Your computer is already a distributed system.
Why isn't your OS?

Andrew Baumann

Simon Peter, Adrian Schüpbach, Akhilesh Singhania, Timothy Roscoe
Systems Group, ETH Zurich

Paul Barham, Rebecca Isaacs
Microsoft Research, Cambridge

Introduction

- Observation: Modern multicore hardware is a network, and exhibits classic networking effects
- The OS should be designed as a distributed system

Outline

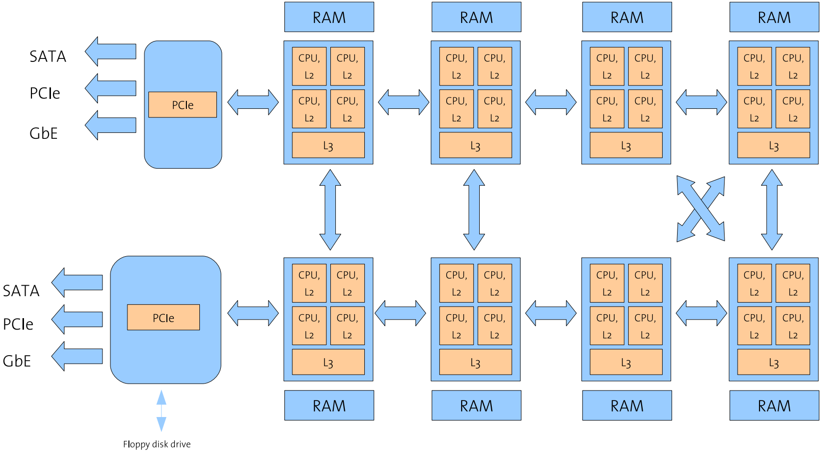
Observations
- Latency
- Heterogeneity
- Dynamic changes

Implications
- Message passing vs. shared memory
- Replication and consistency
- Heterogeneity

The multikernel architecture

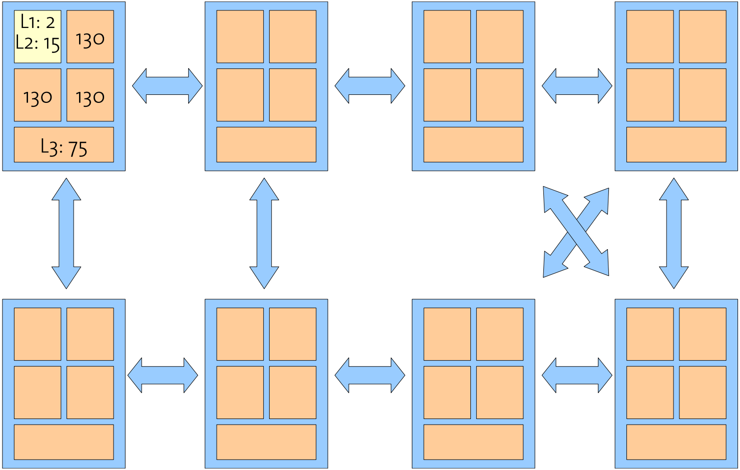
Observations

Does this look like a network to you?



