Tessellation Operating System

Building a real-time, responsive, high-throughput client OS for many-core architectures

Juan A. Colmenares,[1] Sarah Bird,[1] Gage Eads,[1] Steven Hofmeyr,[2] Albert Kim,[1] Rohit Poddar,[1] Hilfi Alkaff,[1] Krste Asanović,[1] and John Kubiatowicz[1]

[1] Par Lab, UC Berkeley – [2] Future Technologies Group, LBNL

http://tessellation.cs.berkeley.edu

Parallel Computing Laboratory

1. Basic Goals

- Support a dynamic mix of high-throughput parallel, interactive, and real-time applications
- Allow applications to consistently deliver performance in presence of other applications with conflicting requirements
- Enable adaptation to changes in the application mix and resource availability

2. Design Principles

Two-level Scheduling

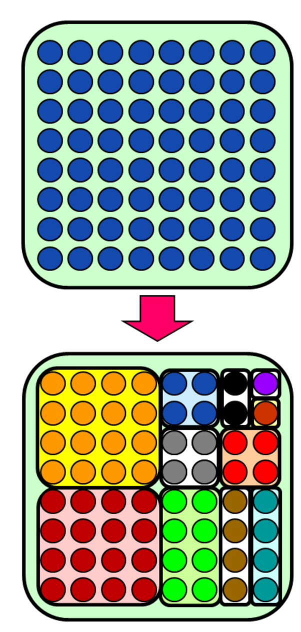
Level 2
Fine-grained Application-specific Scheduling

- Applications utilize their resources in any way they see fit
- Other components of the system cannot interfere with their use of resources

Level 1
Coarse-grained Resource Allocation and Distribution

- Chunks of resources distributed to application or system components
- Option to simply turn off unused resources

