ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systems@ETH

The Barrelfish OS for scalable and heterogeneous multicore systems

Andrew Baumann

Introducing myself

I'm interested in:

- Operating systems
- Distributed/networked systems

I've worked on:

- Mungi single-address-space OS
- Tracing and performance monitoring in K42
- Dynamic update to operating systems
- OS timer usage study
- Rhizoma runtime for self-hosting distributed systems
- Barrelfish OS for heterogeneous multicore systems

Introducing Barrelfish

The Problem

How should we structure an OS for future multicore systems?

Goals:

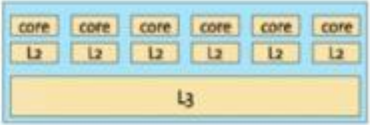
- Run a dynamic set of general-purpose applications
- Reduce code complexity to do this

Challenges:

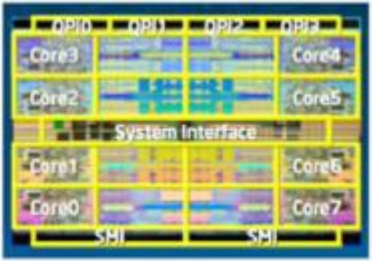
- Scalability to many cores
- Heterogeneity and hardware diversity

System diversity

Sun Niagara T2



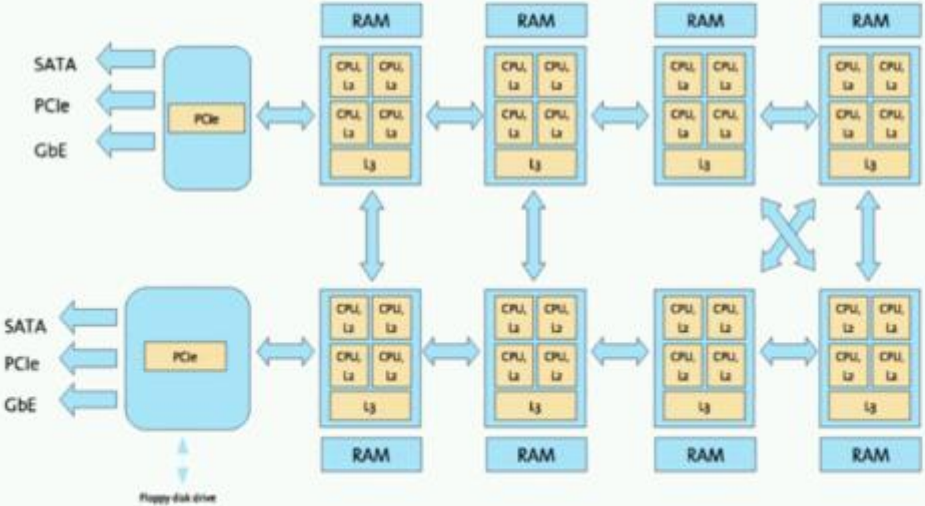
AMD Opteron (Istanbul)



Intel Nehalem (Beckton)

