The Multikernel
A new OS architecture for scalable multicore systems

Andrew Baumann\textsuperscript{1} Paul Barham\textsuperscript{2} Pierre-Evariste Dagand\textsuperscript{3}
Tim Harris\textsuperscript{2} Rebecca Isaacs\textsuperscript{2} Simon Peter\textsuperscript{1} Timothy Roscoe\textsuperscript{1}
Adrian Schüpbach\textsuperscript{1} Akhilesh Singhania\textsuperscript{1}

\textsuperscript{1} Systems Group, ETH Zurich \textsuperscript{2} Microsoft Research, Cambridge \textsuperscript{3} ENS Cachan Bretagne
Introduction

How should we structure an OS for future multicore systems?

- Scalability to many cores
- Heterogeneity and hardware diversity
System diversity

Sun Niagara T2

AMD Opteron (Istanbul)

Intel Nehalem (Beckton)
The interconnect matters

Today's 8-socket Opteron

SATA  PCIe  GbE

Floppy disk drive

CPU, L2  CPU, L2  CPU, L2  CPU, L2

CPU, L2  CPU, L2  CPU, L2  CPU, L2

CPU, L2  CPU, L2  CPU, L2  CPU, L2

L3  L3  L3  L3

RAM

CPU, L2  CPU, L2  CPU, L2  CPU, L2

CPU, L2  CPU, L2  CPU, L2  CPU, L2

CPU, L2  CPU, L2  CPU, L2  CPU, L2

L3  L3  L3  L3

RAM

CPU, L2  CPU, L2  CPU, L2  CPU, L2

CPU, L2  CPU, L2  CPU, L2  CPU, L2

CPU, L2  CPU, L2  CPU, L2  CPU, L2

L3  L3  L3  L3

RAM

CPU, L2  CPU, L2  CPU, L2  CPU, L2

CPU, L2  CPU, L2  CPU, L2  CPU, L2

CPU, L2  CPU, L2  CPU, L2  CPU, L2

L3  L3  L3  L3

RAM
The interconnect matters

Tomorrow's 8-socket Nehalem
The interconnect matters

On-chip interconnects
Core diversity

► Within a system:
  ► Programmable NICs
  ► GPUs
  ► FPGAs (in CPU sockets)

► On a single die:
  ► Performance asymmetry
  ► Streaming instructions (SIMD, SSE, etc.)
  ► Virtualisation support
Summary

- Increasing core counts, increasing diversity
- Unlike HPC systems, cannot optimise at design time
The multikernel model

▶ It’s time to rethink the default structure of an OS
  ▶ Shared-memory kernel on every core
  ▶ Data structures protected by locks
  ▶ Anything else is a device
The multikernel model

- It’s time to rethink the default structure of an OS
  - Shared-memory kernel on every core
  - Data structures protected by locks
  - Anything else is a device
- Proposal: structure the OS as a distributed system
- Design principles:
  1. Make inter-core communication explicit
  2. Make OS structure hardware-neutral
  3. View state as replicated
Outline

Introduction

Motivation
   Hardware diversity

The multikernel model
   Design principles
   The model

Barrelfish

Evaluation
   Case study: Unmap
1. Make inter-core communication explicit

- All communication with messages (no shared state)
1. Make inter-core communication explicit

- All communication with messages (no shared state)
- Decouples system structure from inter-core communication mechanism
  - Communication patterns explicitly expressed
- Naturally supports heterogeneous cores, non-coherent interconnects (PCIe)
- Better match for future hardware
  - …with cheap explicit message passing (e.g. Tile64)
  - …without cache-coherence (e.g. Intel 80-core)
- Allows split-phase operations
  - Decouple requests and responses for concurrency
- We can reason about it
Message passing vs. shared memory: experiment

Shared memory (move the data to the operation):

- Each core updates the same memory locations (no locking)
- Cache-coherence protocol migrates modified cache lines
  - Processor stalled while line is fetched or invalidated
  - Limited by latency of interconnect round-trips
  - Performance depends on data size (cache lines) and contention (number of cores)
Shared memory results

4×4-core AMD system

![Graph showing latency in cycles for SHM1 across different core counts.](image-url)
Shared memory results

4×4-core AMD system
Shared memory results

4 × 4-core AMD system

![Graph showing latency (cycles × 1000) vs. cores for SHM1, SHM2, and SHM4.](image)
Shared memory results

4×4-core AMD system
Message passing vs. shared memory: experiment

Message passing (move the operation to the data):

- A single server core updates the memory locations
- Each client core sends RPCs to the server
  - Operation and results described in a single cache line
  - Block while waiting for a response (in this experiment)
Message passing vs. shared memory: tradeoff

4×4-core AMD system

The Multikernel: A new OS architecture for scalable multicore systems
Message passing vs. shared memory: tradeoff

4×4-core AMD system

![Graph showing latency vs. number of cores for different memory access methods.]

