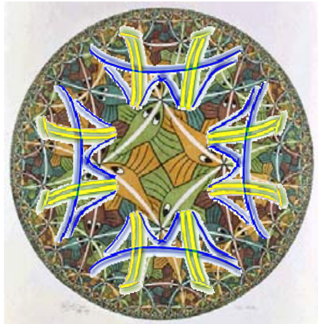
Tessellation OS



Architecting Systems Software in a ManyCore World

John Kubiatowicz
UC Berkeley
kubitron@cs.berkeley.edu

Services Support for Applications

- What systems support do we need for new ManyCore applications?
  - Should we just port parallel Linux or Windows 7 and be done with it?
  - A lot of functionality, hard to experiment with, possibly fragile, …
- Clearly, these new applications will contain:
  - Explicitly parallel components
    - However, parallelism may be "hard won" (not embarrassingly parallel)
    - Must not interfere with this parallelism
  - Direct interaction with Internet and "Cloud" services
    - Potentially extensive use of remote services
    - Serious security/data vulnerability concerns
  - Real Time requirements
    - Sophisticated multimedia interactions
    - Control of/interaction with health-related devices
  - Responsiveness Requirements
    - Provide a good interactive experience to users

PARLab OS Goals: *RAPPidS*

- Responsiveness: Meets real-time guarantees
  - Good user experience with UI expected
  - Illusion of Rapid I/O while still providing guarantees
  - Real-Time applications (speech, music, video) will be assumed
- Agility: Can deal with rapidly changing environment
  - Programs not completely assembled until runtime
  - User may request complex mix of services at moment's notice
  - Resources change rapidly (bandwidth, power, etc)
- Power-Efficiency: Efficient power-performance tradeoffs
  - Application-Specific parallel scheduling on Bare Metal partitions
  - Explicitly parallel, power-aware OS service architecture
- Persistence: User experience persists across device failures
  - Fully integrated with persistent storage infrastructures
  - Customizations not be lost on "reboot"
- Security and Correctness: Must be hard to compromise
  - Untrusted and/or buggy components handled gracefully
  - Combination of *verification* and *isolation* at many levels
  - Privacy, Integrity, Authenticity of information asserted

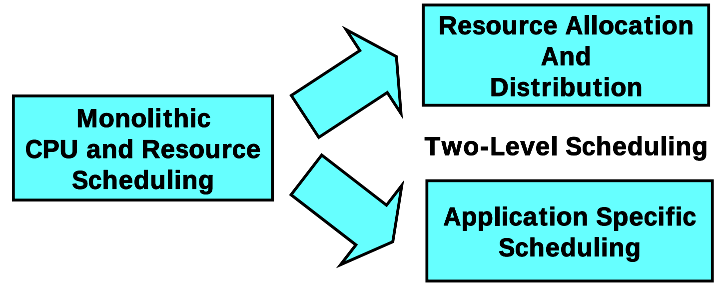
The Problem with Current OSs

- What is wrong with current Operating Systems?
  - They (often?) do not allow expression of application requirements
    - Minimal Frame Rate, Minimal Memory Bandwidth, Minimal QoS from system Services, Real Time Constraints, …
    - No clean interfaces for reflecting these requirements
  - They (often?) do not provide guarantees that applications can use
    - They do not provide performance isolation
    - Resources can be removed or decreased without permission
    - Maximum response time to events cannot be characterized
  - They (often?) do not provide fully custom scheduling
    - In a parallel programming environment, ideal scheduling can depend crucially on the programming model
  - They (often?) do not provide sufficient Security or Correctness
    - Monolithic Kernels get compromised all the time
    - Applications cannot express domains of trust within themselves without using a heavyweight process model
- The advent of ManyCore both:
  - Exacerbates the above with a greater number of shared resources
  - Provides an opportunity to change the fundamental model