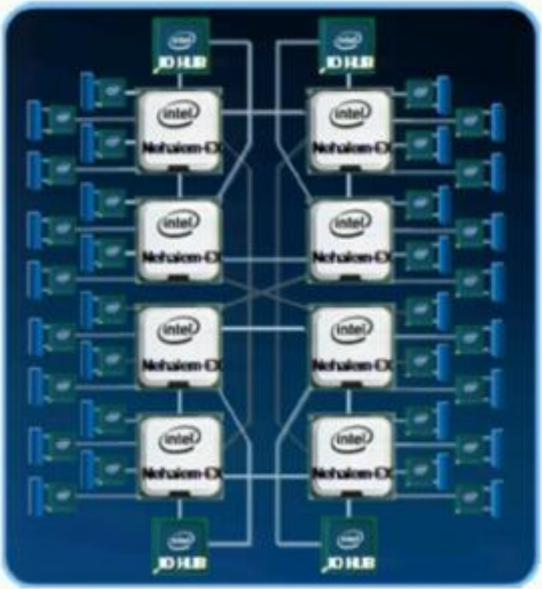
The interconnect matters

Today's 8-socket Opteron



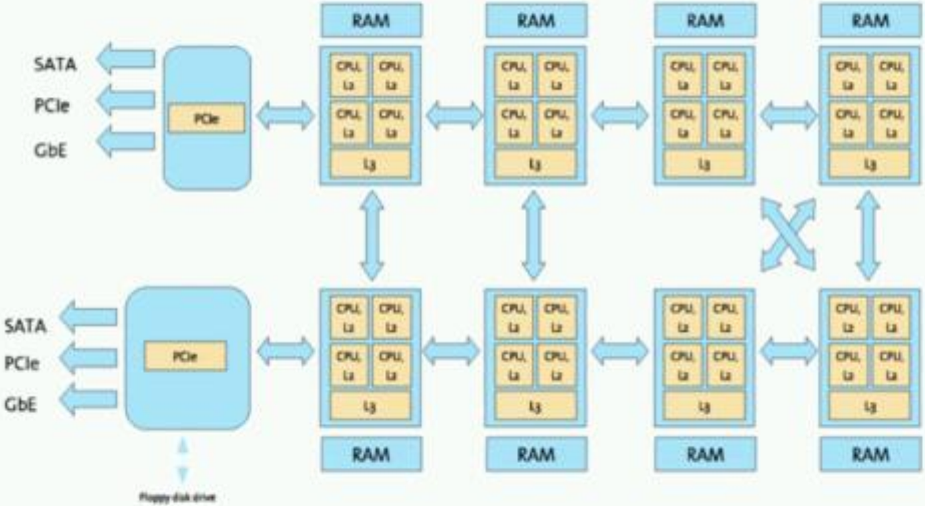
The interconnect matters

Tomorrow's 8-socket Nehalem



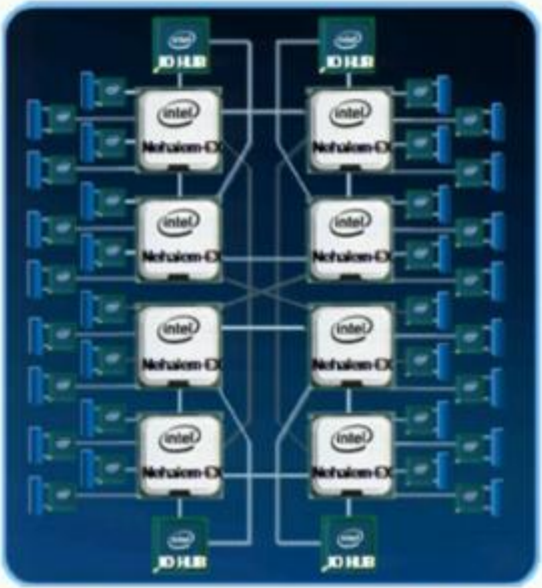
The interconnect matters

Today's 8-socket Opteron



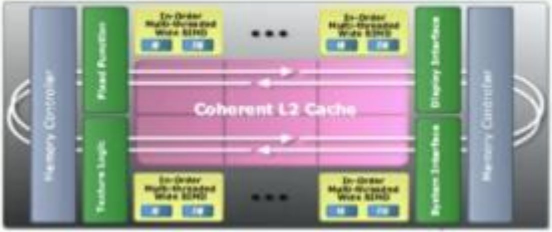
The interconnect matters

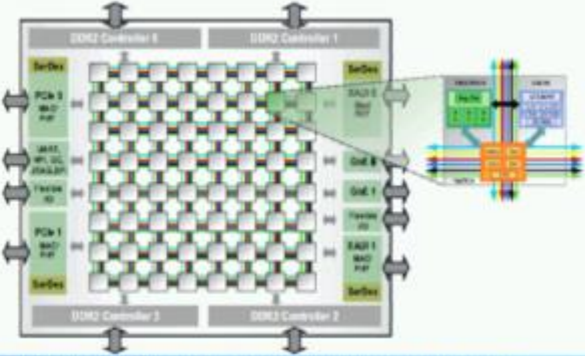
Tomorrow's 8-socket Nehalem



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

Your computer is already a distributed system

[Baumann et al., HotOS'09]

Summary:

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

Your computer is already a distributed system

[Baumann et al., HotOS'09]

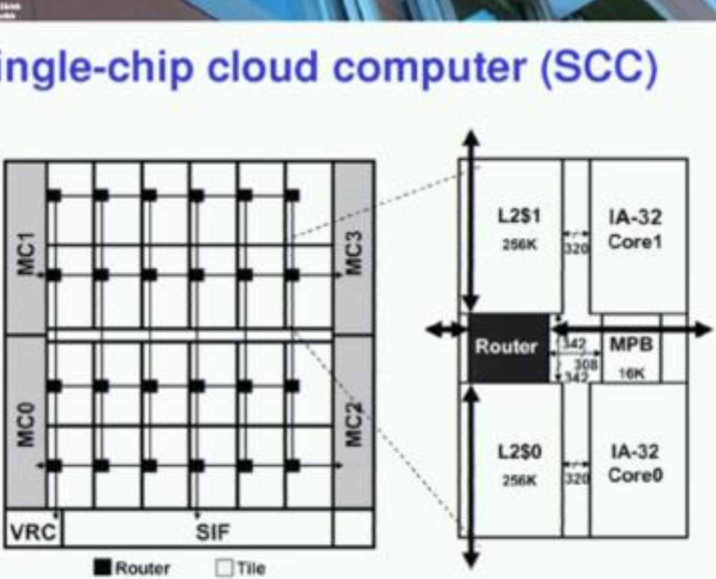
Summary:

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

Two current research examples:

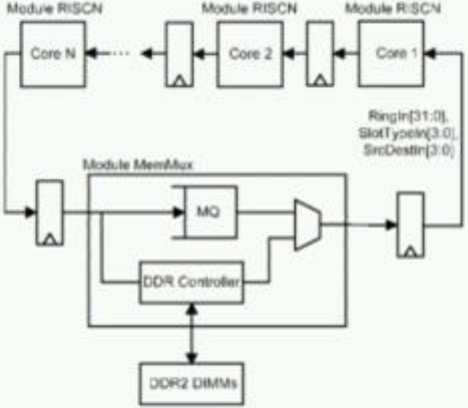
1. Intel SCC (Rock Creek)
2. MSR Beehive

Intel's single-chip cloud computer (SCC)



- 48 IA-32 cores on one die
- 2D mesh interconnect
- Message-passing architecture
- No cache coherence

MSR Beehive



- Ring interconnect
- Message passing
- No cache coherence
- Split-phase memory access

The multikernel model

[Baumann et al., SOSP'09]

- 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

[Baumann et al., SOSP'09]

- 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

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Outline

Introduction

Motivation
- Hardware diversity

The multikernel model
- Design principles
- The model

Barrelfish

Evaluation
- Case study: Unmap (TLB shootdown)

Future work

1. Make inter-core communication explicit

- All communication with messages (no shared state)

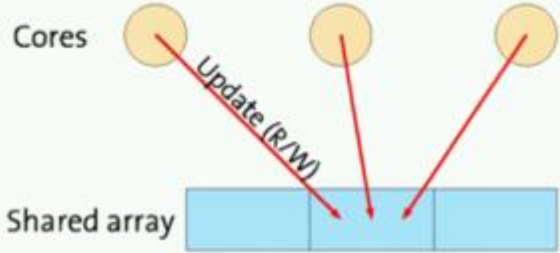
1. Make inter-core communication explicit

- 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 explicit message passing (e.g. Tile64, Beehive, SCC)
  - ...without cache-coherence (e.g. Beehive, SCC)
- Allows split-phase operations
  - Decouple requests and responses for concurrency
- We can reason about it

Message passing vs. shared memory: experiment

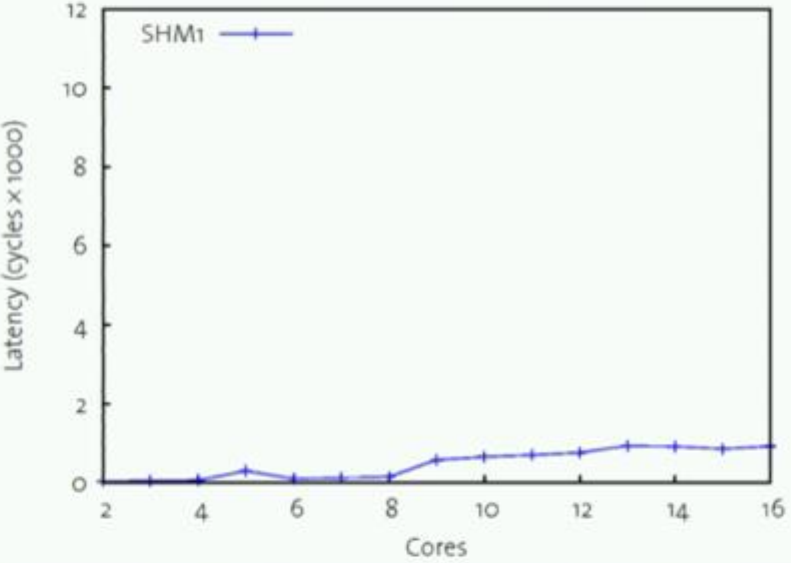
Shared memory (move the data to the operation):

