New directions in OS design: the Barrelfish research operating system

Timothy Roscoe (Mothy)
ETH Zurich
Switzerland
Acknowledgements

• ETH Zurich:
  – Zachary Anderson, Pierre-Evariste Dagand, Stefan Kästle, Dominik Menzi, Simon Peter, Kaveh Razavi, Jan Rellermeyer, Timothy Roscoe, Bram Scheidegger, Raffaele Sandrini, Adrian Schüpbach, Pravin Shinde, Dario Simone, Akhilesh Singhaniya, Animesh Trivedi

• Microsoft Research:
  – Paul Barham, Andrew Baumann, Richard Black, Tim Harris, Orion Hodson, Rebecca Isaacs, Ross McIlroy, Vijayan Prabhakaran

• And many other friends and collaborators...
This talk...

- **Context**
  - Trends in hardware
- **Rethinking OS design**
  - The Multikernel
  - The System Knowledge Base
- **Barrelfish design**
  - Overview
  - Messaging
  - Distributed algorithms
  - Configuring hardware
- **Ongoing work, goals, and Conclusion**
CONTEXT
A general purpose OS

• Mix of untrusted applications
• Soft real-time requirements for some
• Variety of hardware platforms
Lots more cores per chip

- Core counts now follow Moore’s Law
- Cores will come and go
  - Energy!
- Diversity of system and processor configurations will grow
- Cache coherence may not scale to whole machine
Cores will be heterogeneous

- NUMA is the norm today
- Heterogeneous cores for power reduction
- Integrated GPUs / Crypto / NPUs etc.
- Programmable peripherals
Communication latency really matters

Example: 8 * quad-core AMD Opteron

<table>
<thead>
<tr>
<th>Access</th>
<th>cycles</th>
<th>normalized to L1</th>
<th>per-hop cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache</td>
<td>2</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>L2 cache</td>
<td>15</td>
<td>7.5</td>
<td>-</td>
</tr>
<tr>
<td>L3 cache</td>
<td>75</td>
<td>37.5</td>
<td>-</td>
</tr>
<tr>
<td>Other L1/L2</td>
<td>130</td>
<td>65</td>
<td>-</td>
</tr>
<tr>
<td>1-hop cache</td>
<td>190</td>
<td>95</td>
<td>60</td>
</tr>
<tr>
<td>2-hop cache</td>
<td>260</td>
<td>130</td>
<td>70</td>
</tr>
</tbody>
</table>
Implications

• Computers are systems of cores and other devices which:
  – Are connected by highly complex interconnects
  – Entail significant communication latency between nodes
  – Consist of heterogeneous cores
  – Show unpredictable diversity of system configurations
  – Have dynamic core set membership
  – Provide only limited shared memory or cache coherence

The OS model of cooperating processes over a shared-memory multithreaded kernel is dead.
RETHINKING OS DESIGN #1: 
THE MULTIKERNEL ARCHITECTURE
The Multikernel Architecture

• Computers are systems of cores and other devices which:
  – Are connected by highly complex interconnects
  – Entail significant communication latency between nodes
  – Consist of heterogeneous cores
  – Show unpredictable diversity of system configurations
  – Have dynamic core set membership
  – Provide only limited shared memory or cache coherence

⇒ Forget about shared memory.
The OS is a distributed system based on message passing
Multikernel principles

• Share no data between cores
  – All inter-core communication is via explicit messages
  – Each core can have its own implementation

• OS state partitioned if possible, replicated if not
  – State is accessed as if it were a local replica

• Invariants enforced by distributed algorithms, not locks
  – Many operations become split-phase and asynchronous
The multikernel model

User space:
- App
- App
- Application
- Application

Operating System:
- OS node state replica
- OS node state replica
- OS node state replica
- OS node state replica

Arch-specific code:
- Operating System
- Hardware
- x86_64 CPU
- X86_64 CPU
- ARM NIC
- ... GPU w/ CPU features