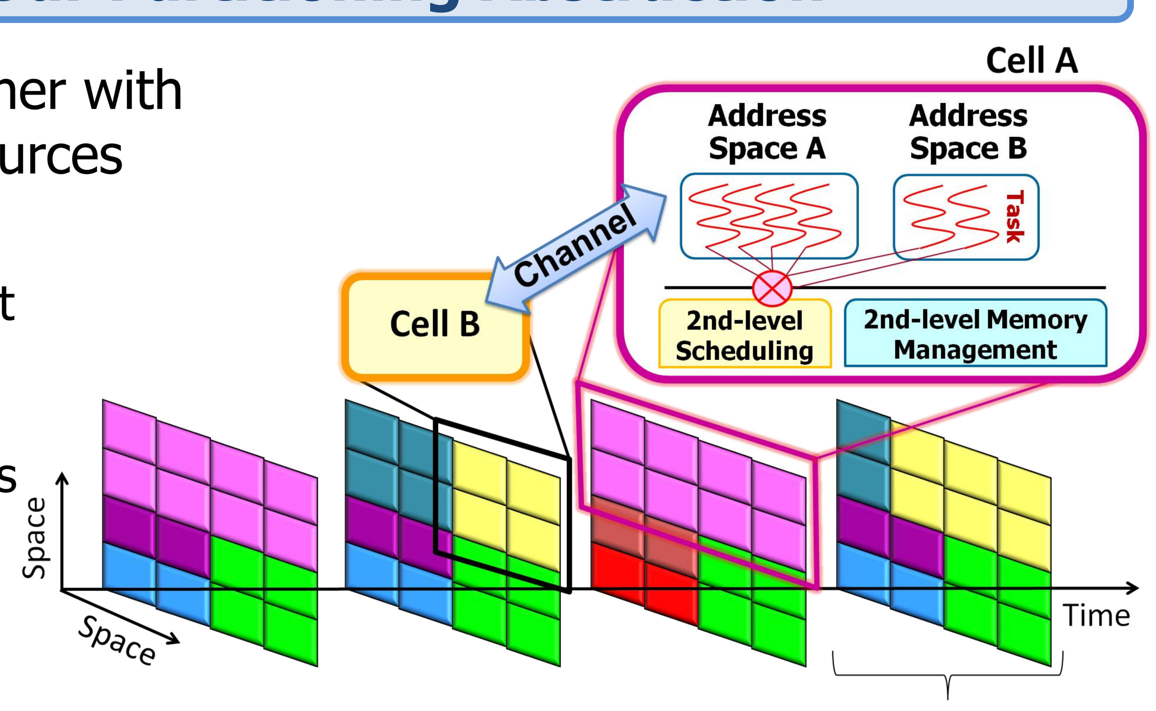
Space-Time Partitioning



- Spatial Partition: Key for performance isolation
  - Hard boundaries and controlled communication between partitions
- Each partition receives a vector of basic resources:
  - A number of hardware threads, a portion of physical memory, cache segments, and memory bandwidth
- A partition may also receive
  - Exclusive access to other resources (e.g., a hardware device and raw storage partition)
  - Guaranteed fractional services from other partitions (e.g., network service)
- Spatial partitioning is *not* static; it may vary over time
  - Partitioning adapts to needs of the system
  - Partitions can be time multiplexed; resources are gang-scheduled

3. The Cell: Our Partitioning Abstraction

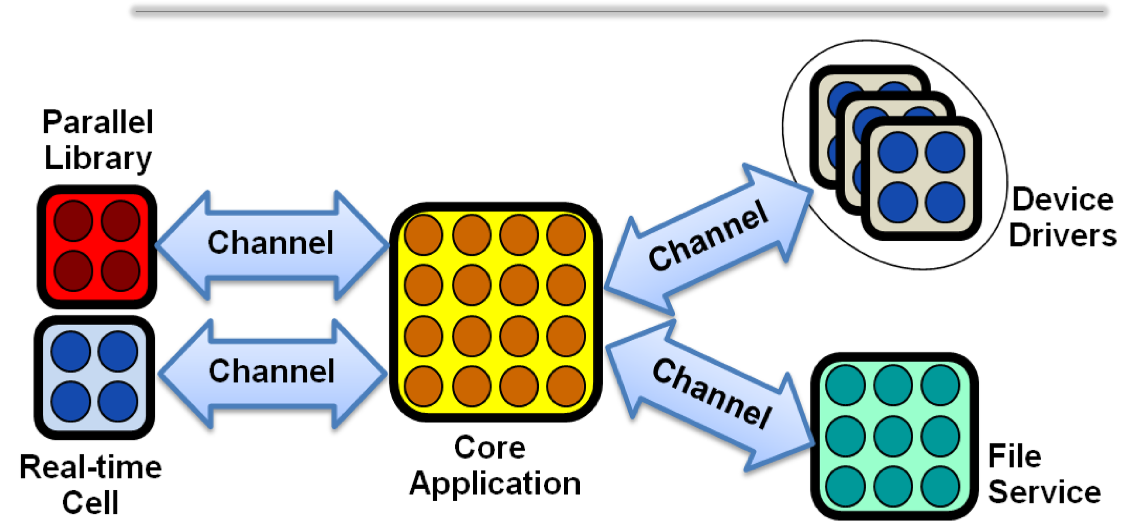
- User-level software container with guaranteed access to resources
- Basic properties of a cell
  - Full control over resources it owns when mapped to hardware
  - One or more address spaces (protection domains)
  - Efficient inter-cell communication channels