- 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)



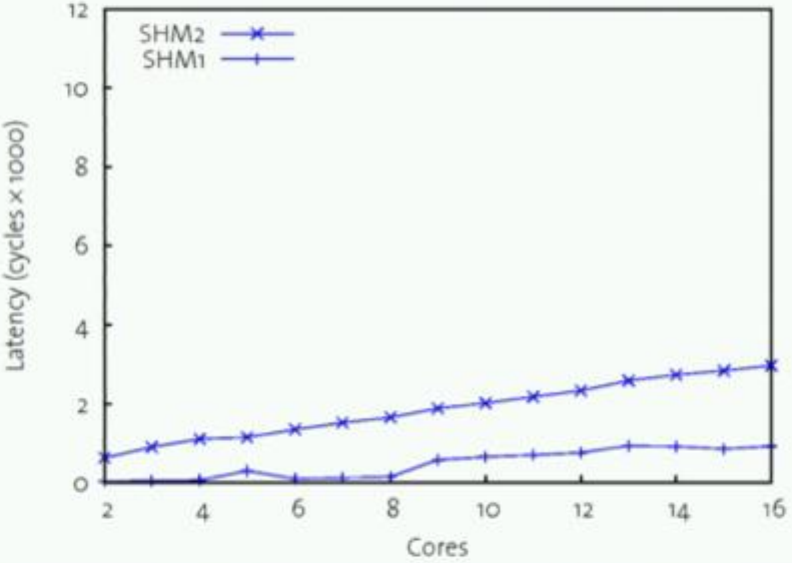
Shared memory results

4×4-core AMD system



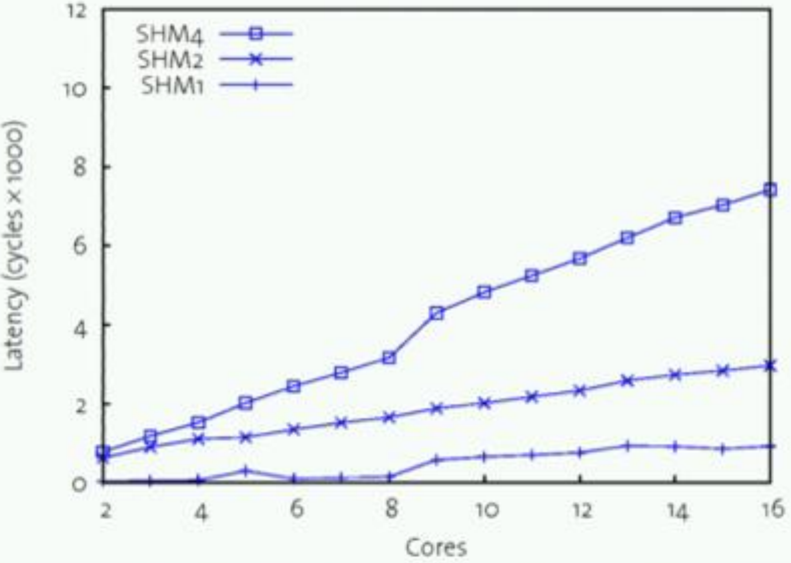
Shared memory results

4×4-core AMD system



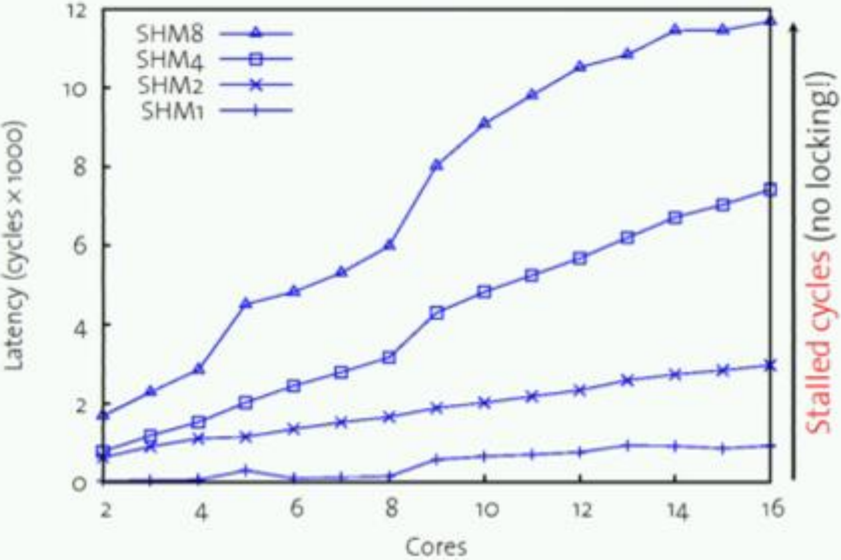
Shared memory results

4×4-core AMD system



Shared memory results

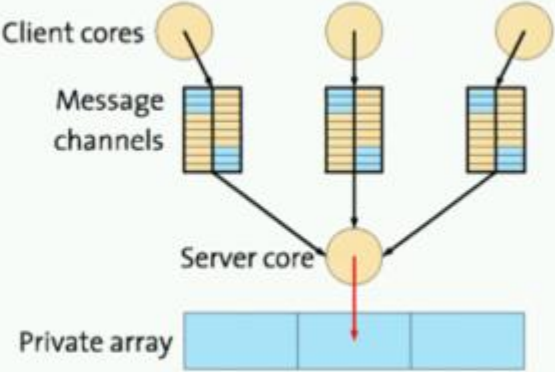
4×4-core AMD system



Message passing vs. shared memory: experiment

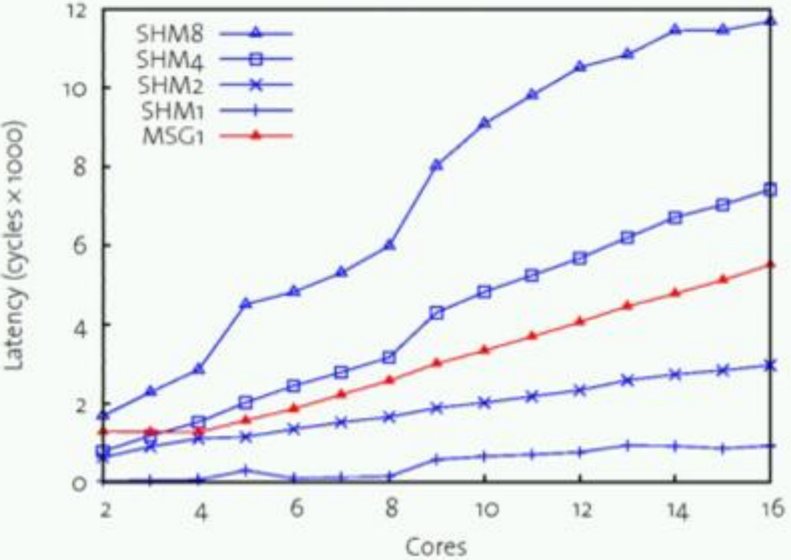
Message passing (move the operation to the data):