Interconnect(s)
...vs a monolithic OS on multicore

App

kernel

Interconnect

Main memory
holds global data structures

x86
x86
x86
x86

11/11/2011
...vs a μkernel OS on multicore
Replication vs sharing as the default

- Replicas used as an optimization in other systems
- In a multikernel, sharing is a local optimisation
  - Shared (locked) replica on closely-coupled cores
  - Only when faster, as decided at runtime
- Basic model remains split-phase messaging
RETHINKING OS DESIGN #2: 
THE SYSTEM KNOWLEDGE BASE
System knowledge base

- Computers are systems of cores and other devices which:
  - Are connected by highly complex interconnects
  - Entail significant communication latency between nodes
  - Consist of heterogeneous cores
  - Show unpredictable diversity of system configurations
  - Have dynamic core set membership
  - Provide only limited shared memory or cache coherence

⇒ Give the OS advanced reasoning techniques to make sense of the hardware and workload at runtime.
What goes in?

1. Resource discovery
   - E.g. PCI enumeration, ACPI, CPUID...
2. Online hardware profiling
   - Inter-core all-pairs latency, cache measurements...
3. Operating system state
   - Locks, process placement, etc.
4. “Things we just know”
   - Assertions from data sheets, etc.
What comes out?

OS and applications submit high-level queries:
- Relational-style queries
- Logic programming
- Satisfiability modulo theories
- Linear, integer programming
- Etc.

• SKB returns results for policies, optimization, etc.
  - Examples later
BARRELFISH
Barrelfish: our multikernel

• ETH Zurich + Microsoft Research
• Currently supports:
  – 32- and 64-bit x86 AMD/Intel
  – Intel Single-chip Cloud Computer
  – Intel MIC (Knight’s Ferry)
  – ARM (& Xscale)
  – Beehive (experimental softcore)
• Published 2009, available now
  – MIT open source licence, www.barrelfish.org
What things run on it?

• Many microbenchmarks
• Parallel benchmarks: Parsec, SPLASH-2, NAS
• Webserver: http://www.barrelfish.org/
• Databases: SQLite, PostgreSQL
• Virtual machine monitor
  – Linux kernel binary
• Microsoft Office 2010!
  – via Drawbridge
Per-core architecture

- Representative of app. on each core (including drivers and services)
- Upcalled from CPU driver
- Local thread scheduling
- Communicates with peer dispatchers

- User space (extra privilege)
- Communicates with other monitors
- Manages distributed operations
- Performs long-running operations

- Kernel space
- Manages CPU, MMU, APIC
- Multiplexes core btw. dispatchers
- Handles interrupts
- Implements protection via capability validation
- Single-threaded, non-preemptable

Monitor

Application (dispatcher)

Application (dispatcher)

CPU driver

CPU
System Knowledge Base

• Port of popular ECLiPse CLP system
  – Constraint Logic Programming
    • not the IDE!
  – Very expressive (Prolog + constraints)
  – Quite slow – fairly old technology (vs., e.g. Z3)
  – But easy to hack for a prototype

• Starts very early in boot process
  – Initially runs from its own RAMdisk
  – Pulls more files from file system later

• Used for some surprising functions...
Non-original ideas in Barrelish Techniques we liked

- Capabilities for resource management (seL4)
- Minimize shared state (Tornado, K42)
- Upcall processor dispatch (Psyche, Sched. Activations)
- Push policy into user space domains (Exokernel, Nemesis)
- User-space RPC decoupled from IPIs (URPC)
- Lots of information (Infokernel)
- Single-threaded non-preemptive kernel per core (K42)
- Run drivers in their own domains (μkernels, Xen)
- Specify device registers in a little language (Devil)
EXAMPLE DESIGN CHALLENGES AND OPPORTUNITIES
Multikernel architecture + SKB

⇒ ??