Outline

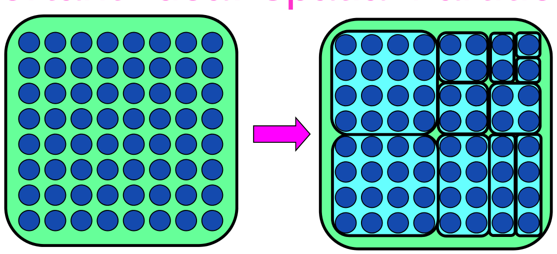
- Space-Time Partitioning
  - Two-Level Scheduling
  - Spatial Partitioning
  - Cell Model
- The Resource Management Architecture
  - Space-Time Resource Graph
  - Policy Service Architecture
  - User-Level Scheduling Support (Lithe)
- Tessellation implementation
  - Hardware Support
  - Status
  - Future Directions

A First Step: Two Level Scheduling



- Split monolithic scheduling into two pieces:
  - Course-Grained Resource Allocation and Distribution
    - Chunks of resources (CPUs, Memory Bandwidth, QoS to Services) distributed to application (system) components
    - Option to simply turn off unused resources (Important for Power)
  - Fine-Grained Application-Specific Scheduling
    - Applications are allowed to utilize their resources in any way they see fit
    - Other components of the system cannot interfere with their use of resources

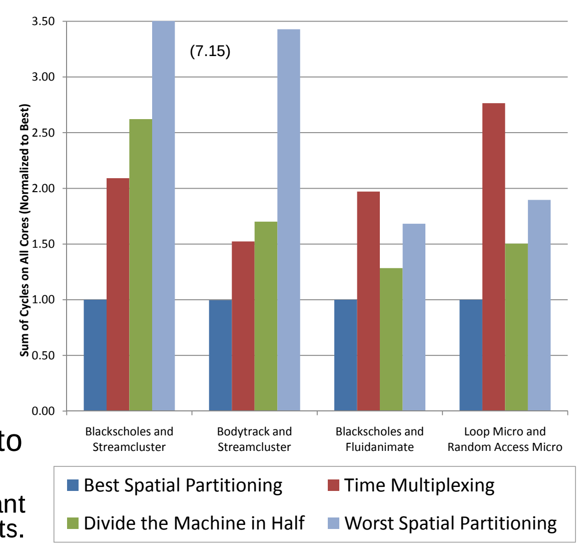
Important Idea: Spatial Partitioning



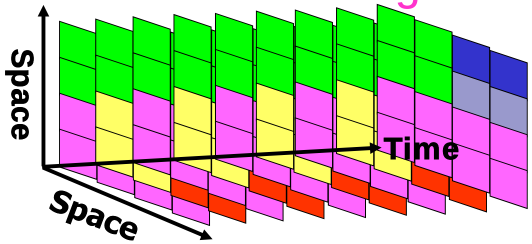
- Spatial Partition: group of processors within hardware boundary
  - Boundaries are "hard", communication between partitions controlled
  - Anything goes within partition
- Key Idea: Performance and Security Isolation
- Each Partition receives a vector of resources
  - Some number of dedicated processors
  - Some set of dedicated resources (exclusive access)
    - Complete access to certain hardware devices
    - Dedicated raw storage partition
  - Some guaranteed fraction of other resources (QoS guarantee):
    - Memory bandwidth, Network bandwidth
    - fractional services from other partitions

Performance w/ Spatial Partitioning

- RAMP Gold: FPGA-Based Emulator
  - 64 single-issue in-order cores
    - Up to 8 slices using page coloring
  - Private L1 Inst and Data Caches
  - Shared L2 Cache
    - Up to 8 slices using page coloring
  - Memory bandwidth partitionable into 3.4 GB/s units
- Spatial partitioning shows the potential to do quite well
  - However it is important to pick the right points.



