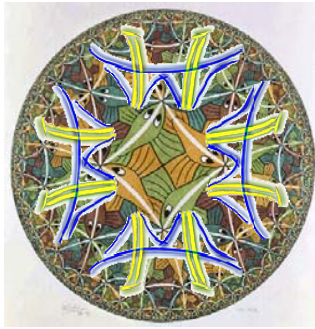


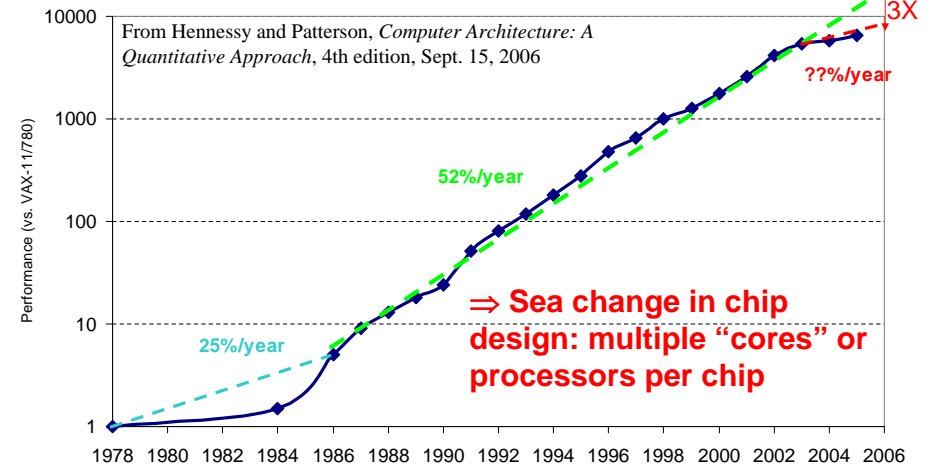
# Tessellation OS



## Architecting Systems Software in a ManyCore World

John Kubiatoiwicz  
UC Berkeley  
kubitron@cs.berkeley.edu

## Uniprocessor Performance (SPECint)



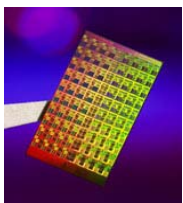
- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

November 12th, 2009

Tessellation OS

Tessellation: 2

## ManyCore Chips: The future is here



- Intel 80-core multicore chip (Feb 2007)
  - 80 simple cores
  - Two floating point engines /core
  - Mesh-like "network-on-a-chip"
  - 100 million transistors
  - 65nm feature size

- "ManyCore" refers to many processors/chip
  - 64? 128? Hard to say exact boundary
- How to program these?
  - Use 2 CPUs for video/audio
  - Use 1 for word processor, 1 for browser
  - 76 for virus checking???
- Something new is clearly needed here...

November 12th, 2009

Tessellation OS

Tessellation: 3

## Parallel Processing for the Masses

- Why is the presence of ManyCore a problem?
  - Parallel computing has been around for 40 years with mixed results
    - Many researchers, several generations, widely varying approaches
  - Parallel computing has never become a generic software solution (especially for client applications)
  - Suddenly, parallel computing will appear at all levels of our computation stack
    - Cellphones
    - Cars (yes, Bosch is thinking of replacing some of the 70 processors in a high end car with ManyCore chips)
    - Laptops, Desktops, Servers...
- Time for the computer industry to panic a bit???
- Perhaps

November 12th, 2009

Tessellation OS

Tessellation: 4

# Why might we succeed this time?

- No Killer Microprocessor to Save Programmers (No Choice)
  - No one is building a faster serial microprocessor
  - For programs to go faster, SW must use parallel HW
- New Metrics for Success (Different Criteria)
  - Perhaps linear speedup is not the primary goal
  - Real Time Latency/Responsiveness and/or MIPS/Joule
  - Just need some new killer parallel apps vs. all legacy SW must achieve linear speedup
- Necessity: All the Wood Behind One Arrow (More Manpower)
  - Whole industry committed, so more working on it
  - If future growth of IT depends on faster processing at same price (vs. lowering costs like NetBook)
- User-Interactive Applications Exhibit Parallelism (New Apps)
  - Multimedia, Speech Recognition, situational awareness
- Multicore Synergy with Cloud Computing (Different Focus)
  - Cloud Computing apps parallel even if client not parallel
  - Manycore is cost-reduction, not radical SW disruption

November 12th, 2009

Tessellation OS

Tessellation: 5

# Outline

- What is the problem (Did this already)
- Berkeley Parlab
  - Structure
  - Applications
  - Software Engineering
- Space-Time Partitioning
  - RAPPidS goals
  - Partitions, QoS, and Two-Level Scheduling
- The Cell Model
  - Space-Time Resource Graph
  - User-Level Scheduling Support (Lithe)
- Tessellation implementation
  - Hardware Support
  - Tessellation Software Stack
  - Status