- Messaging stack
- Fast messaging passing on diverse hardware
- Group communication in a machine
- Distributed algorithms
- Hardware configuration using CLP
Communication stack

- Low-level messaging
- Highly exposed, fixed MTU
- User-space where possible
- Polled or event-based
- Typically IPI
- Adjunct to Interconnect Drivers

Group communication → Routing → Smart stubs → Interconnect drivers → Notification drivers → User-space dispatcher → CPU

Systems@ETH Zürich
Message-passing on x86

- **LMP**
  - Asynchronous
    - Deliver fixed-size message to the domain, and if necessary unblock it
    - Use shared-memory for more complex messages
  - Synchronous
    - Lightweight RPC (*Bershad et al*, *TOCS*, 8(1), Feb 1990)

- **UMP**
  - User-level RPC (*Bershad et al*, *TOCS*, 9(2), May 1991)
    - Use a region of memory as a channel
    - Transfer cache-line-sized messages (64 bytes)
  - Tailor to the cache-coherence protocol for good performance
Other interconnect drivers

• LMP: L4-like single-core IPC
• UMP: URPC-like shared-memory messaging
• SCC: High performance tile memory
• Beehive: per-core message FIFO
• PCIe: x86 ↔ SCC message passing
• Tunnelling: multi-hop routed transport
• Ethernet: cross-machine

Etc.
Message-passing

• Has to be very fast!
  – Specialize the implementation to the hardware
  – Select implementation at bind-time
• Must work between heterogeneous cores and across different machine architectures
  – Standardize the API
  – Generate code from an IDL (“Flounder”)
Communication stack

- Consensus
  - Replica maintenance

- Multihop routing
  - Multicast tree construction

- Portability layer (C API)
  - Generated from IDL

- Low-level messaging
  - Highly exposed, fixed MTU
  - User-space where possible
  - Polled or event-based

- Typically IPI
  - Adjunct to Interconnect Drivers

Group communication
Routing
Smart stubs
Interconnect drivers
Notification drivers
User-space dispatcher
CPU driver
CPU
Examples of distributed algorithms in Barrelfish

• Keeping TLBs consistent
  – Each core holds a cache of virtual memory mappings, known as the TLB
  – Requires 1-phase commit

• Keep the capability database consistent
  – Virtual memory protection is enforced with capabilities
  – The capability database is replicated on every core
  – Requires 2-phase commit

• If cores can sleep (eg to save power), we may need a group membership protocol and more sophisticated consensus algorithms
TLB shootdown

• When a mapping changes, the TLB must be flushed
  – On a multi-core machine, the TLB of every core that might contain the mapping must be flushed
• Requires global coordination (on every OS)
  – Send a message to every core with a mapping
  – Wait for acks (must be short!)
• Linux/Windows:
  – Send IPI (interprocessor interrupt)
  – Spin on shared ack count
• Barrelfish:
  – Monitor runs 1-phase commit protocol to remote cores
  – Can exploit knowledge of interconnect topology to improve performance
Case study for TLB shootdown

Hardware: 8 * quad-core AMD Opteron

Diagram showing the arrangement of CPUs, L1, L2, and L3 caches connected through PCIe slots.
TLB shootdown: n*unicast

- **Write**
  - Cache-lines
  - ... (to blue circles)

- **Read**
  - Cache-lines
  - ... (to blue circles)
TLB shootdown: 1*broadcast

write

cache-lines

read
Messaging costs

Latency (cycles x 1000) vs Cores

Broadcast vs Unicast
TLB shootdown: multicast

Same package (shared L3)
TLB shootdown: NUMA-aware multicast

More interconnect hops

Same package (shared L3)
Aggregation tree in action
Messaging costs

![Graph showing messaging costs](image_url)