- 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)



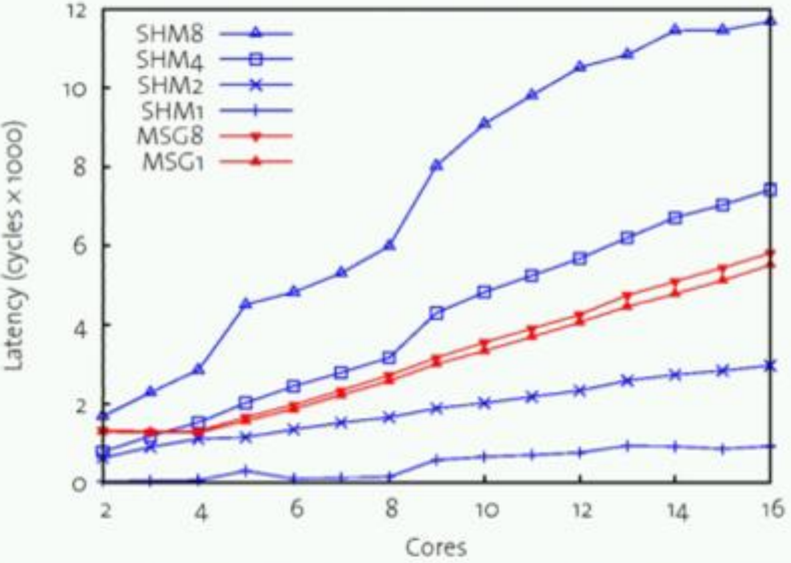
Message passing vs. shared memory: tradeoff

4×4-core AMD system



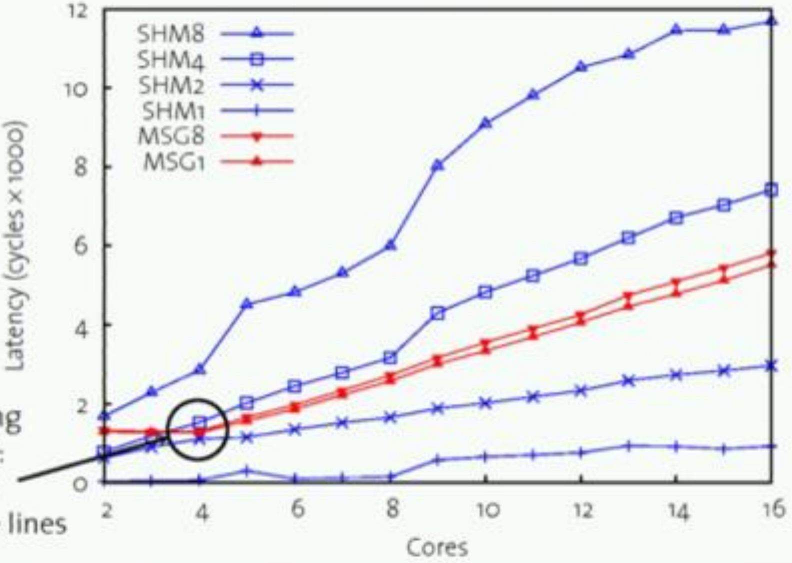
Message passing vs. shared memory: tradeoff