November 12th, 2009

Tessellation OS

Tessellation: 6

# ParLab: a Fresh Approach to Parallelism

- What is the ParLAB?
  - A new Laboratory on Parallelism at Berkeley
    - Remodeled "open floorplan" space on 5<sup>th</sup> floor of Soda Hall
    - 10+ faculty, some two-feet in, others collaborating
  - Funded by Intel, Microsoft, and other affiliate partners
  - Goal: Productive, Efficient, Correct, Portable SW for 100+ cores & scale as core increase every 2 years (!)
  - Application Driven! (really!)
- Some History
  - Berkeley researchers from many backgrounds started meeting in Feb. 2005 to discuss parallelism
    - Circuit design, computer architecture, massively parallel computing, computer-aided design, embedded hardware and software, programming languages, compilers, scientific programming, and numerical analysis
    - Considered successes in high-performance computing (LBNL) and parallel embedded computing (BWRC)
  - Led to "Berkeley View" Tech. Report 12/2006 and new Parallel Computing Laboratory ("Par Lab")
    - Won invited competition from Intel/MS of top 25 CS Departments

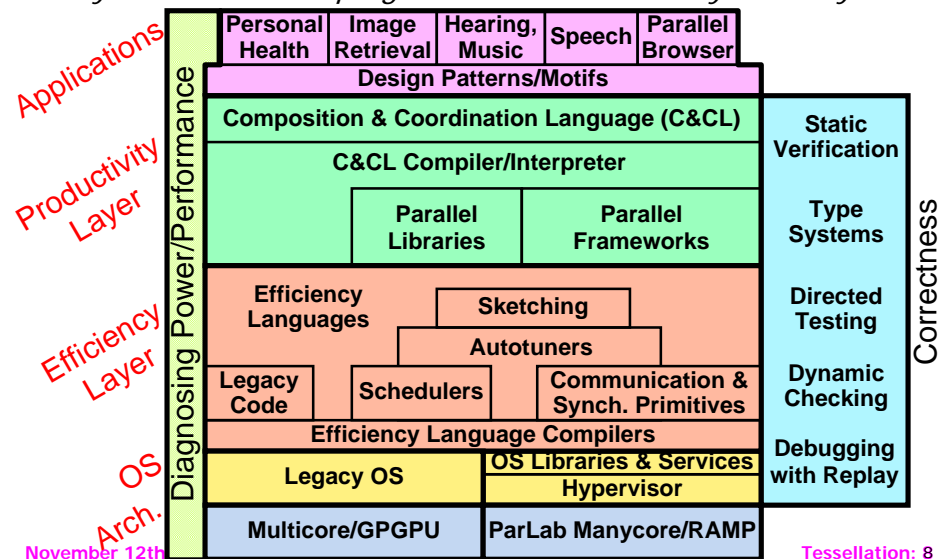
November 12th, 2009

Tessellation OS

Tessellation: 7

# Par Lab Research Overview

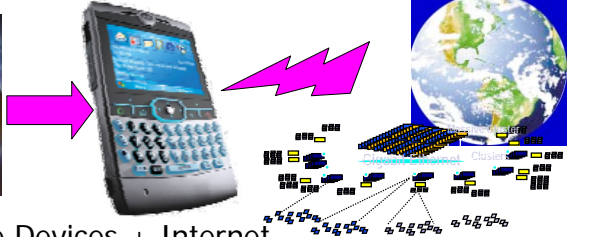
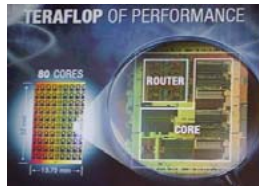
*Easy to write correct programs that run efficiently on manycore*



November 12th

Tessellation: 8

# Target Environment: Client Computing



- ManyCore + Mobile Devices + Internet
  - Lots of Computational Resources
    - Must enable massive parallelism (not get in the way)
  - Many (relatively) Limited Resources:
    - Power, I/O bandwidth, Memory Bandwidth, User patience...
    - Must use these as efficiently as possible
  - Services backed by vast Internet resources
    - Information can be preserved elsewhere
    - Access to remote resources must be streamlined
    - Obvious use of ManyCore in Services – but this is not the real problem
- Things we are willing to change:
  - Software Engineering, Libraries, APIs, Services, Hardware

November 12th, 2009

Tessellation OS

Tessellation: 9

# Music and Hearing Application (David Wessel)

- Musicians have an insatiable appetite for computation + real-time demands
  - More channels, instruments, more processing, more interaction!
  - Latency must be low (5 ms)
  - Must be reliable (No clicks!)
- 1. Music Enhancer
  - Enhanced sound delivery systems for home sound systems using large microphone and speaker arrays
  - Laptop/Handheld recreate 3D sound over ear buds
- 2. Hearing Augmenter
  - Handheld as accelerator for hearing aid
- 3. Novel Instrument User Interface
  - New composition and performance systems beyond keyboards
  - Input device for Laptop/Handheld



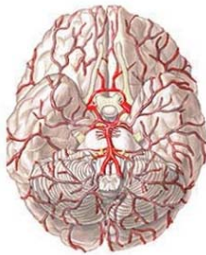
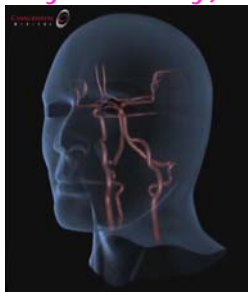
Berkeley Center for New Music and Audio Technology (CNMAT) created a compact loudspeaker array: 10-inch-diameter icosahedron incorporating 120 tweeters.