Space-Time Partitioning

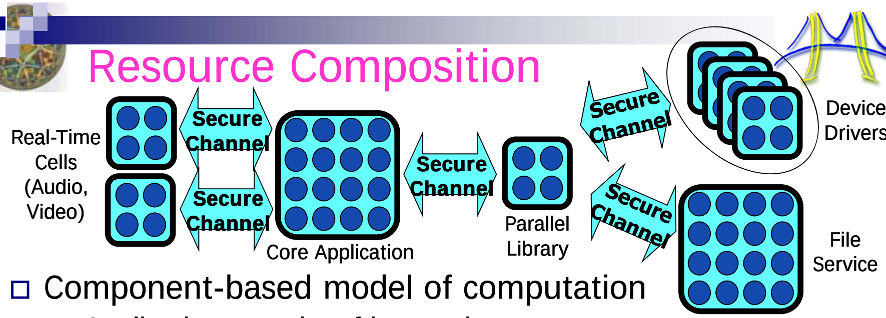


- Spatial Partitioning Varies over Time
  - Partitioning adapts to needs of the system
  - Some partitions persist, others change with time
  - Further, Partititions can be Time Multiplexed
    - Services (i.e. file system), device drivers, hard realtime partitions
    - Some user-level schedulers will time-multiplex threads within a partition
- Controlled Multiplexing, *not* uncontrolled virtualization
  - Multiplexing at coarser grain (100ms?)
  - Schedule planned several slices in advance
  - Resources gang-scheduled, use of affinity or hardware partitioning to avoid cross-partition interference

Defining the Partitioned Environment

- Our new abstraction: Cell
  - A user-level software component, with guaranteed resources
  - Is it a process? Is it a Virtual Private Machine? Neither, Both
  - Different from Typical Virtual Machine Environment which duplicates many Systems components in each VM
- Properties of a Cell
  - Has full control over resources it owns ("Bare Metal")
  - Contains at least one address space (memory protection domain), but could contain more than one
  - Contains a set of secured channel endpoints to other Cells
  - Contains a security context which may protect and decrypt information
  - Interacts with trusted layers of Tessellation (e.g. the "NanoVisor") via a heavily Paravirtualized Interface
    - E.g. Manipulate address mappings without knowing format of page tables
- When mapped to the hardware, a Cell gets:
  - Gang-schedule hardware thread resources ("Harts")
  - Guaranteed fractions of other physical resources
    - Physical Pages (DRAM), Cache partitions, memory bandwidth, power
  - Guaranteed fractions of system services

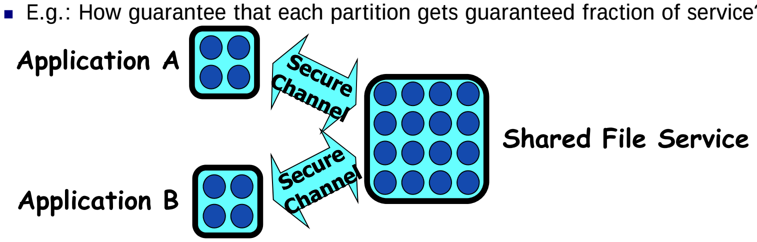
Resource Composition



- Component-based model of computation
  - Applications consist of interacting components
  - Produces composable: Performance, Interfaces, Security
- CoResident Cells ⇒ fast inter-domain communication
  - Could use hardware acceleration for fast secure messaging
  - Applications could be split into mutually distrusting partitions w/ controlled communication (echoes of μKernels)
- Fast Parallel Computation within Cells
  - Protection of computing resources not required within partition
    - High walls between partitions ⇒ anything goes within partition
    - Shared Memory/Message Passing/whatever within partition

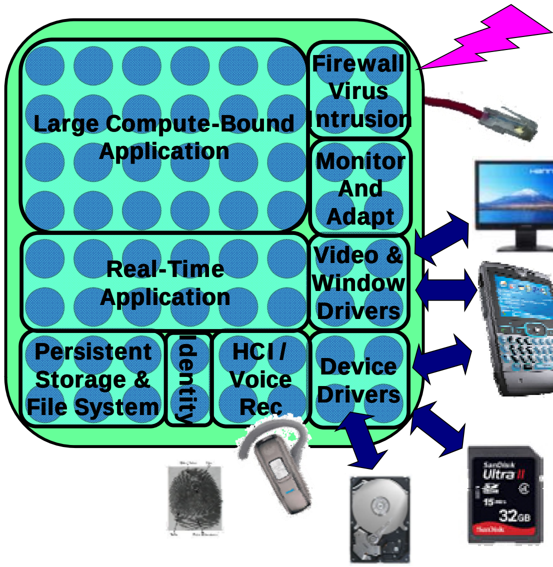
It's all about the communication

- We are interested in communication for many reasons:
  - Communication crosses resource and security boundaries
  - Efficiency of communication impacts (de)composability
- Shared components complicate resource isolation:
  - Need distributed mechanism for tracking and accounting of resources
    - E.g.: How guarantee that each partition gets guaranteed fraction of service?