- SHM8
- SHM4
- SHM2
- SHM1
- MSG8
- MSG1

Latency (cycles × 1000)

Cores

12.10.2009

The Multikernel: A new OS architecture for scalable multicore systems
Message passing vs. shared memory: tradeoff

4×4-core AMD system

Messaging faster for:
- ≥4 cores
- ≥4 cache lines

The Multikernel: A new OS architecture for scalable multicore systems
Message passing vs. shared memory: tradeoff

4×4-core AMD system

![Graph showing latency vs. number of cores for different memory types and update costs.](#)
Message passing vs. shared memory: tradeoff

$4 \times 4$-core AMD system

![Graph showing latency (cycles × 1000) vs. cores with different memory regions labeled as SHM8, SHM4, SHM2, SHM1, MSG8, MSG1, and Server. The graph indicates the actual cost of update at the server.]

“spare” cycles if RPC was split-phase
2. Make OS structure hardware-neutral

- Separate OS structure from hardware
- Only hardware-specific parts:
  - Message transports (highly optimised / specialised)
  - CPU / device drivers
2. Make OS structure hardware-neutral

- Separate OS structure from hardware
- Only hardware-specific parts:
  - Message transports (highly optimised / specialised)
  - CPU / device drivers
- Adaptability to changing performance characteristics
- Late-bind protocol and message transport implementations
3. View state as replicated

- Potentially-shared state accessed as if it were a local replica
  - Scheduler queues, process control blocks, etc.
3. View state as replicated

- Potentially-shared state accessed *as if* it were a local replica
  - Scheduler queues, process control blocks, etc.
- Required by message-passing model
- Naturally supports domains that do not share memory
- Naturally supports changes to the set of running cores
  - Hotplug, power management
Replication vs. sharing as default

- Replicas used as an optimisation in previous systems:
  - Tornado, K42: clustered objects
  - Linux: read-only data, kernel text
Replication vs. sharing as default

- Replicas used as an optimisation in previous systems: 
  - Tornado, K42 clustered objects 
  - Linux read-only data, kernel text

- In a multikernel, sharing is a local optimisation
  - Shared (locked) replica for threads or closely-coupled cores 
  - Hidden, local 
  - Only when faster, as decided at runtime 
  - Basic model remains split-phase
The multikernel model

User space:

OS:

Hardware:

Interconnect
Outline

Introduction

Motivation
   Hardware diversity

The multikernel model
   Design principles
   The model

Barrelfish

Evaluation
   Case study: Unmap
Barrelfish

- From-scratch implementation of a multikernel
- Supports x86-64 multiprocessors (ARM soon)
- Open source (BSD licensed)
Barrelfish structure
Monitors and CPU drivers

- CPU driver serially handles traps and exceptions
- Monitor mediates local operations on global state
- URPC inter-core (shared memory) message transport on *current* (cache-coherent) x86 HW
Non-original ideas in Barrelfish

Multiprocessor techniques:

▶ Minimise shared state (Tornado, K42, Corey)
▶ User-space messaging decoupled from IPIs (URPC)
▶ Single-threaded non-preemptive kernel per core (K42)

Other ideas we liked:

▶ Capabilities for all resource management (seL4)
▶ Upcall processor dispatch (Psyche, Sched. Activations, K42)
▶ Push policy into application domains (Exokernel, Nemesis)
▶ Lots of information (Infokernel)
▶ Run drivers in their own domains (µkernels)
▶ EDF as per-core CPU scheduler (RBED)
▶ Specify device registers in a little language (Devil)
Applications running on Barrellfish

- Slide viewer (this one!)
- Webserver (www.barrellfish.org)
- Virtual machine monitor (runs unmodified Linux)
- SPLASH-2, OpenMP (benchmarks)
- SQLite
- ECL\textsuperscript{i}PS\textsuperscript{e} (constraint engine)
- more...
Outline

Introduction

Motivation
  Hardware diversity

The multikernel model
  Design principles
  The model

Barrelfish

Evaluation
  Case study: Unmap
Evaluation goals
How do we evaluate an alternative OS structure?