4×4-core AMD system



Message passing vs. shared memory: tradeoff

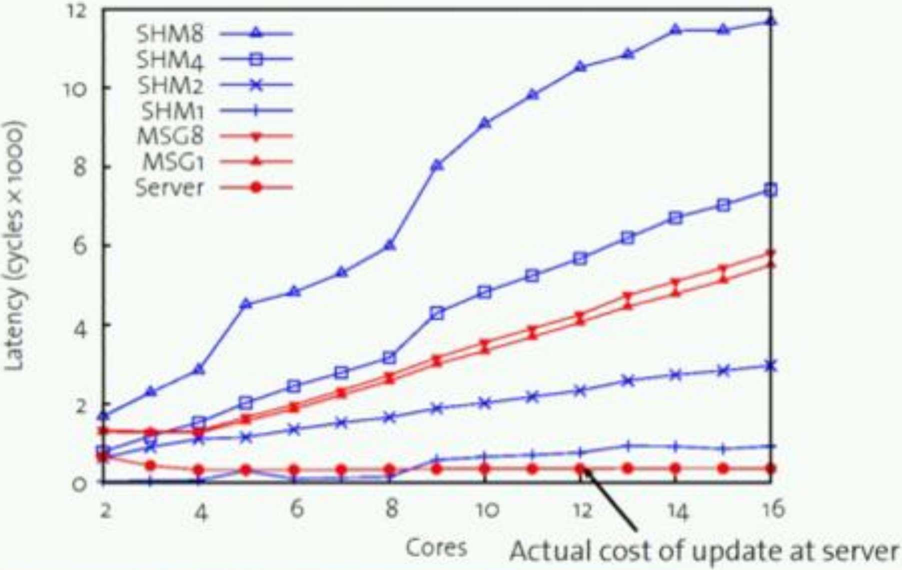
4×4-core AMD system



Messaging faster for:
≥4 cores
≥4 cache lines

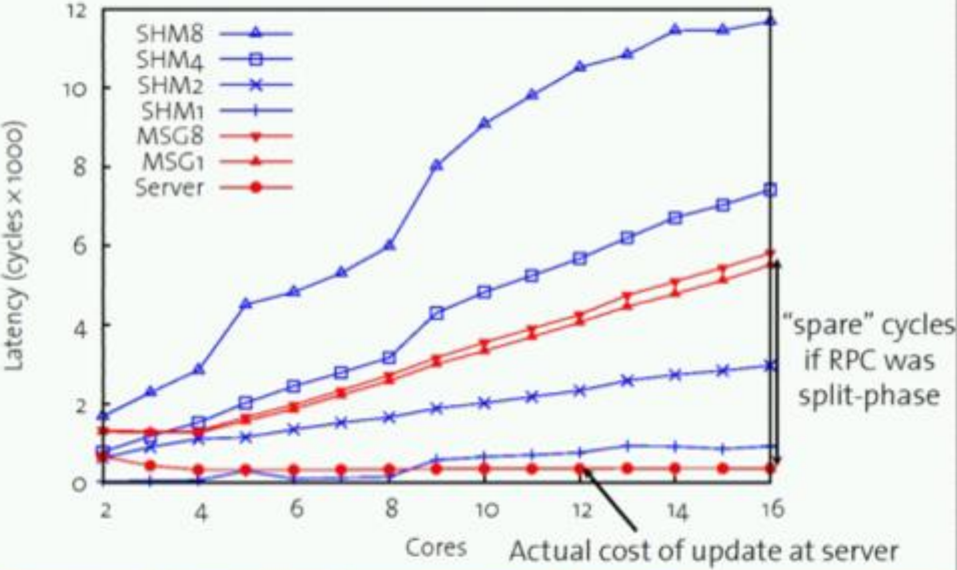
Message passing vs. shared memory: tradeoff

4×4-core AMD system



Message passing vs. shared memory: tradeoff

4×4-core AMD system



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
Altix Linux	read-only data, kernel text

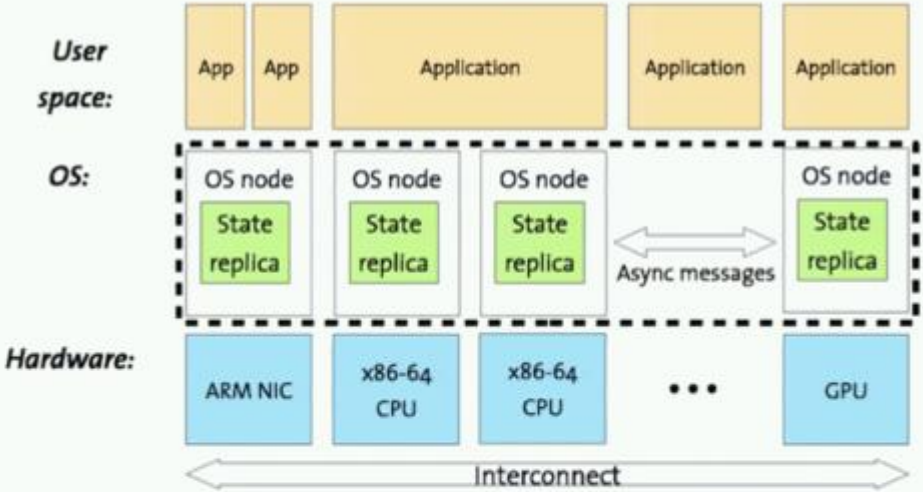
Replication vs. sharing as default

- Replicas used as an optimisation in previous systems:

  Tornado, K42 clustered objects
  Altix 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



Outline

- Introduction
- Motivation
  - Hardware diversity
- The multikernel model
  - Design principles
  - The model
- Barrelfish
- Evaluation
  - Case study: Unmap (TLB shootdown)
- Future work

Barrelfish

- From-scratch implementation of a multikernel
- Open source (BSD licensed)
- Supports x86-32/64 multiprocessors
  - Beehive, SCC, ARM ports well on the way
  - Combinations of these, in the same system

Who's involved?

Systems@ETH Zürich

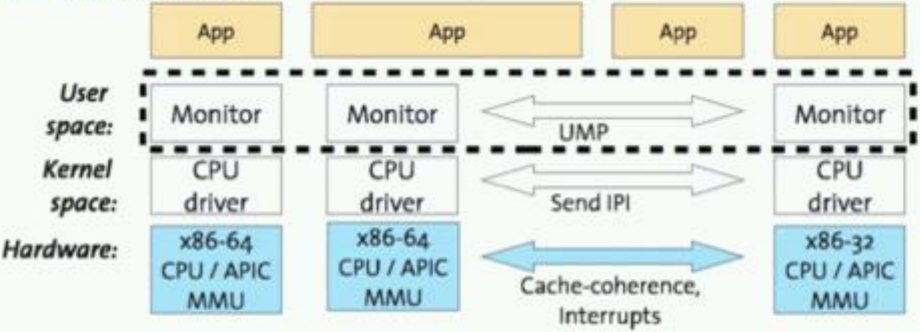
- Andrew Baumann
- Pierre-Evariste Dagand
- Simon Peter
- Timothy Roscoe
- Adrian Schüpbach
- Akhilesh Singhania

Microsoft Research Cambridge

- Paul Barham
- Richard Black
- Tim Harris
- Orion Hodson
- Rebecca Isaacs
- Ross McIlroy

Barrelfish structure

Monitors and CPU drivers



- CPU driver serially handles traps and exceptions
- Monitor mediates local operations on global state
- UMP inter-core message transport / interconnect driver
  - URPC on *current* (cache-coherent) x86 HW
- Other system facilities implemented as user-level services

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 Barrelfish

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

Outline

Introduction

Motivation
- Hardware diversity

The multikernel model
- Design principles
- The model

Barrelfish

Evaluation
- Case study: Unmap (TLB shootdown)

Future work

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

UMP performance (one-way)

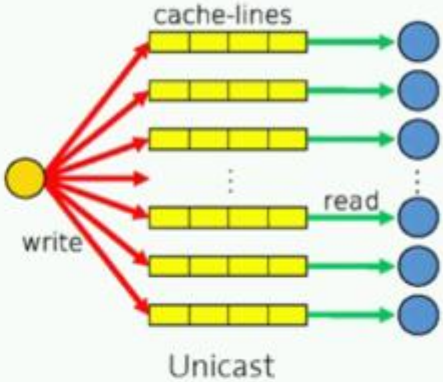
System	Cache	Latency cycles	Throughput msgs/kcycle
2×4-core Intel	shared	180	11.97
	non-shared	570	3.78
2×2-core AMD	same die	450	3.42
	one-hop	532	3.19
4×4-core AMD	shared	448	3.57
	one-hop	545	3.53
	two-hop	659	3.19
8×4-core AMD	shared	538	2.77
	one-hop	613	2.79
	two-hop	682	2.71