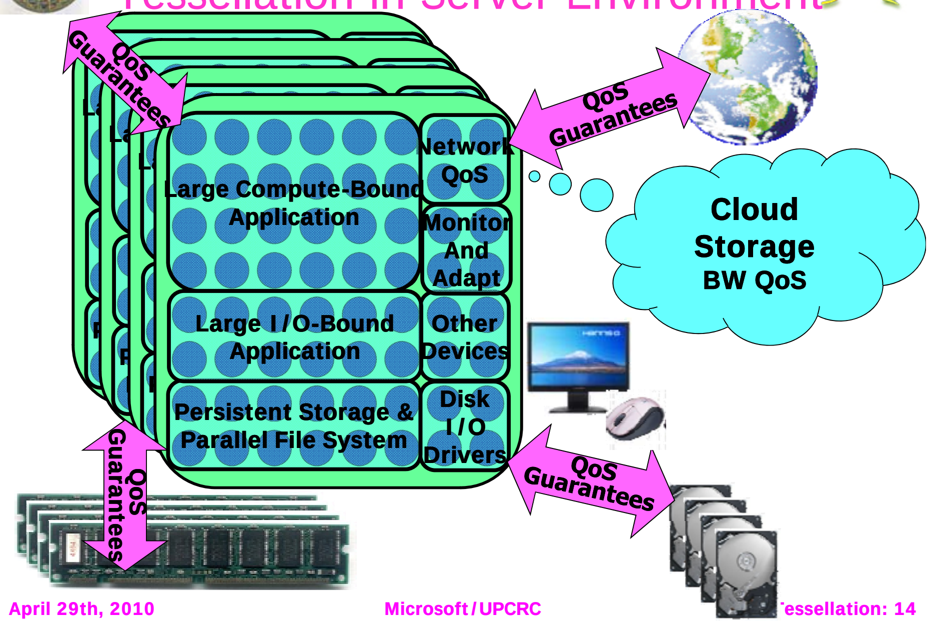
- How does presence of a message impact Cell activation?
  - Not at all (regular activation) or immediate change (interrupt-like)
- Communication defines Security Model
  - Mandatory Access Control Tagging (levels of information confidentiality)
  - Ring-based security (enforce call-gate structure with channels)

Tessellation: The Exploded OS



- Normal Components split into pieces
  - Device drivers (Security/Reliability)
  - Network Services (Performance)
    - TCP/IP stack
    - Firewall
    - Virus Checking
    - Intrusion Detection
  - Persistent Storage (Performance, Security, Reliability)
  - Monitoring services
    - Performance counters
    - Introspection
  - Identity/Environment services (Security)
    - Biometric, GPS, Possession Tracking
- Applications Given Larger Partitions
  - Freedom to use resources arbitrarily

Tessellation in Server Environment



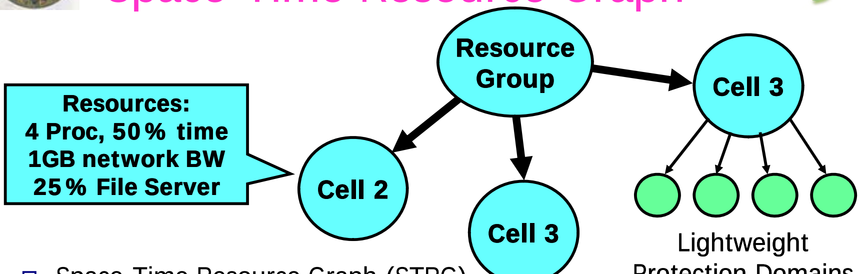
Outline

- Space-Time Partitioning
  - Two-Level Scheduling
  - Spatial Partitioning
  - Cell Model
- The Resource Management Architecture
  - Space-Time Resource Graph
  - Policy Service Architecture
  - User-Level Scheduling Support (Lithe)
- Tessellation implementation
  - Hardware Support
  - Status
  - Future Directions