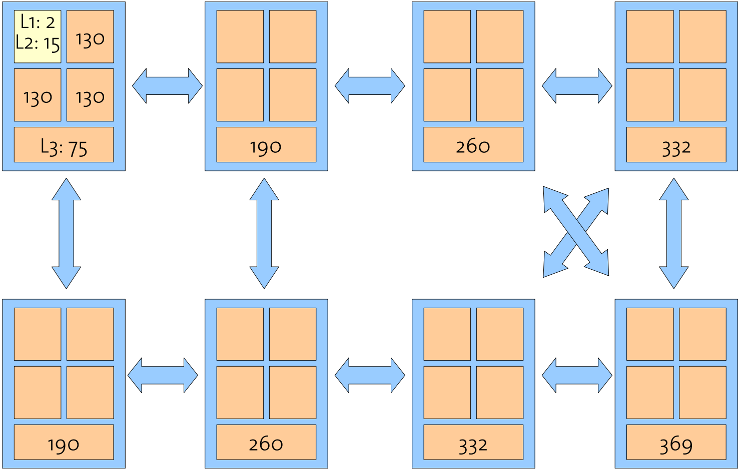
Communication latency

Cycles to access cache from core 0



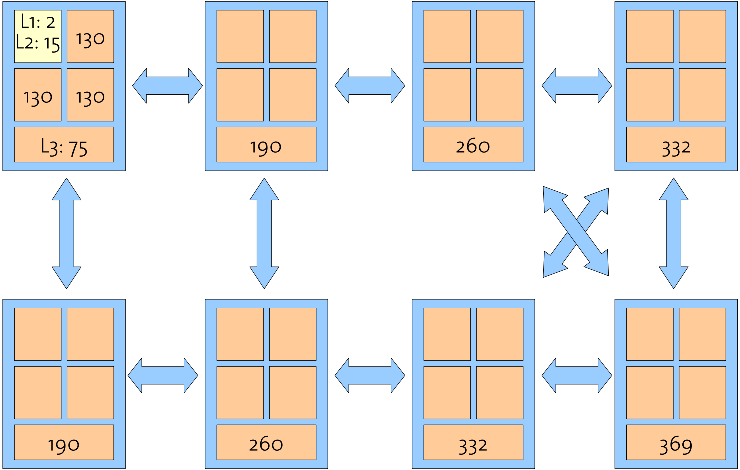
Communication latency

Cycles to access cache from core 0



Communication latency

Cycles to access cache from core 0



- Can shared data structures take advantage of this?

Node heterogeneity

- Within a system:
  - Programmable NICs
  - GPUs
  - FPGAs (in CPU sockets)
- Architectural differences on a single die:
  - Streaming instructions (SIMD, SSE, etc.)
  - Virtualisation support, power management

Node heterogeneity

- Within a system:
  - Programmable NICs
  - GPUs
  - FPGAs (in CPU sockets)
- Architectural differences on a single die:
  - Streaming instructions (SIMD, SSE, etc.)
  - Virtualisation support, power management
- Existing OS architectures have trouble accommodating this

Dynamic changes

- Hot-plug of devices, memory, (cores?)
- Power-management

Dynamic changes

- Hot-plug of devices, memory, (cores?)
- Power-management
- Partial failure

Summary

- Latency, heterogeneity, dynamic changes
- All classic characteristics of a distributed, networked system
- Why don't we program the machine this way?

The OS as a distributed system

What are the implications of building an OS as a distributed system?

- Extreme position: clean slate design
- Fully explore ramifications
- No regard for compatibility

Outline

Observations
- Latency
- Heterogeneity
- Dynamic changes

Implications
- Message passing vs. shared memory
- Replication and consistency
- Heterogeneity

The multikernel architecture

Message passing vs. shared memory

- Access to remote shared data can be seen as a blocking RPC
  - Processor stalled while line is fetched or invalidated
  - Limited by latency of interconnect round-trips
- Performance scales with size of data (number of cache lines)

Message passing vs. shared memory

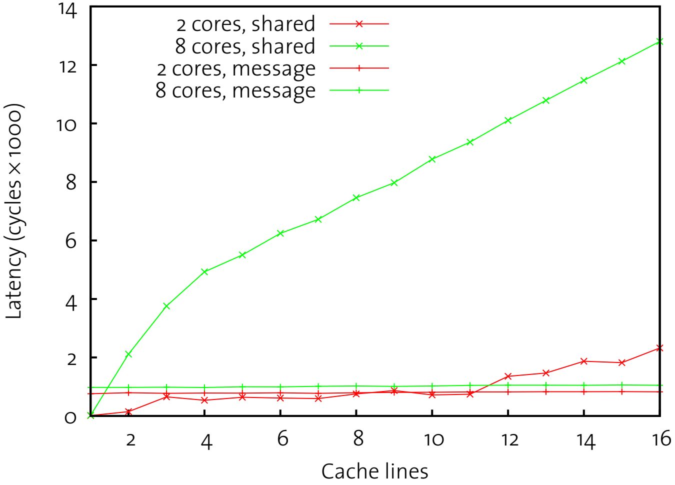
- Access to remote shared data can be seen as a blocking RPC
  - Processor stalled while line is fetched or invalidated
  - Limited by latency of interconnect round-trips
- Performance scales with size of data (number of cache lines)
- By sending an explicit RPC (message), we:
  - Send a compact high-level description of the operation
  - Reduce the time spent blocked, waiting for the interconnect
- Potential for more efficient use of interconnect bandwidth
- Cf. RPC vs. DSVM in distributed systems

Why message passing?