November 12th, 2009

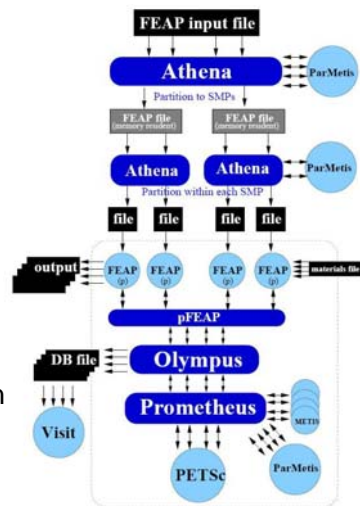
Tessellation OS

Tessellation: 10

# Health Application: Stroke Treatment (Tony Keaveny)



Bottom view of brain © ADAM, Inc.



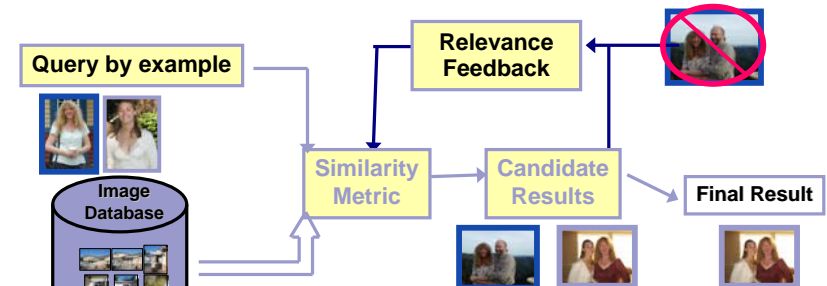
- Stroke treatment time-critical, need supercomputer performance in hospital
- Goal: First true 3D Fluid-Solid Interaction analysis of Circle of Willis
- Based on existing codes for distributed clusters

November 12th, 2009

Tessellation OS

Tessellation: 11

# Content-Based Image Retrieval (Kurt Keutzer)



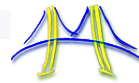
- Built around Key Characteristics of personal databases
  - Very large number of pictures (>5K)
  - Non-labeled images
  - Many pictures of few people
  - Complex pictures including people, events, places, and objects

November 12th, 2009

Tessellation OS

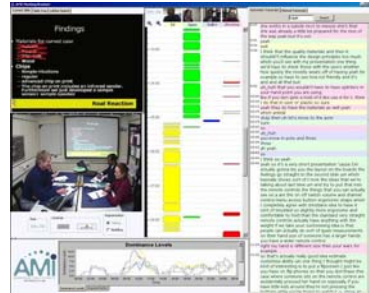
Tessellation: 12

# Robust Speech Recognition

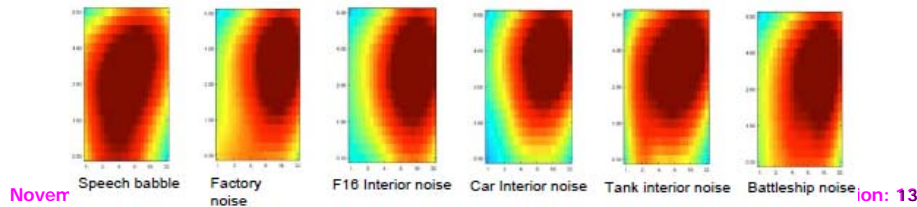


(Nelson Morgan)

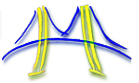
- Meeting Diarist
  - Laptops/ Handhelds at meeting coordinate to create speaker identified, partially transcribed text diary of meeting



■ Use cortically-inspired manystream spatio-temporal features to tolerate noise

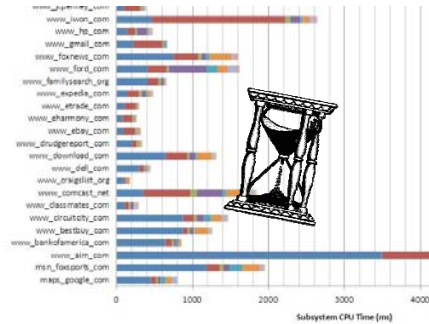


# Parallel Browser



(Ras Bodik)

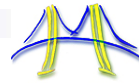
- Goal: Desktop quality browsing on handhelds
  - Enabled by 4G networks, better output devices
- Bottlenecks to parallelize
  - Parsing, Rendering, Scripting



lation OS

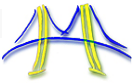
Tessellation: 14

# Parallel Software Engineering



- How do we hope to tackle parallel programming?
  - Through Software Engineering and Control of Resources
- Two type of programmers:
  - Productivity programmers (90% of programmers)
    - Not parallel programmers, rather domain specific programmers
  - Efficiency programmers (10% of programmers)
    - Parallel programmers, extremely competent at handling parallel programming issues
- Target new ways to express software so that is can be execute in parallel
  - Parallel Patterns
- System support to avoid "getting in the way" of the result
  - Parallel Libraries, Autotuning, On-the-fly compilation
  - Explicitly managed resource containers (Partitions)

# Architecting Parallel Software with Patterns



(Kurt Keutzer/Tim Mattson)

Our initial survey of many applications brought out common recurring patterns:

"Dwarfs" -> Motifs

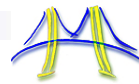
- Computational patterns
- Structural patterns

Insight: Successful codes have a comprehensible software architecture:

- Patterns give human language in which to describe architecture



## Motif (nee "Dwarf") Popularity (Red Hot / Blue Cool)



- How do compelling apps relate to 12 motifs?