- Good baseline performance
  - Comparable to existing systems on current hardware
- Scalability with cores
- Adapability to different hardware
- Ability to exploit message-passing for performance
Case study: Unmap (TLB shootdown)

▶ Send a message to every core with a mapping, wait for all to be acknowledged

▶ Linux/Windows:
  1. Kernel sends IPIs
  2. Spins on shared acknowledgement count/event

▶ Barrelfish:
  1. User request to local monitor domain
  2. Single-phase commit to remote cores

▶ How to implement communication?
Unmap communication protocols

write

cache-lines

Unicast

read
Unmap communication protocols

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unicast

Unica
Unmap communication protocols

Raw messaging cost

Latency (cycles x 1000) vs Cores

Broadcast

Unicast
Why use multicast

8×4-core AMD system
Why use multicast

8×4-core AMD system
Multicast communication
Multicast communication

“NUMA-aware” multicast
Unmap communication protocols

Raw messaging cost

![Graph showing latency vs. cores for different communication protocols. The x-axis represents the number of cores, ranging from 2 to 32. The y-axis represents latency (cycles × 1000), ranging from 0 to 14. The graph compares broadcast, unicast, multicast, and NUMA-Aware multicast.]
System knowledge base

- Constructing multicast tree requires hardware knowledge
  - Mapping of cores to sockets (CPUID data)
  - Messaging latency (online measurements)
- More generally, Barreelfish needs a way to reason about diverse system resources
System knowledge base

- Constructing multicast tree requires hardware knowledge
  - Mapping of cores to sockets (CPUID data)
  - Messaging latency (online measurements)
- More generally, Barrelfish needs a way to reason about diverse system resources
- We tackle this with constraint logic programming [Schüpbach et al., MMCS’08]
- System knowledge base stores rich, detailed representation of hardware, performs online reasoning
  - Initial implementation: port of the ECLiPSe constraint solver
- Prolog query used to construct multicast routing tree
Unmap latency

![Graph showing unmap latency for Windows, Linux, and Barrelfish across different cores.](image)
Summary of other results

- No penalty for shared-memory (SPLASH, OpenMP)
- Network throughput: 951.7Mbit/s (same as Linux)
- Pipelined web server
  - Static: 640 Mbit/s vs. 316 Mbit/s for lighttpd/Linux
  - Dynamic: 3417 requests/s (17.1Mbit/s) bottlenecked on SQL
Conclusion

- Modern computers are inherently distributed systems
- It’s time to rethink OS structure to match
- **The Multikernel**: model of the OS as a distributed system
  1. Explicit communication, replicated state
  2. Hardware-neutral OS structure
Conclusion

▶ Modern computers are inherently distributed systems
▶ It’s time to rethink OS structure to match
▶ The Multikernel: model of the OS as a distributed system
  1. Explicit communication, replicated state
  2. Hardware-neutral OS structure
▶ Barrelfish: our concrete implementation
  ▶ Reasonable performance on current hardware
  ▶ Better scalability/adaptability for future hardware
  ▶ Promising approach

www.barrelfish.org
Backup slides
URPC implementation

- Current hardware provides one communication mechanism: cache-coherent shared memory
- Can we “trick” cache-coherence protocol to send messages?
  - User-level RPC (URPC) [Bershad et al., 1991]

- Channel is shared ring buffer
- Messages are cache-line sized
- Sender writes message into next line
- Receiver polls on last word
- Marshalling/demarshalling, naming, binding all implemented above
Polling for receive
Tradeoff vs. IPIs

- Polling is cheap: line is local to receiver until message arrives
- Hardware-imposed costs for IPI (on $4 \times 4$-core AMD):
  - $\approx 800$ cycles to send (from user-mode)
  - $\approx 1200$ cycles lost in receive (to user-mode)
Polling for receive
Tradeoff vs. IPIs

- Polling is cheap: line is local to receiver until message arrives
- Hardware-imposed costs for IPI (on 4×4-core AMD):
  - ≈800 cycles to send (from user-mode)
  - ≈1200 cycles lost in receive (to user-mode)
- There is a tradeoff here!
- IPIs are decoupled from fast-path messaging, used only for:
  1. Specific (batches of) operations that require low latency, even when other tasks are executing
  2. Awakening cores that have blocked to save power (alternatively, MONITOR/MWAIT)