- We can reason about it
- Decouples system structure from inter-core communication mechanism
  - Communication patterns explicitly expressed
  - Naturally supports heterogeneous cores
  - Naturally supports 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)

Message passing vs. shared memory: tradeoff

2×4-core Intel (shared bus)



Shared: clients modify shared array (no locking) Message: URPC to server core

Replication

Given no sharing, what do we do with the state?

- Some state naturally partitions
- Other state must be replicated
- Used as an optimisation in previous systems:

  Tornado, K42 clustered objects

  Linux read-only data, kernel text
- We argue that replication should be the default

Consistency

- How do we maintain consistency of replicated data?
- Depends on consistency and ordering requirements, e.g.:

TLBs (unmap) single-phase commit

Memory reallocation (capabilities) two-phase commit

Cores come and go (power management, hotplug) agreement

Change of programming model: why wait?

- In a traditional OS, achieving consistency implies blocking
- e.g. unmap, global TLB shootdown

Idea: change programming model:

- Don't wait: do something else in the meantime
- Make such operations split-phase from user space
  - Propose a change, receive success/failure notification

$\Longrightarrow$ tradeoff latency vs. overhead

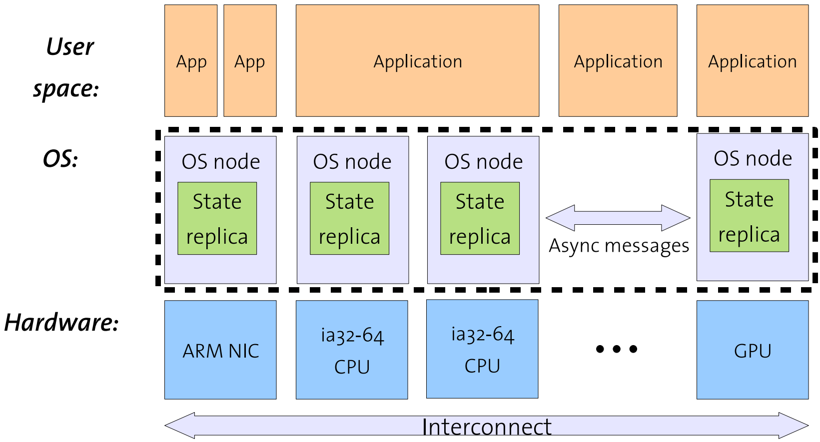
Heterogeneity

- Message-based communication handles core heterogeneity
  - Can specialise implementation and data structures
- Doesn't deal with other aspects
  - What should run where?
  - How should complex resources be allocated?
- Our solution based on constraint logic programming [Schüpbach et al., MMCS'08]
- System knowledge base stores rich, detailed representation of hardware, performs online reasoning

Outline

- Observations
  - Latency
  - Heterogeneity
  - Dynamic changes
- Implications
  - Message passing vs. shared memory
  - Replication and consistency
  - Heterogeneity
- The multikernel architecture

The multikernel architecture

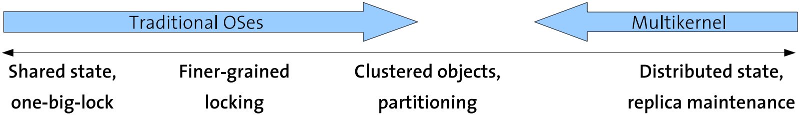


Optimisation

Sharing as an optimisation in multikernels

- We've replaced shared memory with explicit messaging
- But sharing/locking might be faster between some cores
  - Hyperthreads, or cores with shared L2/3 cache

$\Longrightarrow$ Re-introduce shared memory as optimisation
  - Hidden, local
  - Only when faster, as *decided at runtime*
  - Basic model remains split-phase



Conclusion

- Modern computers are inherently distributed systems
  - Communication latency, network effects
  - Heterogeneity
  - Dynamic behaviour
- We should be programming them as such
  - Message passing vs. sharing
  - Replication, consistency
  - Explicit management of heterogeneity
- Multikernel: a new OS architecture based on these ideas

Conclusion

- Modern computers are inherently distributed systems
  - Communication latency, network effects
  - Heterogeneity
  - Dynamic behaviour
- We should be programming them as such
  - Message passing vs. sharing
  - Replication, consistency
  - Explicit management of heterogeneity
- Multikernel: a new OS architecture based on these ideas
- Barrelfish: our implementation

www.barrelfish.org