- Broadcast
- Unicast
- Multicast
- NUMA-Aware Multicast

Latency (cycles x 1000) vs. Cores
SKB query to find the lowest-latency multicast tree

multicast_tree_cost(StartCore, [SendH|SendList], Cost) :-
    multicast_sanity_check,
    % determine package of start core
    cpu_thread(StartCore, StartPackage, _, _),
    % construct list of other packages
    findall(X, (cpu_thread(_,X,_,_), X =\= StartPackage), L),
    filter(L,PackageList),
    % compute possible links to those packages as SendList1
    sends(StartCore, PackageList, SendList1),
    % compute links from start core to its neighbours
    sendNeighbours(StartCore, Neighbours),
    append(SendList1, Neighbours, SendList2),
    % annotate with RTT of each link
    annotate_rtt(SendList2, SendList3),
    % sort by decreasing RTT
    sort(3, >\=, SendList3, [SendH|SendList]),
    % determine cost as maximum single-link RTT
    % XXX: this is not quite right, we really care about maximum end-to-end RTT
    sendto(_,_,Cost) = SendH.

% goal to be called.
% Construct a list of sendto/3 goals, sort them by latency in decreasing order and
% minimize the value of the longest latency
multicast_tree(StartCore,SendList) :-
    minimize(multicast_tree_cost(StartCore, SendList, Cost), Cost).
Trace: 2PC NUMA-aware multicast

First core forwards the message to the other 3 nodes on the same package

Core 0 sends a message to a core on each remote package

Gathers the replies

Total time ~70,000 cycles
Applying distributed computing ideas to an OS

More subtle than it sounds...

• Algorithmic complexity typically measured in rounds
  – On a network, propagation time dominates
    ⇒ maximise # messages in flight

• On a single machine, propagation time appears < 0!
  – Rx and Tx overlap
  – Cost of Tx operation dominates
    ⇒ minimise # back-to-back msg Tx/Rx operations

• Optimal algorithm may be different!
  – E.g. centralized agreement hits scaling wall much faster than on a LAN.
And finally: PCIe programming: how hard can it be?

- Correct PCI bridge configuration turns out to be a nightmare!
  - Bridge/alignment constraints
  - Fixed devices
  - Devices that may or may not be PCI devices
  - Holes in physical address space
  - Quirks
  - Hardware bugs
  - Hotplugging of devices (including bridges)

$\Rightarrow$ almost no OS today does a proper job (amazing but true)
Barrelfish: PCIxexpress configuration in Prolog!

- Barrelfish PCI programming:
  1. SKB boots before PCI
  2. PCI enumeration populates SKB with devices
  3. SKB solves PCI config as a constraint satisfaction problem
  4. PCI driver programs bridges and devices with BAR values
  5. Incremental algorithm can handle hotplug

- Handles all the weird corner cases
- Exceptions are expressed independently of the main algorithm
  - Most hardware bugs require a one-liner
- Can optimize for free space (e.g. for HotPlug)

- Details: See Adrian Schüpbach et.al., ASPLOS 2011, ACM TOCS 2012
Wider insight

OS kernels use data structures for two purposes:

1. Traversal on fast path to route data and control correctly
   ⇒ Must be small, efficient, fast, specialized to architecture, etc.
2. Traversal for policy, resource allocation, etc.
   ⇒ Must be expressive, generic, self-describing, extensible, etc.

We argue something like this is essential for dealing with ever-more diverse and complex hardware
CONCLUSION
Ongoing work (selection)

• Outside the box
  – Barrelfish over a rack
• Storage system design
  – Direct mapped PCM?
  – Support for databases and file systems
• Network stack
  – Model machine as distributed router
• Languages for message-based systems programming
  – THC: asynchronous extensions to C (Tim Harris)
The Big Goal

• Key ideas:
  – Multikernel: the machine is a distributed system
  – System Knowledge Base: use decent reasoning online to guide the OS

• So far: great source of research directions!

• The Big Goal: a new platform for OS Research.
  – Flexible architecture
  – Less historical baggage
  – Well-matched to diverse future hardware
  – Plenty of research opportunities
Many thanks for listening!

• Interested? Please join us!

http://www.barrelfish.org/