	Embed	SPEC	DB	Games	ML	CAD	HPC	Health	Image	Speech	Music	Browser
1 Finite State Mach.	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
2 Circuits	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
3 Graph Algorithms	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
4 Structured Grid	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
5 Dense Matrix	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
6 Sparse Matrix	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
7 Spectral (FFT)	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
8 Dynamic Prog	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
9 Particle Methods	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
10 Backtrack/ B&B	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
11 Graphical Models	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
12 Unstructured Grid	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red

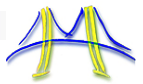
November 12th, 2009

Tessellation OS

Tessellation: 17



## Architecting Parallel Software



### Decompose Tasks/Data

#### Order tasks

#### Identify Data Sharing and Access

#### Identify the Software Structure

- Pipe-and-Filter
- Agent-and-Repository
- Event-based
- Bulk Synchronous
- MapReduce
- Layered Systems
- Arbitrary Task Graphs

#### Identify the Key Computations

- Graph Algorithms
- Dynamic programming
- Dense/Sparse Linear Algebra
- (Un)Structured Grids
- Graphical Models
- Finite State Machines
- Backtrack Branch-and-Bound
- N-Body Methods
- Circuits
- Spectral Methods



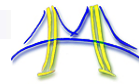
November 12th, 2009

Tessellation OS

Tessellation: 18



## Par Lab is Multi-Lingual



- Applications require ability to compose parallel code written in many languages and several different parallel programming models
  - Let application writer choose language/model best suited to task
  - High-level productivity code and low-level efficiency code
  - Old legacy code plus shiny new code
- Correctness through all means possible
  - Static verification, annotations, directed testing, dynamic checking
  - Framework-specific constraints on non-determinism
  - Programmer-specified semantic determinism
  - Require common spec between languages for static checker
- Common linking format at low level (Lithe) not intermediate compiler form
  - Support hand-tuned code and future languages & parallel models

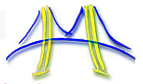
November 12th, 2009

Tessellation OS

Tessellation: 19



## Selective Embedded Just-In-Time Specialization (SEJITS) for Productivity (Armando Fox)



- Modern scripting languages (e.g., Python and Ruby) have powerful language features and are easy to use
- Idea: Dynamically generate source code in C within the context of a Python or Ruby interpreter, allowing app to be written using Python or Ruby abstractions but automatically generating, compiling C at runtime
- Like a JIT but
  - Selective:** Targets a particular method and a particular language/platform (C+OpenMP on multicore or CUDA on GPU)
  - Embedded:** Make specialization machinery productive by implementing in Python or Ruby itself by exploiting key features: introspection, runtime dynamic linking, and foreign function interfaces with language-neutral data representation

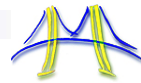
November 12th, 2009

Tessellation OS

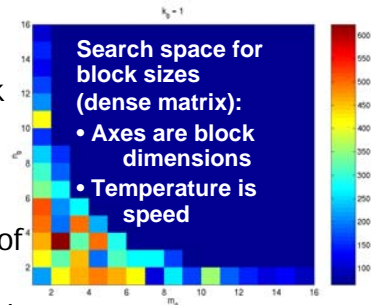
Tessellation: 20

# Autotuning for Code Generation

(Demmel, Yelick)



- Problem: generating optimal code like searching for needle in haystack
- Manycore → even more diverse
- New approach: “Auto-tuners”
  - 1st generate program variations of combinations of optimizations (blocking, prefetching, ...) and data structures
  - Then compile and run to heuristically search for best code for *that* computer
- Examples: PHiPAC (BLAS), Atlas (BLAS), Spiral (DSP), FFT-W (FFT)



November 12th, 2009

Tessellation OS

Tessellation: 21

# Outline

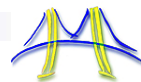
- What is the problem (Did this already)
- Berkeley Parlab
  - Structure
  - Applications
  - Software Engineering
- Space-Time Partitioning
  - RAPPidS goals
  - Partitions, QoS, and Two-Level Scheduling
- The Cell Model
  - Space-Time Resource Graph
  - User-Level Scheduling Support (Lithe)
- Tessellation implementation
  - Hardware Support
  - Tessellation Software Stack
  - Status

November 12th, 2009

Tessellation OS

Tessellation: 22

# 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?
- 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

November 12th, 2009

Tessellation OS

Tessellation: 23

# PARLab OS Goals: RAPPidS



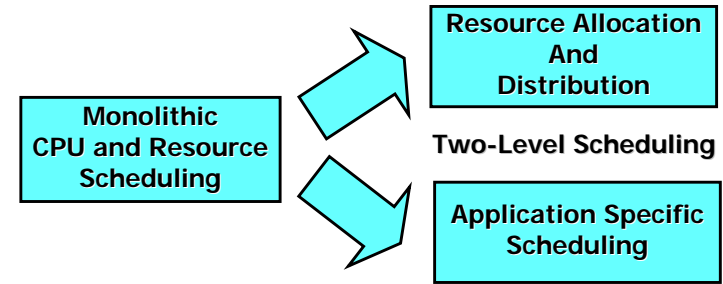
- Responsiveness: Meets real-time guarantees
  - Good user experience with UI expected
  - Illusion of Rapid I/O while still providing guarantees
  - Real-Time applications (speech, music, video) will be assumed
- Agility: Can deal with rapidly changing environment
  - Programs not completely assembled 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