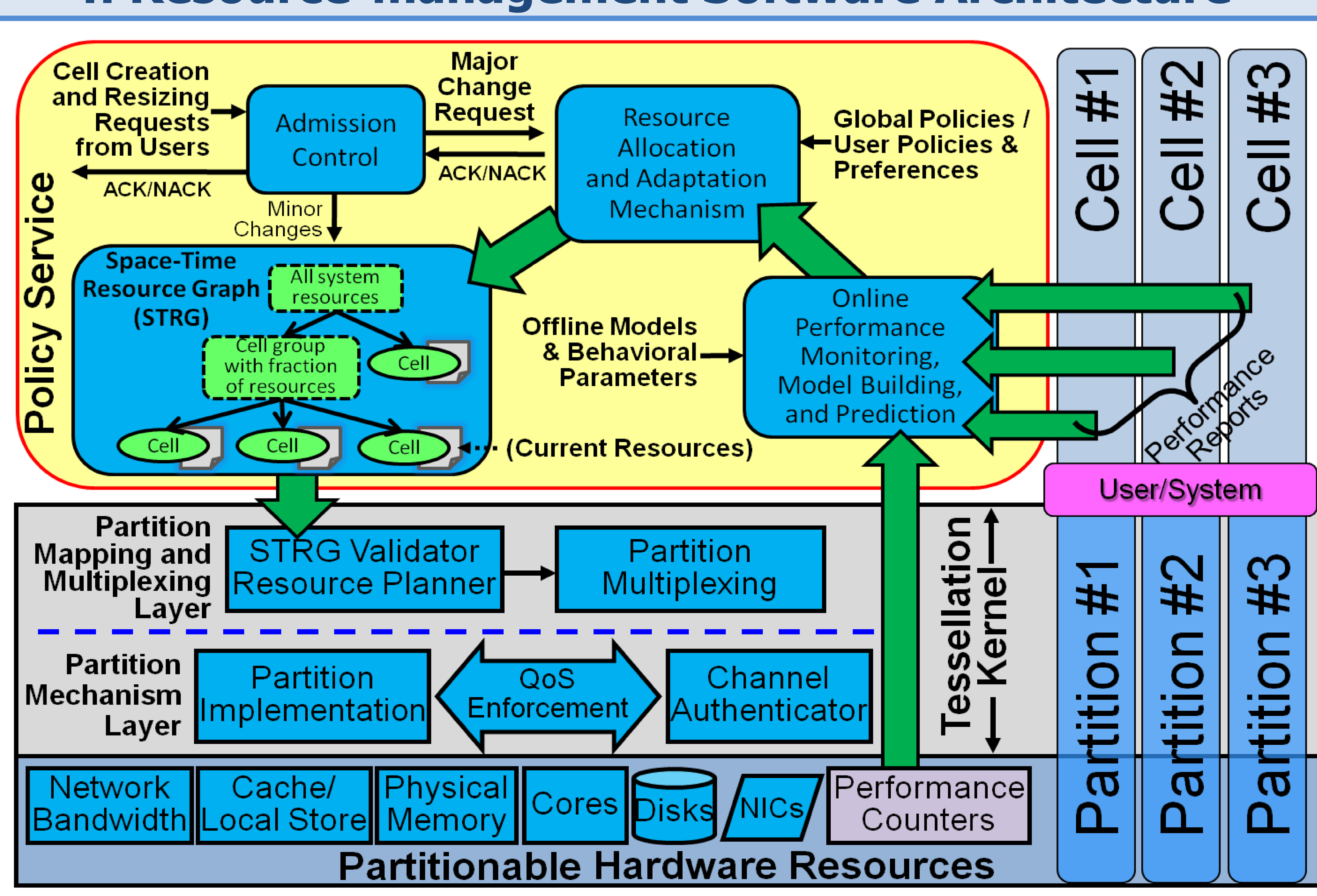
Yellow partition grows due to system adaptation

Component-based Model with Composable Performance

- Applications = Set of interacting components deployed on different cells
  - Applications split into performance-incompatible and mutually distrusting cells
  - OS services are independent agents that provide QoS