- Two HyperTransport requests on AMD
- Batching/pipelining for free

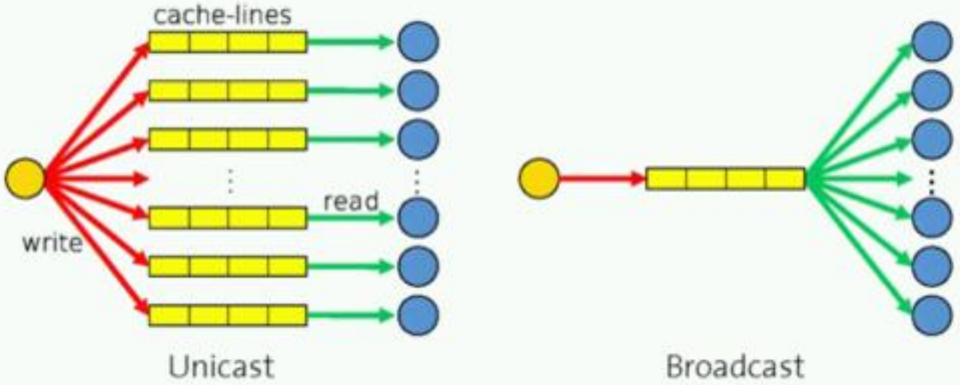
Case study: Unmap (TLB shootdown)