November 12th, 2009

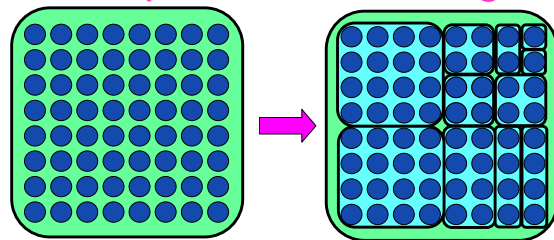
Tessellation OS

Tessellation: 24

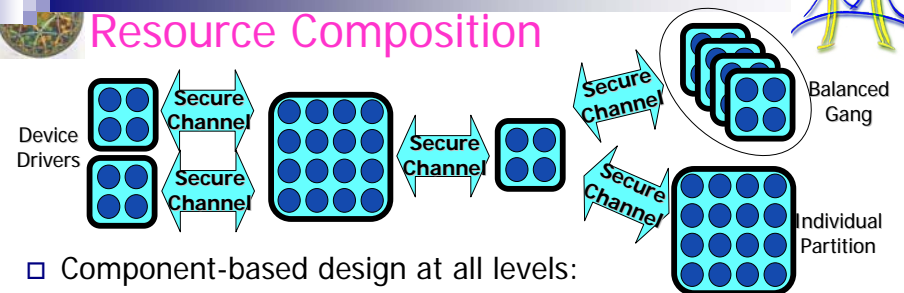
- What is wrong with current Operating Systems?
  - They do not allow expression of application requirements
    - Minimal Frame Rate, Minimal Memory Bandwidth, Minimal QoS from system Services, Real Time Constraints, ...
    - No clean interfaces for reflecting these requirements
  - They do not provide 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 do not provide fully custom scheduling
    - In a parallel programming environment, ideal scheduling can depend crucially on the programming model
  - They 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



- 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

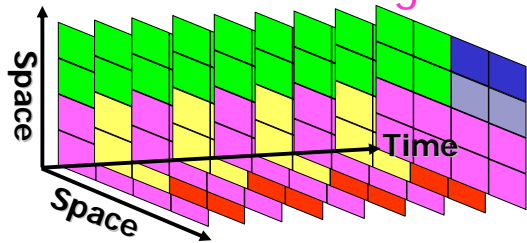


- Spatial Partition: group of processors acting within hardware boundary
  - Boundaries are "hard", communication between partitions controlled
  - Anything goes within partition
- Each Partition receives a *vector* of resources
  - Some number of dedicated processors
  - Some set of dedicated resources (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



- Component-based design at all levels:
  - Applications consist of interacting components
  - Requires composable: Performance, Interfaces, Security
- Spatial Partitioning Helps:
  - Protection of computing resources not required within partition
    - High walls between partitions ⇒ anything goes within partition
    - "Bare Metal" access to hardware resources
    - Shared Memory/Message Passing/whatever within partition
  - Partitions exist simultaneously ⇒ fast inter-domain communication
    - Applications split into mutually distrusting partitions w/ controlled communication (echoes of  $\mu$ Kernels)
    - Hardware acceleration/tagging for fast secure messaging

# Space-Time Partitioning



- Spatial Partitioning Varies over Time
  - Partitioning adapts to needs of the system
  - Some partitions persist, others change with time
  - Further, Partitions 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
- Global Partitioning Goals:
  - Power-performance tradeoffs
  - Setup to achieve QoS and/or Responsiveness guarantees
  - Isolation of real-time partitions for better guarantees

November 12th, 2009

Tessellation OS

Tessellation: 29

# Another Look: Two-Level Scheduling

- First Level: Gross partitioning of resources
  - Goals: Power Budget, Overall Responsiveness/QoS, Security
  - 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 ⇒ 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
  - CPU scheduling tuned to specific applications
  - Resources distributed in application-specific fashion
  - External events (I/O, active messages, etc) deferrable as appropriate
- Justifications for two-level scheduling?
  - Global/cross-app decisions made by 1<sup>st</sup> level
    - E.g. Save power by focusing I/O handling to smaller number of cores
  - App-scheduler (2<sup>nd</sup> level) better tuned to application
    - Lower overhead/better match to app than global scheduler
    - No global scheduler could handle all applications

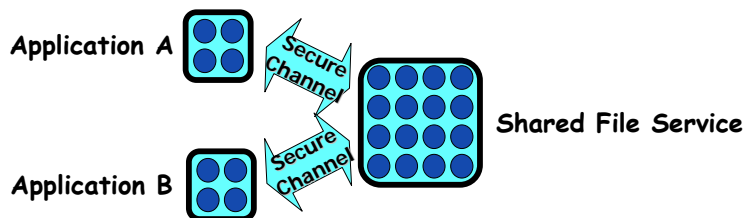
November 12th, 2009

Tessellation OS

Tessellation: 30

# It's all about the communication

- We are interested in communication for many reasons:
  - Communication represents a security vulnerability
  - Quality of Service (QoS) boils down message tracking
  - Communication efficiency impacts decomposability
- Shared components complicate resource isolation:
  - Need distributed mechanism for tracking and accounting of resource usage
    - E.g.: How do we guarantee that each partition gets a guaranteed fraction of the service:

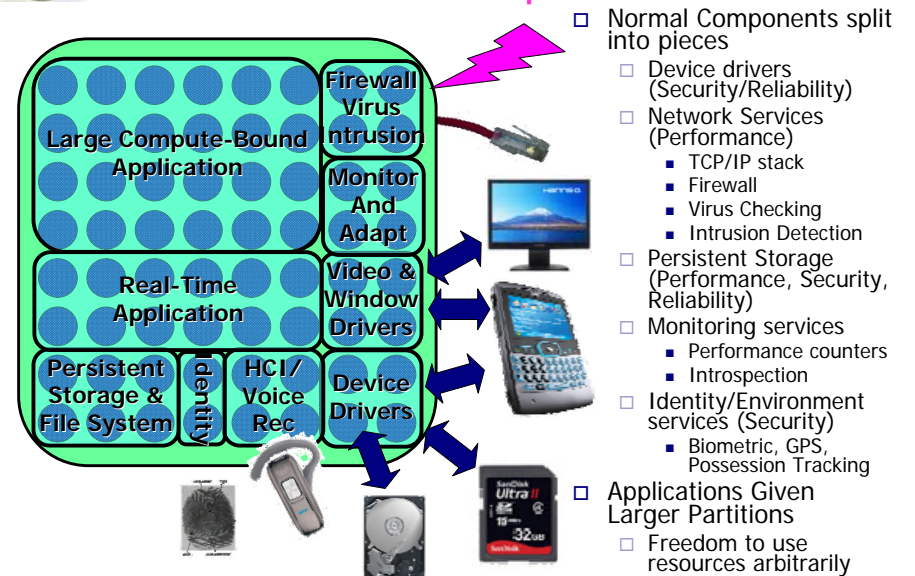


November 12th, 2009

Tessellation OS

Tessellation: 31

# Tessellation: The Exploded OS



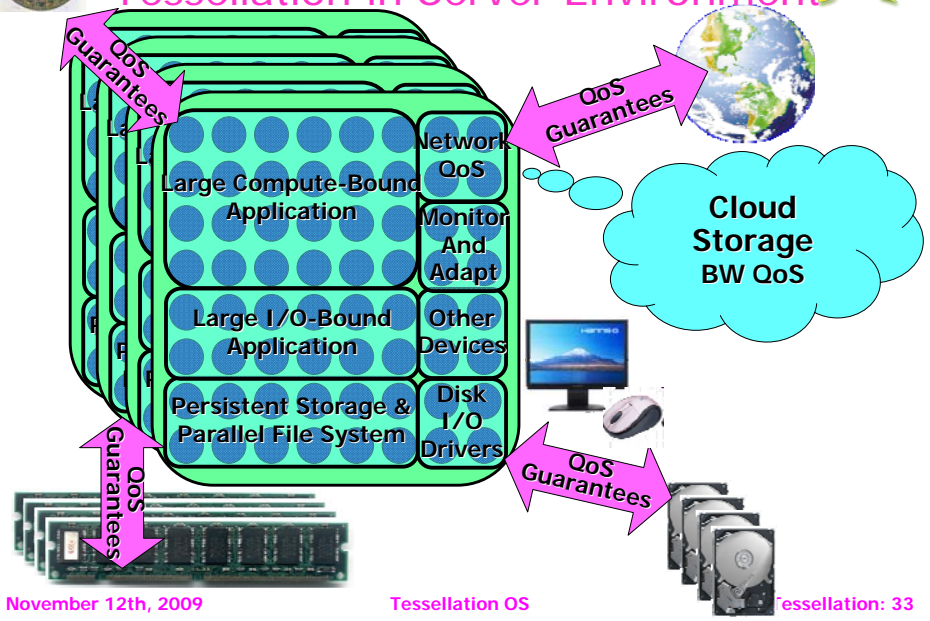
- Normal Components split into pieces
  - Device drivers (Security/Reliability)
  - Network Services (Performance)
    - TCP/IP stack
    - Firewall
    - Virus Checking
    - Intrusion Detection
  - Persistent Storage (Performance, Security, Reliability)
  - Monitoring services
    - Performance counters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

November 12th, 2009

Tessellation OS

Tessellation: 32

# Tessellation in Server Environment



November 12th, 2009

Tessellation OS

Tessellation: 33

# Outline

- What is the problem (Did this already)
- Berkeley Parlab
  - Structure
  - Applications
  - Software Engineering
- Space-Time Partitioning
  - RAPPidS goals
  - Partitions, QoS, and Two-Level Scheduling
- The Cell Model
  - Space-Time Resource Graph
  - User-Level Scheduling Support (Lithe)
- Tessellation implementation
  - Hardware Support
  - Tessellation Software Stack
  - Status

