Refactoring the OS around Explicit Resource Containers with Continuous Adaptation

Operating Systems Research in the Par Lab

The OS Group
Par Lab, UC Berkeley
http://tessellation.cs.berkeley.edu

Presented by Juan A. Colmenares, Gage Eads and Sarah Bird at the End of the Par Lab Symposium
May 30, 2013
Berkeley, CA
Acknowledgment

• Research supported by
  – Intel (Award #024894)
  – Microsoft (Award #024263)
  – U.C. Discovery funding (Award #DIG07-102270)

• Additional support from Par Lab affiliates
  – National Instruments, NEC, Nokia, NVIDIA, Samsung, Sun Microsystems

Disclaimer
No part of this presentation necessarily represents the views and opinions of the aforementioned sponsors
Team and Collaborators

- Hilfi Alkaff (UCB)
- Krste Asanović (UCB Faculty)
- Rimas Avižienis (UCB)
- Davide Bartolini (Politecnico di Milano / UCB)
- Eric Battenberg (UCB)
- Sarah Bird (UCB)
- David Chou (UCB)
- Juan Colmenares (UCB/Samsung)
- Henry Cook (UCB)
- Gage Eads (UCB)
- Brian Gluzman (UCB)
- Ben Hindman (UCB)
- Steven Hofmeyr (LBL)
- Eduardo Huerta (ICSI)
- Costin Iancu (LBL)
- Israel Jacques (UCB)
- John Kubiatowicz (Lead, UCB Faculty)
- Albert Kim (UCB)
- Kevin Klues (UCB)
- Akihito Kohiga (NEC)
- Eric Love (UCB)
- Rose Liu (MIT)
- Miquel Moretó (UCB/UPC)
- Nitesh Mor (UCB)
- Paul Pearce (UCB)
- Heidi Pan (MIT/Intel)
- Nils Peters (UCB/Qualcomm)
- Barret Rhoden (UCB)
- Eric Roman (LBL)
- Ian Saxton (UCB)
- John Shalf (LBL)
- Burton Smith (MSR)
- Andrew Waterman (USB)
- David Wessel (UCB Faculty)
- David Zhu (UCB)
At The Beginning: A.D. 2008

Many-core trend

Mixed workloads on client devices

High throughput parallel apps

Interactive apps

Real-time apps

Users’ expectations increase with core count

You got more, then

Gimme More!
Common OS Anecdotes

- Screen freezes when running a heavy compile job
- Video chat becomes choppy when a local app starts
- Impossible for me to watch a video with a scientific simulation in the background
- Hey OS, what can I do? Sometimes it goes ...
Can We Solve Those Problems? Can We Reinvent the OS to ... ?

• Take advantage of many-core platforms
• Properly serve simultaneous applications of different types and with conflicting requirements
• Meet users’ expectations about performance

We said: “Yes, we can!”

and started

Tessellation OS, Lithe, and PACORA
Goals in Tessellation OS

• Support a dynamic mix of high-throughput, parallel, interactive, and real-time applications

• Allow applications to deliver guaranteed or at least consistent performance

• Enable adaptation to changes in the application mix and resource availability
Focus on Resources to Provide Performance Guarantees

• What do we want to guarantee?
  – Throughput (e.g., requests/sec)
  – Latency to response (e.g., service time)
  – Others: energy/power budget

• What type of guarantees?
  – Probabilistic with high confidence

• The “impedance-mismatch” problem
  – Service Level Agreements (SLAs) indicate properties that programmer/user wants
  – The resources required to satisfy the SLA are not things that programmer/user really understands
Adaptive Resource-Centric Computing (ARCC) [DAC’13]

Resource Allocation (Control Plane)

Partitioning and Distribution

Observation and Modeling

Performance Reports

Resource Assignments

Running System (Data Plane)

Application1

GUI Service
QoS-aware Scheduler

Block Service
QoS-aware Scheduler

Network Service
QoS-aware Scheduler

Application2

Channel

Cell

Channel

Cell

Channel

Cell

Channel
Adaptive Resource-Centric Computing (ARCC)

**Cells: Performance-Isolated Resource Containers**
- Provide guaranteed access to assigned resources
- Give full user-level control of the resources

