Tessellation: Space-Time Partitioning in a Manycore Client OS

Rose Liu\textsuperscript{1,2}, Kevin Klues\textsuperscript{1}, Sarah Bird\textsuperscript{1}, Steven Hofmeyr\textsuperscript{3}, Krste Asanovic\textsuperscript{1}, John Kubiatowicz\textsuperscript{1}

\textsuperscript{1}Parallel Computing Laboratory, UC Berkeley
\textsuperscript{2}Data Domain
\textsuperscript{3}Lawrence Berkeley National Laboratory
Client Device

- Single-user device
- Runs a heterogeneous mix of interactive, real-time and batch applications simultaneously
- Generally battery constrained
Why a new Client OS?

- Enter the Manycore world → Must address parallelism
  - Current client OSs weren’t designed for parallel applications
- Existing OSs addressing parallelism targets servers or HPC contexts, not clients
  - Servers – emphasis on throughput vs. Client – emphasis on user experience/responsiveness
  - HPC – machine dedicated to one parallel application vs. Client – runs many heterogeneous parallel applications
  - Client - Longer battery life
Outline

- Why a new OS for Manycore Clients?
- A Case for Space-time Partitioning
  - Define space-time partitioning
  - Use cases for space-time partitioning
- Implementing Space-Time Partitioning in a Manycore OS
- Status
Spatial Partitions

Isolated unit containing a subset of physical machine resources
Spatial Partitions

Isolated unit containing a subset of physical machine resources
Spatial Partitions

Isolated unit containing a subset of physical machine resources
Spatial Partitions

QoS enforced
share of
interconnect
bandwidth

Isolated unit containing a subset of physical machine resources
Spatial Partitions

Energy or Power Budget

Isolated unit containing a subset of physical machine resources
Machine divided into spatial partitions

Wireless radio

Memory
Put applications in spatial partitions

- Media Player
- Browser
- Memory

Radio
Benefits of spatial partitions

- Each app can run a custom user-level runtime for best performance
- Provides apps with resource guarantees for performance predictability
- Functional & Performance Isolation
  - Natural unit for fault containment, energy management
Put OS Services in spatial partitions

- Memory
- Network Driver
- Media Player
- Browser
- Filesystem

Wireless radio
Put sub-components in spatial partitions

- Media Player
- Network Driver
- Video decoder
- GUI
- Browser
- Filesystem
- Memory
- Wireless radio
Put virtual machines in spatial partitions
Partitions need to communicate

Spatial Communication occurs without a context switch
Communication Challenges

Communication relaxes the isolation boundaries of partitions and introduces issues like:

- Security
- Service-level QoS and/or resource accounting of requestors within service partitions

![Diagram of communication components and relationships](image-url)
Space-time partitioning virtualizes spatial partitions

- Partition Context Switch Cost ~ Process Context Switch Cost
- Time multiplex at a coarse granularity to allow for user-level scheduling
Space-Time Partition Scheduling

Descheduled Partitions:

Real-time app is always scheduled

Partition resources put in low power state

Partitions are dynamically resized while running without a reboot or application restart
Space-Time Partition Scheduling

**Descheduled Partitions:**

**Challenges:**
1. How to determine the right resource allocation for a partition?
2. What granularity to time multiplex each partition? Don’t need to use same time quanta for all partitions.
3. We can deschedule partitions from each type of resource independently. E.g. time multiplex off cores more frequently than multiplex partition data off caches. How to determine ‘best’ policy?

**Time**

Partitions are dynamically resized while running without a reboot or application restart
Communication in space and time

Wireless radio

Media Player → Network Driver

Video decoder → GU

Browser → Windows VM

FileSystem → De-scheduled Partitions

Memory

GUI
Outline

- Why a new OS for Manycore Clients?
- A Case for Space-time Partitioning
- Implementing Space-Time Partitioning in a Manycore OS (Tessellation)
- Status
Tessellation Kernel

Application
Or
OS Service

Library OS Functionality

Custom Scheduler

Interconnect Bandwidth  Message Passing  Cache  Physical Memory  CPUs  Performance Counters

Hardware Partitioning Mechanisms
Tessellation Kernel

Application
Or
OS Service

- Library OS Functionality
- Custom Scheduler

Marshalls syscalls into messages for the respective OS Service Partition

App-specific scheduler for best parallel performance. (See Lithe talk on user-level scheduling.)

