  - Dynamic system for managing parallelism
- **Metrics for evaluating system software capabilities**
- **Revisiting system software design choices in light of light**

Viable Paths Forward

- **Education**
  - Careers in HPC
  - Labs need to emphasize intellectual freedom
  - Adequate investments
  - Beowulf boot camp
- **Market size – consequences of open source**

Viable Paths Forward

- **TS Operating System**
  - Lightweight kernels
  - Emerging behavior from LWKs
  - Development time is relatively small
  - Single machine that is self-regulating
  - Lightweight synergistic types of constructs that are symbiotic
    - Small group in short amount of time

Viable Paths Forward

- **Development environments**
  - Virtualization for experimentation
    - Scaling
- **Limitations from legacy application's constraints**
  - Accommodate legacy applications
  - Refactor for optimal performance
  - Well defined migration path
- **System software resource overheads in terms of memory, time, power, heat, dollars overpowers applications software**
- **Mass storage**
  - software for mass storage should be this new programming model
  - fault oblivious, full use of tlp
- **Tools for performance and correctness**
  - Usability at scale

Operating Systems

- **System software resource overheads in terms of memory, time, power, heat, dollars overwhelms applications software**
  - can't afford resources to make it possible
- **Demand paging will not occur**
- **Support for new architectural constructs or models**
  - Don't assume uniformity of system resources in a specific application
  - Multiple levels of memory, different characteristics
    - Need a new memory model
    - One physical port
    - vNUMA
- **Threads / Scheduling**
  - Don't want kernel level threads scheduling
  - Need predictability
  - Preemption in alternative devices
  - Performance variability across different memory hierarchies, devices
- **Can't drop a process on a raw processor w/o OS, protection domains**

Programming Languages / Compilers

- **Programs that write programs**
- **Components**
- **Correctness**
- **Expressiveness**
- **Should functional programming remerge**
  - Erlang, haskell,
- **New languages: X10, Chapel, Fortress**
- **Languages that allow application to specify multi-grained parallelism and locality**
  - Different synchronization mechanisms
- **Transparency v. visibility**
- **Mainstream and elite users**
- **Data structures, affinity, distributions**

IO and Storage

- **Storage is a parallel application**
  - Same problems: distributed data, parallelism, failures
  - Same tools for 30 years
    - C, Unix-like OS
- **Performance and scaling**
  - N-way to one scaling
  - Same load balancing problems
- **IO/storage is often under-provisioned**
- **Checkpointing (1 EB)**
  - Checkpointing needs more work
    - This isn't your grandfather's checkpointing

Runtime

- **Control flow migration in runtime**
  - Continuations
  - Latency hiding opportunities
- **Runtime will have fluctuating resource demands on system**
- **Runtime will have a shorter 'wavelength' than OS**
- **No protection, lightweight**
- **Triumph of user-level runtime**
- **Transparent support for correctness and performance analysis**