- 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

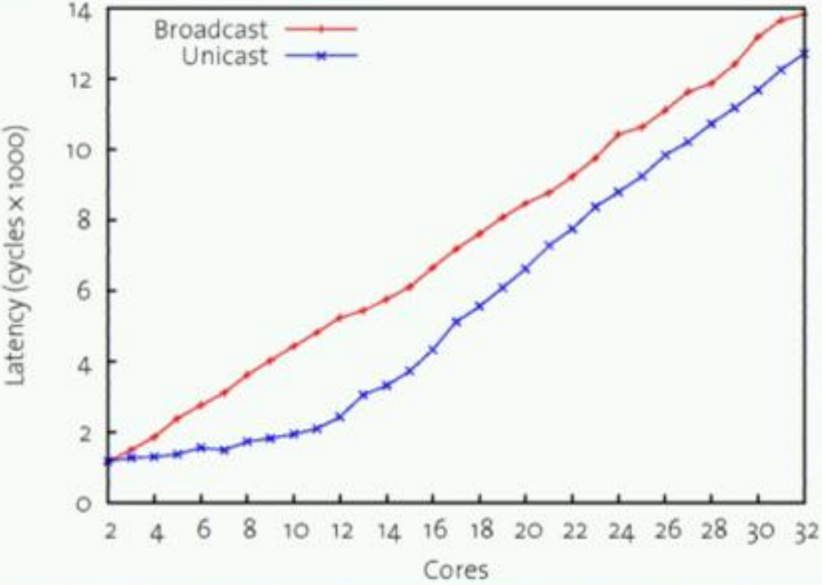


Unmap communication protocols



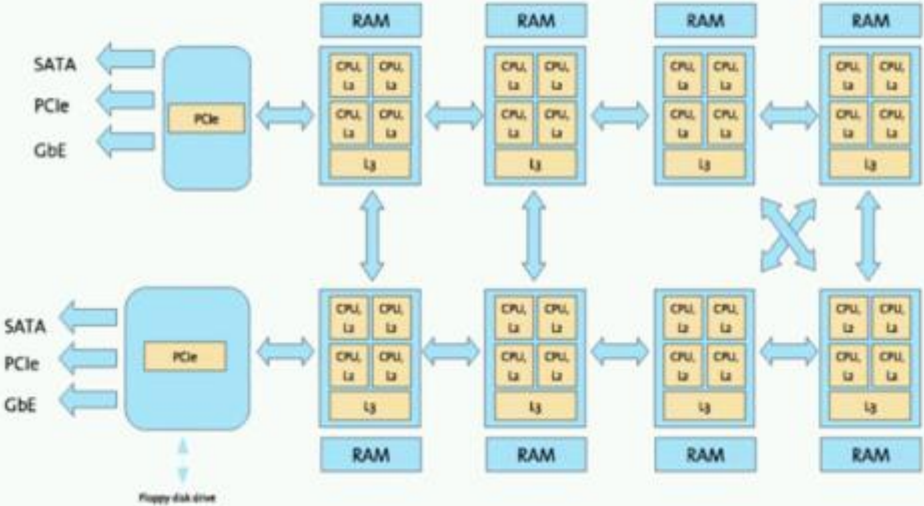
Unmap communication protocols

Raw messaging cost



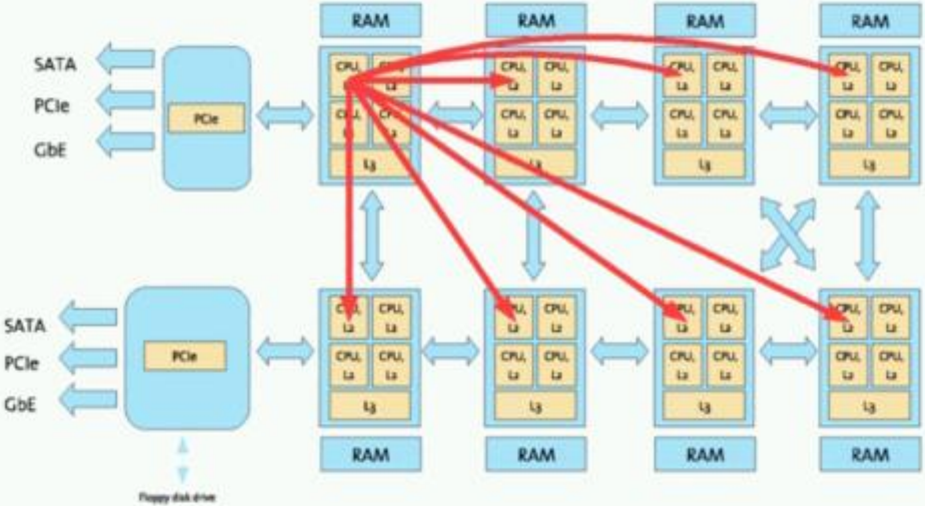
Why use multicast

8×4-core AMD system

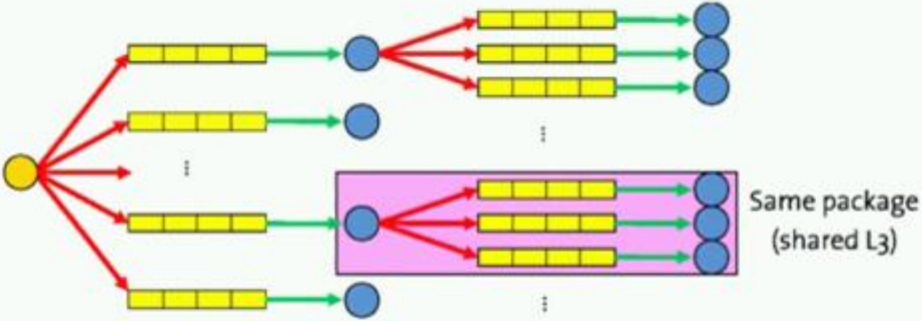


Why use multicast

8×4-core AMD system

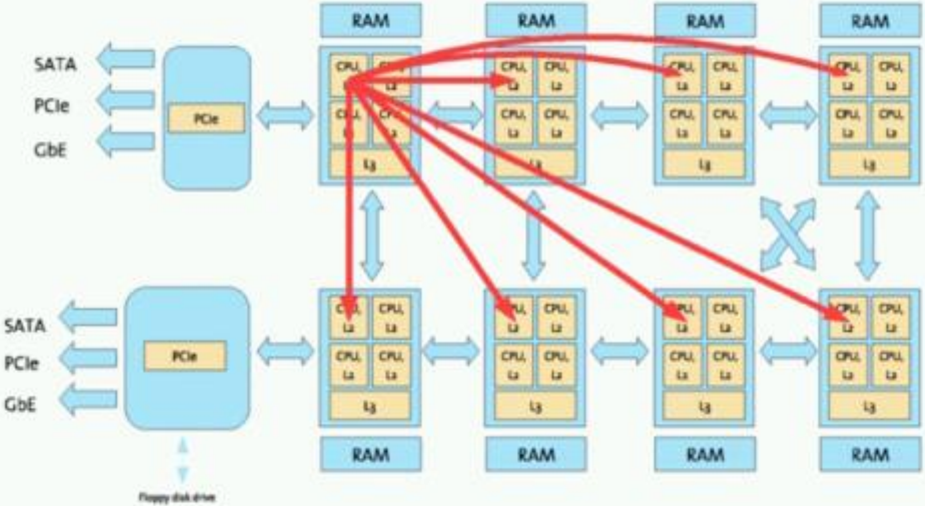


Multicast communication

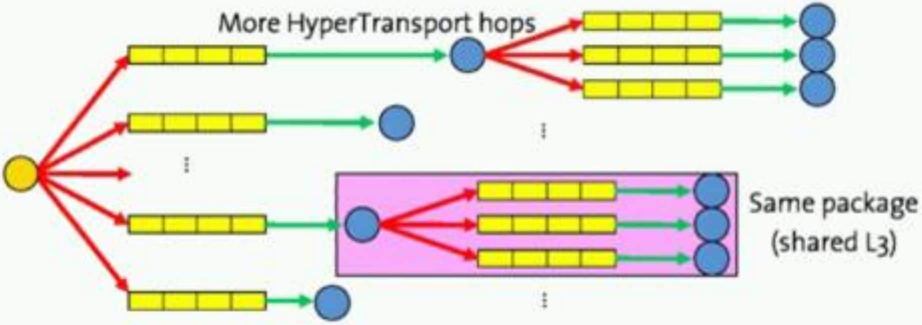


Why use multicast

8×4-core AMD system



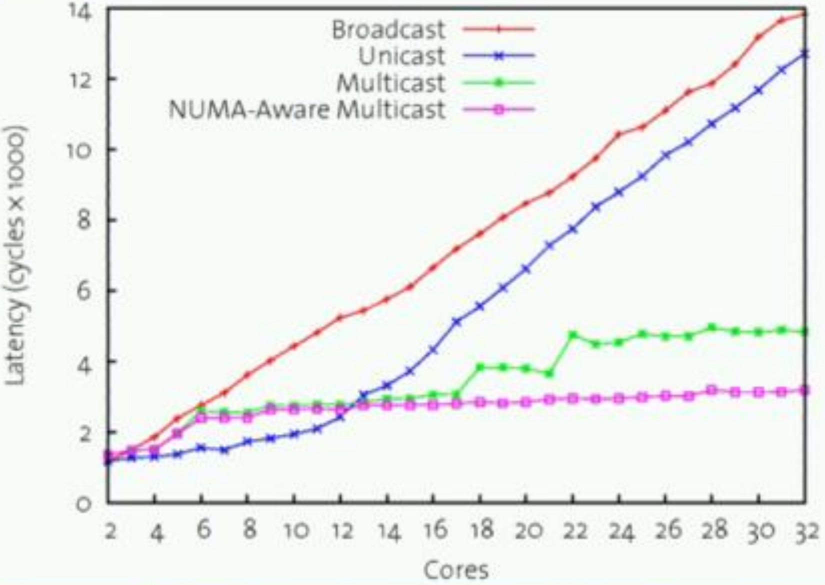
Multicast communication



- "NUMA-aware" multicast

Unmap communication protocols

Raw messaging cost



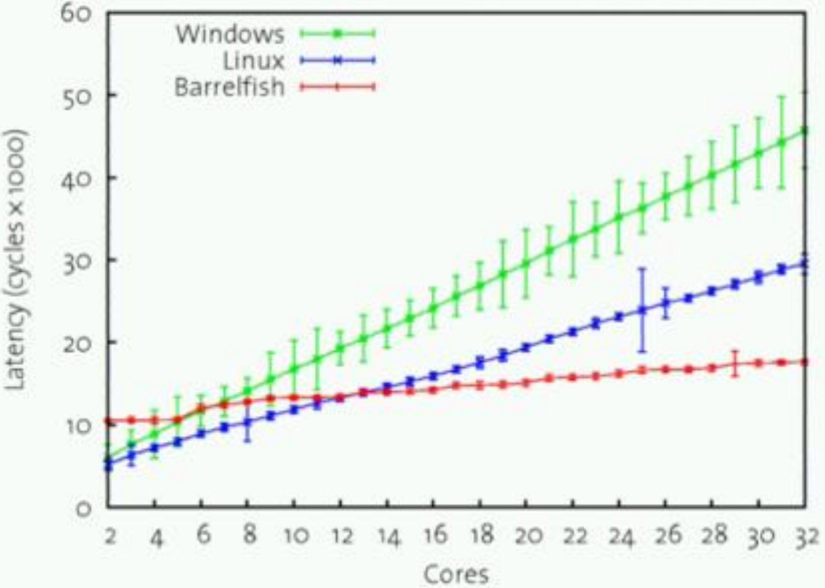
System knowledge base

- 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

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 ECL^{i}PS^e constraint solver
- Prolog query used to construct multicast routing tree

