Decoupling Cores, Kernels and Operating Systems

Gerd Zellweger, Simon Gerber, Kornilios Kourtis, Timothy Roscoe
Systems Group, ETH Zürich
Outline

- Motivation
  Trends in hardware and software

- Booting and shutting down cores dynamically
  Decoupling the kernel state

- Evaluation
  Kernel updates, specialized kernels
What’s happening to hardware

- Constrained by power consumption

- Reconfigurable cores (dynamically changed behavior)
  - DVFS, Turbo Boost, SMT
  - Core Fusion [ISCA ‘07]
  - Dark silicon [ISCA ‘10]

- Heterogeneous cores
  - Fast and power hungry vs. slow and power efficient
  - Asymmetric multiprocessing
  - Conservation Cores [ASPLOS ‘10]
What current operating systems look like
What current operating systems look like
What current operating systems look like
What’s happening to software

- OS needs to adapt to different workloads

- Adapting at build-, boot-, and run-time
  - Debugging support: profiling, tracing etc.
  - Real-time support

- On-the-fly kernel updates
  - KSplice (Linux) [EuroSys ‘09]
  - K42 [ATC ‘07]
Multikernel [SOSP ‘09]
Implementation

- Barrelfish OS

- Treating cores as pluggable devices
  - Booting a core dynamically with boot drivers
  - Shutting down a core

- Decoupling Cores, Kernels and the Operating System
  Externalizing kernel state
Booting a core with boot drivers

- OS service for target core management
- Dynamically chooses kernel for core based on runtime information
  - Boots any core with any suitable kernel
  - Run any OSNode on any compatible core
- Implements boot, shutdown, reboot protocol
Shutting down a core

- Harder than booting a core
  - Need to deal with per-core state: Scheduler queues, memory pools, page-tables…
  - Takes time (and energy)
- However, we want to remove the core as fast as possible
- General approach (cf. Chameleon [ASPLOS ’12])
  - Get state out of the way quickly
  - Dismantle it later, lazily (if needed)
Shutting down a core
Shutting down a core
Shutting down a core

Application
OSNode
Kernel
CPU

Application
OSNode
Kernel
CPU

Application
OSNode
OSNode
Kernel
CPU
Shutting down a core
Shutting down a core
Shutting down a core

Highly scalable, only two cores involved
What is the OSNode?

**OSNode:** All state for a single core and kernel

How do we capture this OSNode?

- **Capabilities:**
  - Tracks all application state
  - Tracks all OS state
  cf. seL4, EROS, KeyKOS
- **KCB (Kernel control block)**
  - Hardware specific state
  - Entry point to capability tree
  - Represented as a capability itself
Decoupling Cores, Kernels and Operating Systems

State externalization & dynamic core booting is a much more general mechanism
Decoupling Cores, Kernels and Operating Systems

State externalization & dynamic core booting is a much more general mechanism
Decoupling Cores, Kernels and Operating Systems

State externalization & dynamic core booting is a much more general mechanism
Decoupling Cores, Kernels and Operating Systems

State externalization & dynamic core booting is a much more general mechanism
Decoupling Cores, Kernels and Operating Systems

State externalization & dynamic core booting is a much more general mechanism
Evaluation

- Core management
  Adding and removing cores in the system

- Kernel updates
  Hot-swapping the kernel

- Specialized kernels
  e.g., eliminate OS jitter
## Core management (Haswell, 1x4 cores, no HT)

<table>
<thead>
<tr>
<th>Booting a core</th>
<th>No Load</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux 3.13</td>
<td>14 ms</td>
<td>20 ms</td>
</tr>
<tr>
<td>Barrelfish/DC</td>
<td>7.5 ms</td>
<td>7.5 ms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Removing a core</th>
<th>No Load</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux 3.13</td>
<td>46 ms</td>
<td>2542 ms</td>
</tr>
<tr>
<td>Barrelfish/DC</td>
<td>0.0008 ms</td>
<td>0.0008 ms</td>
</tr>
</tbody>
</table>
Use-case: Kernel Updates

- Shut-down target core
Use-case: Kernel Updates

- Shut-down target core
- Reboot core with a new kernel image
Use-case: Kernel Updates

- Shut-down target core
- Reboot core with a new kernel image
- Dispatch previous OSNode
Kernel updates: PostgreSQL & TPC-H

Latency of Query [ms]

Time [Query]
Use-case: Temporary real time task

- A thread that needs to run with hard real time performance
  - E.g., phone baseband stack, control application, robotics etc.
- A lot of effort spent to make this work in a general purpose OS
- Many real time OS for embedded systems (RTLinux, LynxOS, QNX, …)
Use-case: Real time application
Use-case: Real time application

RT Kernel

OS

OS
Use-case: Real time application
Use-case: Specialized kernels

- Shut-down target core
Use-case: Specialized kernels

- Shut-down target core
- Temporarily park the target OSNode
Evaluation: PostgreSQL & TPC-H

- One OSNode per core
- Parked OSNode (2 OSNodes per core)
- Move OSNode back to original core
Use-case: Specialized kernels

- Shut-down target core
- Temporarily park the target OSNode
- Boot simple real-time kernel that runs just one application
  - Does not take interrupts
  - No timers
  - No scheduler
- Temporarily provides task with hard real-time guarantees
Almost all samples between 6-7k

Outliers (OS jitter)

# cycles for 1k memory stores

No samples outside of 6-7k range
Future Work & Applications

- Transfer OSNodes between power efficient and high performance cores
- Dynamic OS instrumentation
  - Profiling, tracing kernels
- A/B kernel testing
- Specialized kernel to run applications in guest ring 0
cf. Arrakis
Decoupling the kernel state

Result: highly dynamic OS architecture

- Kernels can be rebooted, updated and specialized
- Cores can be allocated and de-allocated arbitrarily
- For many versions of the “dark silicon” hardware, this may be the only way for system software

Conclusion
Backup
Dealing with interrupts

1. Timers, etc. local to core and CPU driver
   - Handled internally to CPU driver
2. Inter-processor interrupts (IPIs)
   - Indirection table of OSNodes → physical cores
3. Device interrupts
   - Must be re-routed to new core
Device interrupts

Device driver

kernel 1

core 1

IRQh

vectors

IOAPIC

core 2
Device interrupts
Device interrupts

New kernel initializes

Device driver

core 1

vectors

core 2

vectors

IOAPIC

IRQh

kernel 2
Device interrupts

"Register interrupts" message

Device driver

kernel 2

core 2

IRQh

vectors

core 1

vectors

IOAPIC
Device interrupts

- Device driver
- IRQh
- kernel 2
- vectors
- core 2
- Interrupt rerouted
- core 1
- IOAPIC