November 12th, 2009

Tessellation OS

Tessellation: 34

# Defining the Partitioned Environment

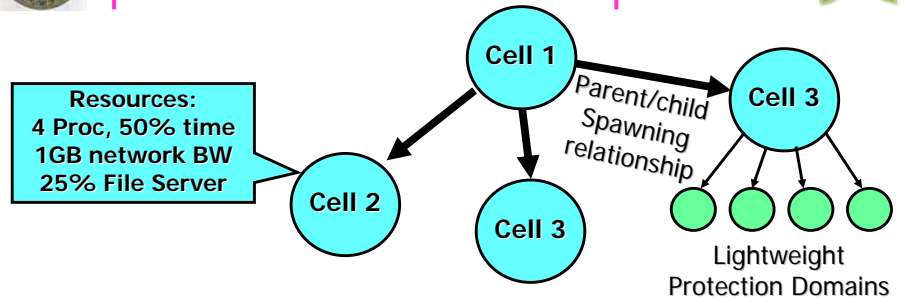
- Cell: a bundle of code, with guaranteed resources, running at user level
  - Has full control over resources it owns ("Bare Metal")
  - Contains at least one address space (memory protection domain), but could contain more than one
  - Contains a set of secured channel endpoints to other Cells
  - Interacts with trusted layers of Tessellation (e.g. the "NanoVisor") via a heavily Paravirtualized Interface
    - E.g. Can manipulate its address mappings but does not know what page tables even look like
  - We think of these as components of an application or the OS
- 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

November 12th, 2009

Tessellation OS

Tessellation: 35

# Space-Time Resource Graph



- Space-Time resource graph: the explicit instantiation of resource assignments
  - Directed Arrows Express Parent/Child Spawning Relationship
  - All resources have a Space/Time component
    - E.g. X Processors/fraction of time, or Y Bytes/Sec
- What does it mean to give resources to a Cell?
  - The Cell has a position in the Space-Time resource graph and
  - The resources are added to the cell's resource label
  - Resources cannot be taken away except via explicit APIs

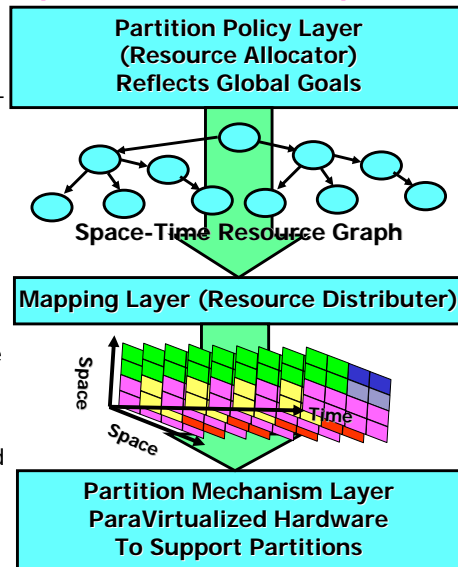
November 12th, 2009

Tessellation OS

Tessellation: 36

## Implementing the Space-Time Graph

- Partition Policy layer (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 processors, merely distributing resources
- Partition Mechanism Layer
  - Implements hardware partitions and secure channels
  - Device Dependent: Makes use of more or less hardware support for QoS and Partitions



November 12th, 2009

Tessellation OS

Tessellation: 37

## What happens in a Cell Stays in a Cell

- Cells are performance and security isolated from all other cells
  - Processors and resources are gang-scheduled
    - All fine-grained scheduling done by a user-level scheduler
  - Unpredictable resource virtualization does not occur
    - Example: no paging without linking a paging library
  - Cells can control delivery of all events
    - Message arrivals (along channels)
    - Page faults, timer interrupts (for user-level preemptive scheduling), exceptions, etc
  - Cells start with single protection domain, but can request more as desired
    - Initial protection domain becomes primary
    - For now, protection domains are Address Spaces, but can be other things as well
- CellOS: A layer of code within a Cell that looks like a traditional OS
  - Not required for all Cells!
  - On Demand Paging, Address Space management, Preemptive scheduling of multiple address spaces (i.e. processes)

November 12th, 2009

Tessellation OS

Tessellation: 38

## Scheduling inside a cell

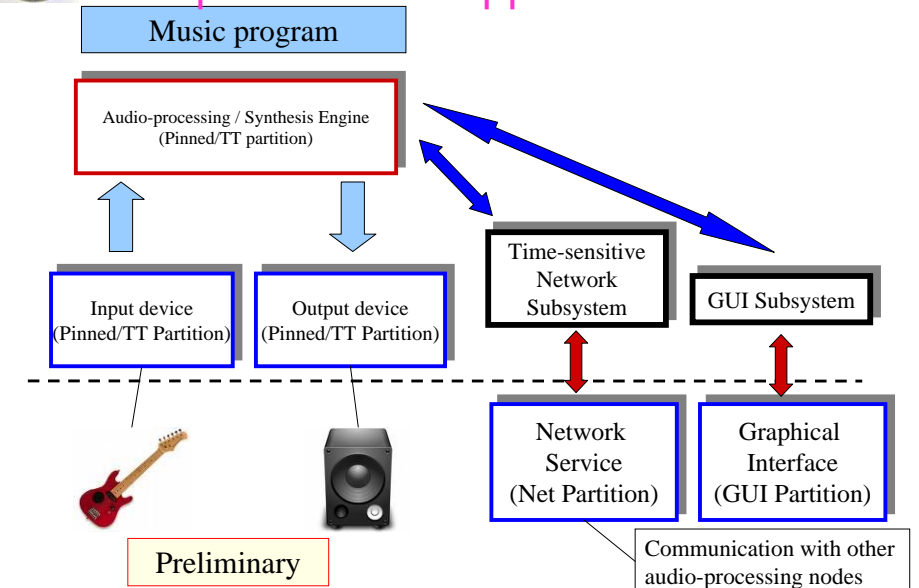
- Cell Scheduler can rely on:
  - Course-grained time quanta allowing efficient fine-grained use of resources
  - Gang-Scheduling of processors within a cell
  - No unexpected removal of resources
  - Full Control over arrival of events
    - Can disable events, poll for events, etc.
- Application-specific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