Hardware Partitioning Mechanisms

- Interconnect Bandwidth
- Message Passing
- Cache
- Physical Memory
- CPUs
- Performance Counters
Tessellation Kernel

Hardware Partitioning Mechanisms

- Interconnect Bandwidth
- Message Passing
- Cache
- Physical Memory
- CPUs
- Performance Counters

Application
Or
OS Service

Library OS
Functionality

Custom
Scheduler
Tessellation Kernel

Partition Management Layer
Partition Mechanism Layer (Trusted)

Hardware Partitioning Mechanisms
- Interconnect Bandwidth
- Message Passing
- Cache
- Physical Memory
- CPUs
- Performance Counters

Application
Or
OS Service

Custom Scheduler

Library OS Functionality

CPUs
Physical Memory
Cache
Message Passing
Interconnect Bandwidth
Performance Counters

Tessellation Kernel
Partition Mechanism Layer

Application
Or
OS Service

Library OS Functionality

Custom Scheduler

Partition Management Layer

Partition Mechanism Layer (Trusted)

Configure Partition Resources enforced by HW at runtime

Interconnect Bandwidth
Message Passing
Cache
Physical Memory
CPUs
Performance Counters

Tessellation
Kernel

Hardware Partitioning Mechanisms
Partition Mechanism Layer

Application

Or

OS Service

Library OS Functionality

Custom Scheduler

Partition Management Layer

Partition Mechanism Layer (Trusted)

Configure Partition Resources enforced by HW at runtime

Configure HW-supported Communication

Interconnect Bandwidth

Message Passing

Cache

Physical Memory

CPUs

Performance Counters

Hardware Partitioning Mechanisms
Partition Management Layer

- Application Or OS Service
  - Library OS Functionality
  - Custom Scheduler

Hardware Partitioning Mechanisms
- Interconnect Bandwidth
- Message Passing
- Cache
- Physical Memory
- CPUs
- Performance Counters

Partition Management Layer
- Partition Mechanism Layer (Trusted)
  - Configure Partition Resources enforced by HW at runtime
  - Configure HW-supported Communication

Partition Allocator

Tessellation
Partition Management Layer

Application
Or
OS Service

Partition Allocator

Partition Resizing
Callback API

Partition
Management
Layer

Partition
Mechanism
Layer (Trusted)

Configure Partition
Resources enforced by
HW at runtime

Configure
HW-supported
Communication

Interconnect
Bandwidth
Message
Passing
Cache
Physical
Memory
CPUs
Performance
Counters

Hardware Partitioning Mechanisms

Library OS
Functionality

Custom
Scheduler

Tessellation

Trusted

Partition Management Layer

Application
Or
OS Service

Library OS
Functionality

Custom
Scheduler

Comm
Reqs

Res. Reqns.

Partition
Resizing
Callback API

Partition
Scheduler

Partition
Allocator

Configure Partition
Resources enforced by
HW at runtime

Configure
HW-supported
Communication

Partition
Mechanism
Layer
(Trusted)

Partition
Management
Layer

Interconnect
Bandwidth
Message
Passing
Cache
Physical
Memory
CPUs
Performance
Counters

Hardware Partitioning Mechanisms
Partition Management Layer

- Library OS Functionality
- Application Or OS Service
- Custom Scheduler

Partition Management Layer

- Partition Scheduler
- Partition Allocator

Partition Mechanism Layer (Trusted)

- Configure Partition Resources enforced by HW at runtime
- Configure HW-supported Communication

Hardware Partitioning Mechanisms

- Interconnect Bandwidth
- Message Passing
- Cache
- Physical Memory
- CPUs
- Performance Counters

Tessellation

Partition Resizing Callback API
Outline

- Why a new OS for Manycore Clients?
- A Case for Space-time Partitioning
  - Define space-time partitioning
  - Use cases for space-time partitioning
- Implementing Space-Time Partitioning in a Manycore OS
- Status
Implementation status

- Basics of Tessellation kernel and primitive OS service up and running
  - Provides rudimentary partition interface
  - Boots on standard x86 hardware
  - No I/O yet – statically linked applications and kernel
Next Steps

- Build fast cross-core communication mechanisms for system calls
  - Context-switch free system calls
  - APIC driven message notification with shared memory
- Add support for the 19 newLib system calls in TOS OS Service partition
Intermediate Infrastructure

- TOS OS Service doesn’t have all drivers, so run BSD with existing drivers on one core to service I/O from TOS OS Service
- Tessellation runs on rest of the cores
Acknowledgements

- We would like to thank the entire ParLab OS group and several people at LBNL for their invaluable contribution to the ideas presented here.

- Research supported by Microsoft Award #024263 and Intel Award #024894 and by matching funding from U.C. Discovery (Award #DIG07-102270).
- This work has also been in part supported by an NSF Graduate Research Fellowship.
Thanks! Questions?

Diagram:
- Media Player
- Network Driver
- Video decoder
- GUI
- Browser
- Windows VM
- Filesystem
- Memory
- Wireless radio
- De-scheduled Partitions