Another Look: Two-Level Scheduling

- First Level: Global partitioning of resources
  - Goals: Power Budget, Overall Responsiveness/QoS, Security
    - Adjust resources to meet system level goals
  - Partitioning of CPUs, Memory, Interrupts, Devices, other resources
  - Constant for sufficient period of time to:
    - Amortize cost of global decision making
    - Allow time for partition-level scheduling to be effective
  - Hard boundaries $\Rightarrow$ interference-free use of resources for quanta
    - Allows AutoTuning of code to work well in partition
- Second Level: Application-Specific Scheduling
  - Goals: Performance, Real-time Behavior, Responsiveness, Predictability
    - Fine-grained, rapid switching
  - CPU scheduling tuned to specific applications
  - Resources distributed in application-specific fashion
  - External events (I/O, active messages, etc) deferrable as appropriate

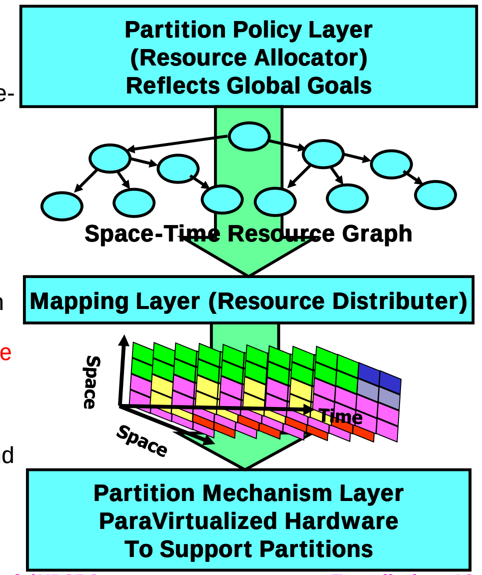
Space-Time Resource Graph



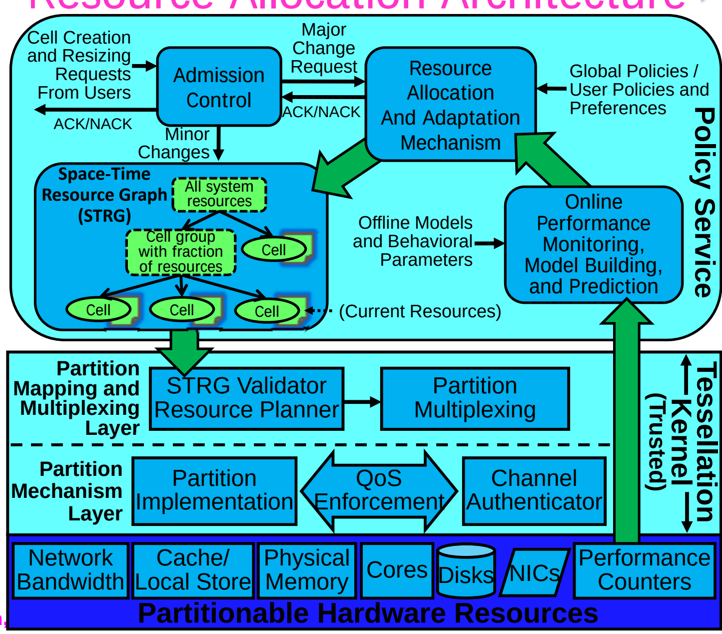
- Space-Time Resource Graph (STRG)
  - the explicit instantiation of resource assignments and relationships
- Leaves of graph hold Cells
  - All resources have a Space/Time component
    - E.g. X Processors/fraction of time, or Y Bytes/Sec
  - Resources cannot be taken away except via explicit APIs
  - Resources include fractions of OS services
- Interior Nodes
  - Resource Groups can hold resources to be shared by children
  - "Pre-Allocated" resources can be shared as excess until needed
  - Some Similarity to Resource Containers