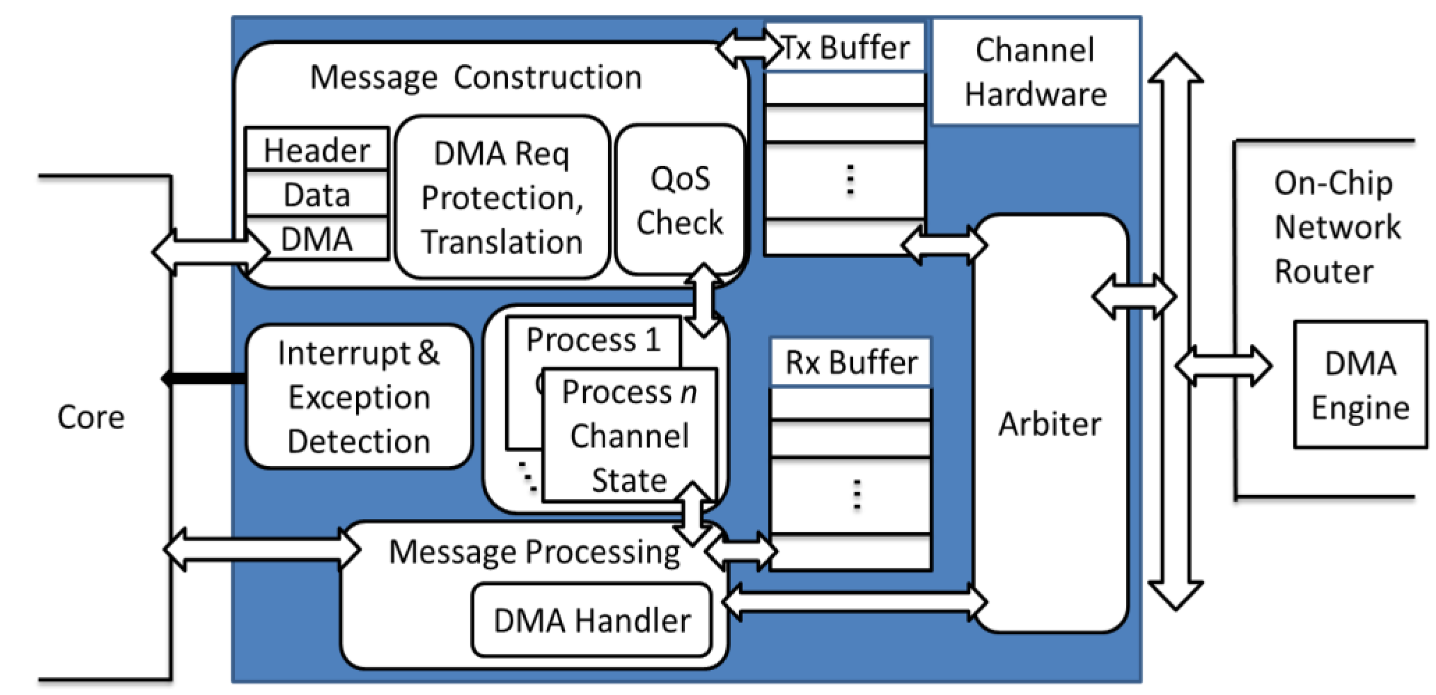
4. Resource-management Software Architecture



4. Beneficial Hardware Enhancements

Hardware-acceleration for Inter-cell Channels

Improves efficiency of inter-cell communication

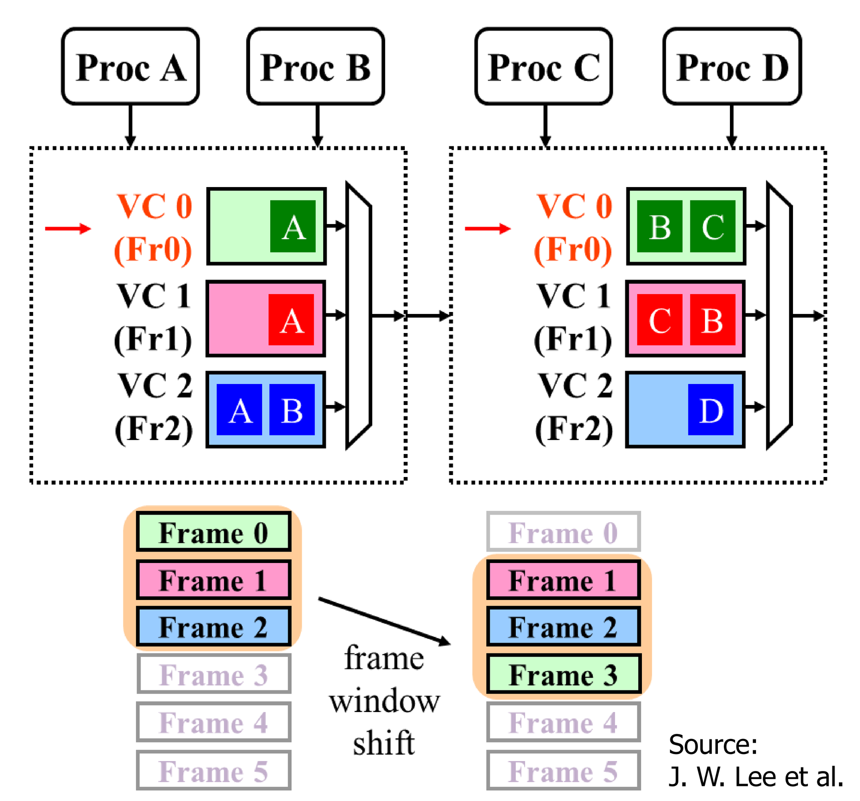


- Channel virtualization enables use by multiple cells
- Hardware-based protection, translation, and message-processing mechanisms minimize kernel intervention
- QoS guarantees in the channel state, enforced in QoS block