Unmap latency

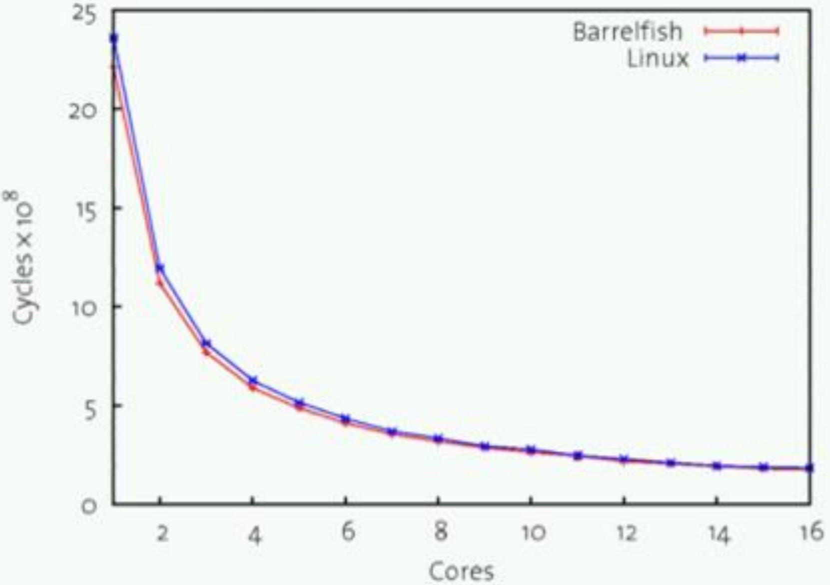


Compute-bound (shared memory) workloads

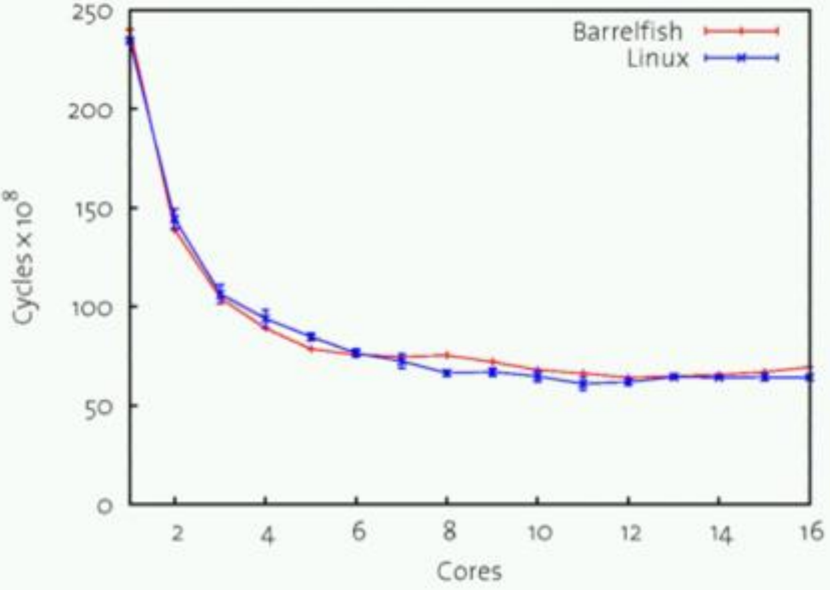
4×4-core AMD system

- Barrelfish support for shared-memory applications
- Benchmarks from NAS OpenMP and SPLASH-2 suites
- User-level synchronisation primitives (e.g. spinlocks)
- Mainly exercise hardware coherence mechanisms

SPLASH-2 Barnes-Hut



NAS OpenMP fast Fourier transform



IO workloads

- Faster IP loopback through efficient use of interconnect
- 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

Outline

- Introduction
- Motivation
  - Hardware diversity
- The multikernel model
  - Design principles
  - The model
- Barrelfish
- Evaluation
  - Case study: Unmap (TLB shootdown)
- Future work

Open questions and future work

- What are the right protocols for replication and agreement?
  - How can local sharing be used to optimise them?
  - Tradeoffs very different to classic distributed systems

Open questions and future work

- What are the right protocols for replication and agreement?
  - How can local sharing be used to optimise them?
  - Tradeoffs very different to classic distributed systems
- What does a native Barrelfish application look like?
  - Integration of high-level languages and multicore runtimes
  - New models for concurrent programming

Open questions and future work

- What are the right protocols for replication and agreement?
  - How can local sharing be used to optimise them?
  - Tradeoffs very different to classic distributed systems
- What does a native Barrelfish application look like?
  - Integration of high-level languages and multicore runtimes
  - New models for concurrent programming
- How do you construct system services above this model?
  - Networking architecture
  - Replicated file system based on cluster/HPC techniques

Open questions and future work

- What are the right protocols for replication and agreement?
  - How can local sharing be used to optimise them?
  - Tradeoffs very different to classic distributed systems
- What does a native Barrelfish application look like?
  - Integration of high-level languages and multicore runtimes
  - New models for concurrent programming
- How do you construct system services above this model?
  - Networking architecture
  - Replicated file system based on cluster/HPC techniques
- Heterogeneity-aware scheduling and resource allocation [Peter et al., HotPar'10]
  - Queuing effects, scheduling based on queue lengths
- Timeout behaviour in the OS [Peter et al., EuroSys'08]

Does the model extend outside a machine?

- A strong boundary at the machine edge is likely to remain:
  - Message guarantees (reliable, in-order)
  - Performance tradeoffs (latency / bandwidth)
  - Hardware support for shared memory
- However, a multikernel lowers the gap:
  - Explicit message-passing
  - Explicit split-phase replica-maintenance APIs

Does the model extend outside a machine?

- A strong boundary at the machine edge is likely to remain:
  - Message guarantees (reliable, in-order)
  - Performance tradeoffs (latency / bandwidth)
  - Hardware support for shared memory
- However, a multikernel lowers the gap:
  - Explicit message-passing
  - Explicit split-phase replica-maintenance APIs

Potential scenarios:

- Models for using resources inside one machine the same way as many machines
  - MapReduce, Dryad, etc.
- Extending your PC with cloud resources

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: concrete implementation
  - Real system
  - Reasonable performance on current hardware
  - Better scalability/adaptability for future hardware
  - Promising approach

www.barrelfish.org