Implementing the Space-Time Graph

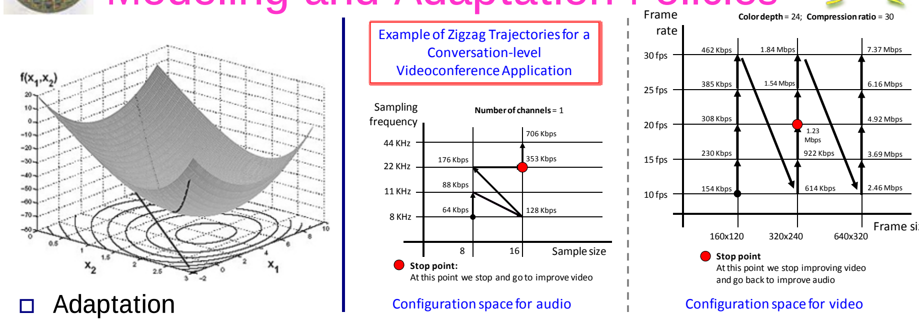
- Partition Policy Service (allocation)
  - Allocates Resources to Cells based on Global policies
  - Produces only implementable space-time resource graphs
  - May deny resources to a cell that requests them (admission control)
- Mapping Layer (distribution)
  - Makes no decisions
  - Time-Slices at a course granularity (when time-slicing necessary)
  - performs bin-packing like operation to implement space-time graph
  - In limit of *many* processors, no time multiplexing of processors, merely distributing of resources
- Partition Mechanism Layer
  - Implements hardware partitions and secure channels
  - Device Dependent: Makes use of more or less hardware support for QoS and Partitions



Resource Allocation Architecture



Modeling and Adaptation Policies



- Adaptation
  - Convex optimization
    - Relative importance of different Cells expressed via scaling functions ("Urgency")
  - Walk through Configuration space
    - Meet minimum QoS properties first, enhancement with excess resources
- User-Level Policies
  - Declarative language for describing application preferences and adaptive desires
- Modeling of Applications
  - Static Profiling: may be useful with Cell guarantees
  - Multi-variable model building
    - Get performance as function of resources
    - Or – tangent plane of performance as function of resources

Scheduling inside a cell

- Cell Scheduler can rely on:
  - Coarse-grained time quanta allows efficient fine-grained use of resources
  - Gang-Scheduling of processors within a cell
  - No unexpected removal of resources
  - Full Control over arrival of events
    - Can disable events, poll for events, etc.
- Pure environment of a Cell $\Rightarrow$ Autotuning will return same performance at runtime as during training phase
- Application-specific scheduling for performance
  - Lithe Scheduler Framework (for constructing schedulers)
    - Will be able to handle premptive scheduling/cross-address-space scheduling
  - Systematic mechanism for building composable schedulers
    - Parallel libraries with different parallelism models can be easily composed
  - Of course: preconstructed thread schedulers/models (Silk, pthreads…) as libraries for application programmers
- Application-specific scheduling for Real-Time
  - Label Cell with Time-Based Labels. Examples:
    - Run every 1s for 100ms synchronized to ± 5ms of a global time base
    - Pin a cell to 100% of some set of processors
  - Then, maintain own deadline scheduler

Discussion

- How to divide application into Cell?
  - Cells probably best for coarser-grained components
    - Fine-grained switching between Cells antithetical to stable resource guarantees
  - Division between Application components and shared OS services natural (obvious?)
    - Both for security reasons and for functional reasons
  - Division between types of scheduling
    - Real-time (both deadline-driven and rate-based), pre-scheduled
    - GUI components (responsiveness most important)
    - High-throughput (As many resources as can get)
    - Stream-based (Parallelism through decomposition into pipeline stages)
- What granularity of Application component is best for Policy Service?
  - Fewer Cells in system leads to simpler optimization problem
- Language-support for Cell model?
  - Task-based, not thread based
  - Cells produced by annotating Software Frameworks with QoS needs?
  - Cells produced automatically by just-in-time optimization?
    - i.e. Selective Just In Time Specialization or SEJITS

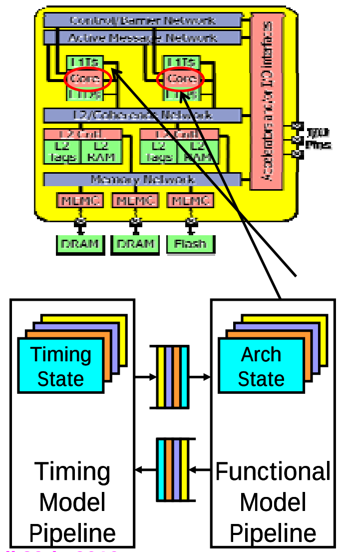
Outline

- Space-Time Partitioning
  - Two-Level Scheduling
  - Spatial Partitioning
  - Cell Model
- The Resource Management Architecture
  - Space-Time Resource Graph
  - Policy Service Architecture
  - User-Level Scheduling Support (Lithe)
- Tessellation implementation
  - Hardware Support
  - Status
  - Future Directions

