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

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

Component-based Model with Composable Resources

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

4. Resource-management Software Architecture

- Cell Creation and Resizing Request Service
- Admission Control
- Major Change Requests
- Resource Allocation
- Global Policies
- Resource Allocation and Admission Mechanism
- Resource Management Policy Framework
- Online Performance Monitoring, Mode Building, and Prediction
- Tesselation OS
- User/OS Interface
- Cell Performance Monitoring
- Partition Mapping
- Resource Monitoring
- Resource Management
- Partitioning

5. Implementation Status

- 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 (e.g., 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 54 1-GHz SPARC V8 cores (RAMP Gold)
- Current prototype was derived from an early version of Akaros (http://akaros.cs.berkeley.edu)

Hardware 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

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

Hardware-acceleration for Inter-cell Channels

- Improves efficiency of inter-cell communication

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


Network Service

- ~35K LOC
- ~10K LOC
- ~40K LOC

Research supported by Microsoft Award #024653, Intel Award #024894, matching U.C. Discovery funding (Award #00G07-10270), and DOE ASCR FastOS Grant #DE-FG02-08ER25849

Tessellation Operating System

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

Juan A. Colmenares, Sarah Bird, Gage Eads, Steven Hofmeyst, Albert Kim, Rohit Poddar, Hilfi Alkaft, Krste Asanovic, and John Kubiatowicz

http://tessellation.cs.berkeley.edu