November 12th, 2009

Tessellation OS

Tessellation: 39

## Example of Music Application

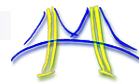


Preliminary

Communication with other audio-processing nodes



# Outline



- What is the problem (Did this already)
- Berkeley Parlab
  - Structure
  - Applications
  - Software Engineering
- Space-Time Partitioning
  - RAPPidS goals
  - Partitions, QoS, and Two-Level Scheduling
- The Cell Model
  - Space-Time Resource Graph
  - User-Level Scheduling Support (Lithe)
- Tessellation implementation
  - Hardware Support
  - Tessellation Software Stack
  - Status

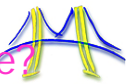
November 12th, 2009

Tessellation OS

Tessellation: 41



# What would we like from the Hardware?



- A good parallel computing platform (Obviously!)
  - Good synchronization, communication
    - On chip => Can do fast barrier synchronization with combinational logic
    - Shared memory relatively easy on chip
  - Vector, GPU, SIMD
    - Can exploit data parallel modes of computation
  - Measurement: performance counters
- Partitioning Support
  - Caches: Give exclusive chunks of cache to partitions
    - Techniques such as page coloring are poor-man's equivalent
  - Memory: Ability to restrict chunks of memory to a given partition
    - Partition-physical to physical mapping: 16MB page sizes?
  - High-performance barrier mechanisms partitioned properly
  - System Bandwidth
  - Power
    - Ability to put partitions to sleep, wake them up quickly
- Fast messaging support
  - Used for inter-partition communication
  - DMA, user-level notification mechanisms
  - Secure Tagging?
- QoS Enforcement Mechanisms
  - Ability to give restricted fractions of bandwidth
  - Message Interface: Tracking of message rates with source-suppression for QoS
  - Examples: Globally Synchronized Frames (ISCA 2008, Lee and Asanovic)

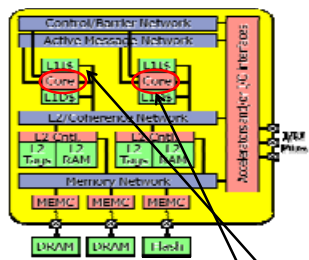
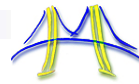
November 12th, 2009

Tessellation OS

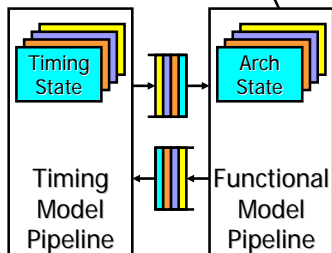
Tessellation: 42



# 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)



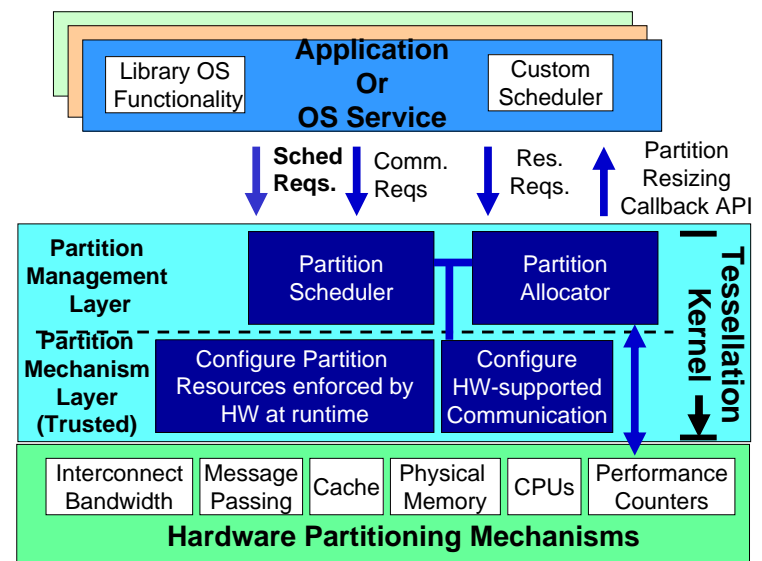
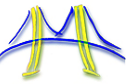
November 12th, 2009

Tessellation OS

Tessellation: 43



# Tessellation Architecture



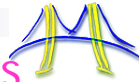
November 12th, 2009

Tessellation OS

Tessellation: 44



## 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)
  - 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
- Currently Two ports
  - 4-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

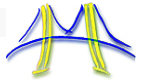
November 12th, 2009

Tessellation OS

Tessellation: 45



## Conclusion



- Berkeley ParLAB
  - Application Driven: New exciting parallel applications
  - Tackling the parallel programming problem via Software Engineering
  - Parallel Programming Motifs
- Space-Time Partitioning: grouping processors & resources behind hardware boundary
  - Focus on Quality of Service
  - Two-level scheduling
    - 1) Global Distribution of resources
    - 2) Application-Specific scheduling of resources
  - Bare Metal Execution within partition
  - Composable performance, security, QoS
- Tessellation OS

November 12th, 2009

Tessellation OS

Tessellation: 46