- **Running System (Data Plane)**
  - Application 1
  - Application 2

- **Cell**
  - QoS-aware Scheduler
  - Block Service
  - Network Service

- **Partitioning and Distribution**
- **Observation and Modeling**
- **Resource Allocation (Control Plane)**
- **Resource Assignments**
- **Performance Reports**
- **Adaptive Resource-Centric Computing (ARCC)**
- **Cells: Performance-Isolated Resource Containers**
  - Provide guaranteed access to assigned resources
  - Give full user-level control of the resources

- **Running System (Data Plane)**
  - Application 1
  - Application 2

- **Cell**
  - QoS-aware Scheduler
  - Block Service
  - Network Service

- **Partitioning and Distribution**
- **Observation and Modeling**
- **Resource Allocation (Control Plane)**
- **Resource Assignments**
- **Performance Reports**
- **Adaptive Resource-Centric Computing (ARCC)**
- **Cells: Performance-Isolated Resource Containers**
  - Provide guaranteed access to assigned resources
  - Give full user-level control of the resources
Adaptive Resource-Centric Computing (ARCC)

Customizable User-level Runtimes in Cells

- To best meet applications’ needs

Running System (Data Plane)

Application 1

Cell

GUI Service

Block Service

Network Service

QoS-aware Scheduler

QoS-aware Scheduler

QoS-aware Scheduler

Cell

Application 2

Resource Allocation (Control Plane)

Resource Assignments

Performance Reports

Partitioning and Distribution

Observation and Modeling

Adaptive Resource-Centric Computing (ARCC)

Cell

Customizable User-level Runtimes in Cells

- To best meet applications’ needs
Adaptive Resource-Centric Computing (ARCC)

**OS Services with QoS Guarantees**
- Reside in dedicated cells, have exclusive control over devices, and arbitrate access to them

- **Running System (Data Plane)**
  - **Application 1**
    - **GUI Service**
      - QoS-aware Scheduler
    - **Block Service**
      - QoS-aware Scheduler
  - **Cell**
  - **Application 2**
    - **Network Service**
      - QoS-aware Scheduler
    - **Cell**

- **Resource Allocation (Control Plane)**
- **Resource Assignments**
- **Performance Reports**
Adaptive Resource-Centric Computing (ARCC)

Resource Allocation (Control Plane)

Partitioning and Distribution

Observation and Modeling

Performance Reports

Running System (Data Plane)

Resource Assignments

Adaptive Resource Allocation

• Automatically discovers the mix of resource assignments that maximizes overall system utility
Two-Level Scheduling

split

Monolithic CPU and Resource Scheduling
into two pieces

- Chunks of resources distributed to applications (Global Decisions)

**Level 1**
Coarse-grained Resource Allocation and Distribution

**Level 2**
Fine-grained Application-specific Scheduling

- Apps use their resources in any way they see fit (Local Decisions)
Space-Time Partitioning

Spatial Partition
• Key for performance isolation

Spatial partitioning is not static and may vary over time
• Partitions can be time *multiplexed*; resources are *gang-scheduled*
• Partitioning *adapts* to system’s needs

• Each partition receives a vector of basic resources
• A partition may also receive
  – Exclusive access to other resources (e.g., a device)
  – Guaranteed fractional services from other partitions
The Cell: Our Partitioning Abstraction
User-level Software Container with Guaranteed Access to Resources

- Full control over resources it owns when mapped to hardware
- Resources exported to user-level
- Adaptive user-level runtimes
- Efficient inter-cell communication channels

Cell A

Address Space A

Address Space B

2nd-level Scheduling

2nd-level Mem Mgmt

Channel

Cell B

Yellow partition grows due to adaptation

Space

Time
• Applications = Set of interacting components deployed on different cells
  – Applications split into performance-incompatible and mutually distrusting cells with controlled communication
  – OS Services are independent servers that provide QoS
Customizable User-Level Runtimes

Lithe: A framework for hierarchical cooperative user-level schedulers [PLDI’10]

- Non-preemptive scheduling
- Key abstraction
  - Hardware threads (harts)
  - No oversubscription!
- Enables efficient composition of parallel libraries
- http://lithe.eecs.berkeley.edu

[Tessellation Kernel]

Application

Library A

Library B

Sched 1

Sched 2

Sched 3

Lithe Runtime

[Partition Support]

