What kind of distributed system is a multicore machine?

Stefan Kaestle
stefan.kaestle@inf.ethz.ch

Systems Group @ ETH Zurich
Who am I?

• PhD student with Prof. Timothy Roscoe
• Working on operating systems (Barrelfish)
  – But this talk is not only about that

• I will present
  – Trends of multicore hardware
  – Ongoing research in the Systems Group
  – Also: Opportunities for future research

Some of it is preliminary work. Lots of unknowns, feedback welcome
MULTICORE INTRODUCTION
Computer 20 years ago

CPU

memory
Computer 10 years ago

CPU

Cache coherency protocol

CPU

memory
Today

• Multicores:
  – Increasing number of cores
  – NUMA nodes
    • Local memory controllers
    • shared resources
  – Interconnect (not exposed)
So what is this talk about?

*aka why am I here?*

- **Multicores:**
  - Increasing number of cores
  - NUMA nodes
    - Local memory controllers, shared resources
  - Interconnect (not exposed)

→ Looks like *distributed systems*
So what is this talk about?

*aqua why am I here?*

- **Multicores:**
  - Increasing number of cores
  - NUMA nodes
  - Interconnect (not exposed)

→ Looks like **distributed systems**

Oh great, so lets just apply traditional DS algorithms
Example: replication of data

PROGRAM AS DISTRIBUTED SYSTEM
Interconnect characteristics

In common:

• Congestion
• Package based (internally)
• Routing
SHARED NOTHING ARCHITECTURE
Multikernel OS (Barrelfish)
Multikernel OS (Barrelfish)

Shared memory programming

Message passing
Multikernel OS (Barrelfish)

- No shared state
- Based on explicit message passing
- Triggers cache-coherency protocol

Shared memory programming

Message passing

Stefan Kästle <stefan.kaestle@inf.ethz.ch>
Multikernel OS (Barrelfish)

- No shared state
- Based on explicit message passing
- Triggers cache-coherency protocol

SCALABILITY: no locks, less synchronization overhead

Shared memory programming

Message passing
Multikernel OS (Barrelfish)

- Shared memory programming
- Message passing
  - Replication
  - 2 Phase Commit
  - Multicast trees
    - for TLB shoot-down
    - Machine aware

Stefan Kästle <stefan.kaestle@inf.ethz.ch>
Reduce interconnect traffic
Interconnect congestion (Shoal)

- Bad memory allocation
- Replication/distribution
Interconnect congestion (Shoal)

- Bad memory allocation
- Replication/distribution

- Reduces traffic on interconnect
So what is this talk about? *aka why am I here?*

- **Multicores:**
  - Increasing number of cores
  - NUMA nodes
    - Local memory controllers, shared resources
    - Interconnect (not exposed)

→ Looks like distributed systems

Okay, so all good? Can we go home now?
So what is this talk about? *aka why am I here?*

- Multicores:
  - Increasing number of cores
  - NUMA nodes
    - Local memory controllers, shared resources
    - Interconnect (not exposed)

→ Looks like distributed systems

Well, not quite ..
DIFFERENCES TO TRADITIONAL DS
Interconnect characteristics

In common:
- Congestion
- Package based (internally)
- Routing

Differences:
- Complexity measures
- Reliable
- Synchronous?
- Static (within a machine)
- Very concrete
- Diversity
- Hierarchical
- Hybrid

And many more ..
Complexity metrics

DIFFERENCES: AN EXAMPLE
Complexity metrics

What dominates in traditional DS?

What on a multicore?

send

propagate

receive
Complexity metrics

• Traditionally:
  – propagation time dominates
  – $\#\text{rounds}$ ($\#\text{messages/round irrelevant}$)

• Multicore:
  – Propagation cheap
  – Send and receive expensive
    • Interrupts, device driver communication, multiplexing, (un-) marshaling, scheduling
Example: broadcast

- Broadcast to \( n \) clients:

- Traditionally: send sequentially

- Multicore: BAD
  - \( \text{cost(seq)} : O(n) \)
Multicore/Broadcast

→ Tree, **NOT** balanced
(ideally: topology-aware, Radix)
Multicore/Broadcast++

- Leverage shared resources

- **Hybrid** algorithm:
  - Message passing across nodes
  - Shared memory inside of nodes

- Compose algorithm at runtime
  - machine-aware
  - scheduling-aware
Conclusions

• Multicores look like traditional DS
  – Apply ideas from DS

• But behave **differently**
  – Need to **re-evaluate** distributed algorithms
Failure Model

DIFFERENCES
Consensus

• RAFT/Paxos
• Need to reduce number of messages
• Treat some clusters of cores as failure-domain
  – Allows to use weaker algorithms inside

→ Compose algorithms
Failure model: TODAY

• Today: machine is reliable
  – Interconnect
    • Messages do not get lost
    • Upper bound on propagation time (synchronous)
Consensus: 2PC

- 2 Phases (1 RTT each)
  - Prepare
  - Commit

- Interconnect reliable

→ No ACKs in Commit Phase
Consensus: Paxos

• Do we want Paxos?
  – probably not, sends too many messages

• But what then?
  – Ongoing research, e.g. 1Paxos (EPFL, claims to be multicore aware)

• Failure domains?
Failure model: near FUTURE

• Parts of the machine can fail
  – Industry is very interested in this

• But: what is the unit of failure?
  – Parts of the machine can be treated as one failure domain (e.g. because the share resources)
    → again: hierarchy