What we might like from Hardware

- A good parallel computing platform (Obviously!)
  - Good synchronization, communication (Shared memory within Cells would be nice)
  - Vector, GPU, SIMD (Can exploit data parallel modes of computation)
  - Measurement: performance counters
- Partitioning Support
  - Caches: Give exclusive chunks of cache to partitions
  - High-performance barrier mechanisms partitioned properly
  - System Bandwidth
  - Power (Ability to put partitions to sleep, wake them up quickly)
- QoS Enforcement Mechanisms
  - Ability to give restricted fractions of bandwidth (memory, on-chip network)
  - Message Interface: Tracking of message rates with source-suppression for QoS
  - Examples: Globally Synchronized Frames (ISCA 2008, Lee and Asanovic)
- Fast messaging support (for channels and possible intra-cell)
  - Virtualized endpoints (direct to destination Cell when mapped, into memory FIFO when not)
  - User-level construction and disposition of messages
  - DMA, user-level notification mechanisms
  - Trusted Computing Platform (automatic decryption/encryption of channel data)

RAMP Gold: FAST Emulation of new Hardware



- RAMP emulation model for Parlab manycore
  - SPARC v8 ISA -> v9
  - Considering ARM model
- Single-socket manycore target
- Split functional/timing model, both in hardware
  - Functional model: Executes ISA
  - Timing model: Capture pipeline timing detail (can be cycle accurate)
- Host multithreading of both functional and timing models
- Built for Virtex-5 systems (ML505 or BEE3)

Tessellation Implementation Status

- First version of Tessellation
  - ~7000 lines of code in NanoVisor layer
  - Supports basic partitioning
    - Cores and caches (via page coloring)
    - Fast inter-partition channels (via ring buffers in shared memory, soon cross-network channels)
    - Use of Memory Bandwidth Partitioning (RAMP)
  - Network Driver and TCP/IP stack running in partition
    - Devices and Services available across network
  - Hard Thread interface to Lithe – a framework for constructing user-level schedulers
  - Initial version of Policy Service to come on line soon
- Currently Two ports
  - 32-core Nehalem system
  - 64-core RAMP emulation of a manycore processor (SPARC)
    - Will allow experimentation with new hardware resources
    - Examples:
      - QoS Controlled Memory/Network BW
      - Cache Partitioning
      - Fast Inter-Partition Channels with security tagging

Future Directions

- Interfaces with Parallel Patterns/Frameworks
  - Annotations with guarantees, QoS requirements, etc
  - Mapping into Cell structure statically or dynamically
- Streaming Parallel (OS?) Services
  - File Systems (local and in Cloud)
  - Other Network services
  - Interesting User Interface Devices/GUI services
    - Music, Video, Speech, Vector
- Investigate Hardware support for partitioning and QoS
- Global Orientation
  - Every software component has unique name in space
  - Can be implemented (and found) either locally or in Cloud
  - Data external to Cells automagically encrypted when crossing Cell boundaries
  - Security Implications of Cell-based architecture
    - How small can our trusted computing base really be?
    - What about the Policy Service?

Conclusion

- Space-Time Partitioning: grouping processors & resources behind hardware boundary
  - Two-level scheduling
    1) Global Distribution of resources
    2) Application-Specific scheduling of resources
  - Bare Metal Execution within partition
  - Composable performance, security, QoS
- Cells: Basic Unit of Resource and Security
  - User-Level Software Component with Guaranteed Resources
  - Secure Channels to other Cells
- Partitioning Service
  - Explicit Admission Control: Sometimes requests for resources must be denied
  - Policy-driven optimization of resources
- Tessellation OS
  - Exploded OS: spatially partitioned, interacting services
  - Exploit Hardware partitioning mechanisms when available
  - Components
    - Partitioning Mechanisms (“NanoVisor”)
    - Policy Service: Resource Management
    - OS services as independent servers

Example of Music Application