Hardware cores

Customizable User-Level Runtimes

**PULSE**: A framework for Preemptive User-Level SchEdulers

• Available preemptive schedulers
  – Round-robin (and pthreads)
  – EDF and Fixed Priority
  – Multiprocessor Constant Bandwidth Server (M-CBS) [ECRTS'04]
  – Juggle: A load balancer for SPMD applications [CLUSTER'12]

• Able to handle cell resizing

GUI Service [CATA’ 12]

An OS Service with QoS Guarantees

- Exploits task parallelism for improved service times
- Provides differentiated service to applications and soft service-time guarantees

Nano-X vs. GUI Service

Service times for 4 30-fps video players and 4 60-fps video players, each sending 1000 expensive requests

Each bar represents 4 video clients. Above each bar is the total number of deadlines missed for the group.

(#): Allocated hardware threads
Network Service \[\text{[DAC’13, JAES’13]}\]
An OS Service with QoS Guarantees

• Supports reservations and proportional share of bandwidth
  – Using mClock scheduling algorithm \[\text{[OSDI’10]}\] (on top of PULSE)
• NIC driver is entirely contained in user-space
  – No system calls when transmitting and receiving buffers

![Graph showing foreground and background throughput](image)

(Avg. throughput = 125.2 KB/s)

\[\text{OSDI’10}\] A. Gulati et al. mClock: handling throughput variability for hypervisor IO scheduling.
Adaptive Resource Allocation in Tessellation OS

DAC'13 J.A. Colmenares, G. Eads et al. Tessellation: Refactoring the OS around explicit resource containers with continuous adaptation.
Continuously minimize the penalty of the system (subject to restrictions on the total amount of resources)

Known Facts and Lessons Learned

- [KF] Implementing an OS from scratch is challenging
- [KF] Supporting IO devices is very important
- [LL] Tessellation’s structuring redistributes complexity
  - Lot of complexity moved from the kernel to user-level runtimes
  - Contending factor: overhead
- [LL] Having a simple kernel is very beneficial
  - Easier to reason about it, especially when providing performance guarantees
- [LL] Coordination between kernel and cell’s user-level runtime is tricky (e.g., during cell resizing)
  - Not many LOCs, but very subtle issues and difficult to debug
Summary

• **Challenge**: Reinventing the OS for many-core platforms
  – Properly serve simultaneous applications of different types
  – Meet users’ performance expectations

• **Approach**: Adaptive Resource-Centric Computing
  – Focuses on Space-Time Partitioning and Two-Level Scheduling
  – Includes
    • Cells: Resource containers
    • Customizable user-level runtimes
    • OS services with QoS guarantees
    • Adaptive resource allocation

• **Implementation**: Tess OS, Lithe, PULSE, and PACORA

• **Effectiveness**: Demonstrated in publications and demos!
Demos on Tessellation OS

• Adaptive Resource Centric Computing
  – Entirely on Tessellation

• Live Musical Performance
  – A synthesizer performing parallel real-time audio processing and controlled via the SLABS multi-touch interface

• Million Song Recommendation (Pardora) System
  – Specialized code on top of TBB/Lithe, plus python code

• Virtual Instrument
  – One of the backends
Next

• Demonstration of Adaptive Resource Centric Computing
  – Gage Eads and Sarah Bird, UC Berkeley

• Testimonial
  – Dave Probert, Microsoft
Adaptive Resource Centric Computing Demonstration
Adaptive Resource Centric Computing Demonstration

Network (QoS from network service)

- Bandwidth Hog
  - TCP client application
  - Single-threaded
  - **Goal:** consume maximum bandwidth

- Video Players (1 vid per thread)
  - Receives video stream from TCP connection
  - 2 video sizes
  - Performs H.264 decoding
  - **Goal:** 30 FPS

Cores (QoS from kernel)

- psearchy
  - Parallel file indexing benchmark
  - Scalable work-queue based parallelism
  - **Goal:** maximize indexing throughput
Come and see our demos!

Questions?

THANKS
Gang Scheduling in Tessellation

- No need of inter-core communication (in the common case) due to use of **synchronized clocks**
- Different time-multiplexing policies for cells

![Diagram showing gang scheduling algorithms](image)

- Identical or Consistent Schedules