Hardware Partitioning Mechanisms

- Provide stronger performance isolation between cells
- Besides those found on commodity hardware, we propose the use of the following mechanisms:

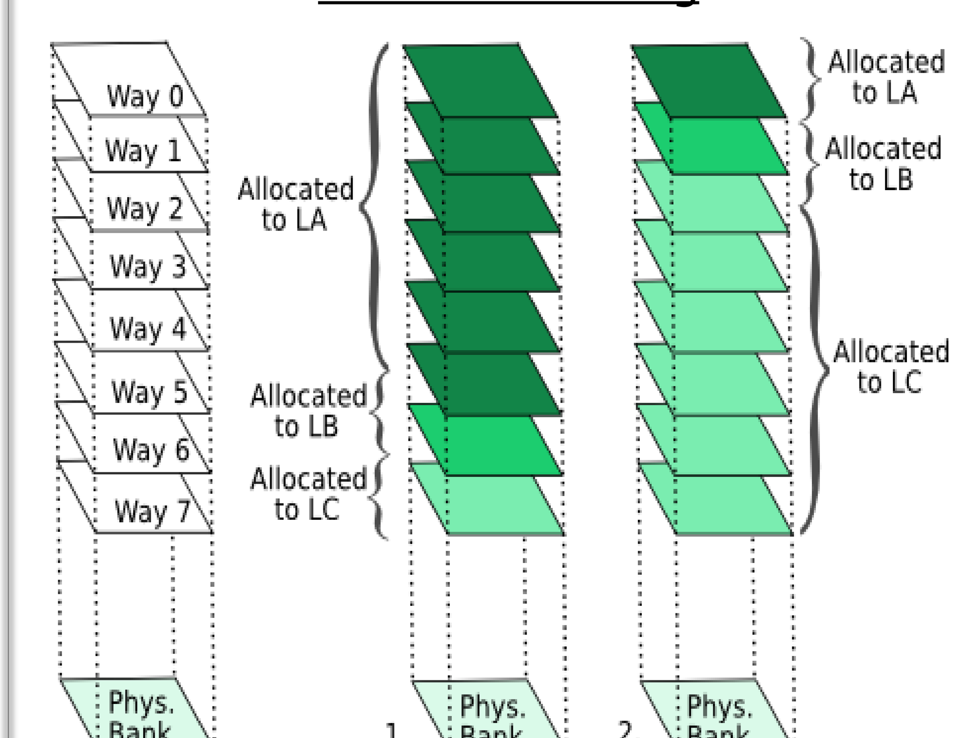
Memory Hierarchy Bandwidth Partitioning



- Globally Synchronized Frames (GSF)
  - A frame-based QoS System
- An allocation of flits are guaranteed to each core per frame (time window)
- Excess flits in a frame are shared

Reference: J. W. Lee et al. "Globally Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks," ISCA 2008

Way- and Bank-Based Cache Partitioning



- Two types of cache partitioning allow for a wide variety of configurations
- Applications can be assigned cache slices – particular ways in a given bank
- Cache slices can be reassigned to represent the changing needs of the system

5. Implementation Status

Kernel	User-level Runtime Support	Network Service (including TCP/IP stack)
~35K LOC	~10K LOC	~40K LOC

- Partitioning support
  - Cores, caches (via page coloring), and memory bandwidth partitioning (on RAMP simulator)
- Inter-cell channels (via ring buffers in shared memory)
  - Hardware channels implementation currently under development on RAMP simulator
- User-level frameworks for implementing
  - Composable cooperative schedulers (i.e., Lithe)
  - Preemptive schedulers (e.g., EDF)
- Basic Services
  - Network Service consisting of a device driver and TCP/IP stack
  - File Service, GUI Service, and Policy Service are under development
- Gang Scheduling for Cells
  - Implemented a communication-free version and a centralized version
- Currently two ports
  - Intel x86 platforms (e.g., 32-core Nehalem system)
  - FPGA-based simulation of 64 1-GHz SPARC V8 cores (RAMP Gold)
- Current prototype was derived from an early version of Akaros (http://akaros.cs.berkeley.edu)

Research supported by Microsoft Award #024263, Intel Award #024894, matching U.C. Discovery funding (Award #DIG07-102270), and DOE ASCR FastOS Grant #DE-FG02-08ER25849