18-447
Computer Architecture
Lecture 30: Interconnection Networks

Prof. Onur Mutlu
Carnegie Mellon University
Spring 2014, 4/28/2014
Lab 7: Multi-Core Cache Coherence

- Last submission accepted on May 9, 11:59:59 pm
- Cycle-level modeling of the MESI cache coherence protocol
Midterm 2 Statistics

- MAX 90.71
- MIN 22.86
- MEDIAN 48.01
- MEAN 42.5
- STD 16.24
Final Exam: May 6

- May 6, 8:30-11:30am, Hamerschlag Hall B103

- Comprehensive (over all topics in course)

- Three cheat sheets allowed

- We might have a review session

- Remember this is 25% of your grade
  - I will take into account your improvement over the course
  - Know all concepts, especially the previous midterm concepts
  - Same advice as before for Midterms I and II
A Note on 742, Research, Jobs

- I am teaching **Parallel Computer Architecture** next semester (Fall 2014)
  - Deep dive into many topics we covered
  - And, many topics we did not cover
  - Research oriented with an open-ended research project
  - Cutting edge research and topics in HW/SW interface

- If you are enjoying 447 and are doing well, you can take it
  - no need to have taken 640/740
  - talk with me

- If you are excited about Computer Architecture research or looking for a job/internship in this area
  - talk with me
Last Two Lectures

- Multiprocessors

- Bottlenecks in parallel processing

- Multiprocessor correctness
  - Sequential consistency
  - Weaker consistency

- Cache coherence
  - Software vs. hardware
  - Update vs. invalidate
  - Snoopy cache vs. directory based
  - VI $\rightarrow$ MSI $\rightarrow$ MESI $\rightarrow$ MOESI $\rightarrow$ ...
Today

- Wrap up cache coherence
- Interconnection networks
Readings: Multiprocessing

- **Required**

- **Recommended**
Readings: Memory Consistency

- **Required**

- **Recommended**
Readings: Cache Coherence

- **Required**
  - Culler and Singh, *Parallel Computer Architecture*
    - Chapter 5.1 (pp 269 – 283), Chapter 5.3 (pp 291 – 305)
  - P&H, *Computer Organization and Design*
    - Chapter 5.8 (pp 534 – 538 in 4th and 4th revised eds.)

- **Recommended**
Cache Coherence
Review: MESI State Machine from Lab 7

- **Invalid**
  - Other cache has write-miss (invalidate)
- **Shared**
  - Write (upgrade and inval. others)
  - Other cache has read-miss (downgrade)
- **Modified**
  - Write (mark dirty)
  - Other cache has write-miss (invalidate)
  - Cache miss (1 requester)
- **Exclusive**
  - Other cache has read-miss (downgrade)
Review: Intel Pentium Pro

Slide credit: Yale Patt
Snoopy Cache vs. Directory Coherence

**Snoopy Cache**
- Miss latency (critical path) is short: request → bus transaction to mem.
- Global serialization is easy: bus provides this already (arbitration)
- Simple: can adapt bus-based uniprocessors easily
  - Relies on broadcast messages to be seen by all caches (in same order):
    - → single point of serialization (bus): *not scalable*
    - → *need a virtual bus (or a totally-ordered interconnect)*

**Directory**
- Adds indirection to miss latency (critical path): request → dir. → mem.
- Requires extra storage space to track sharer sets
  - Can be approximate (false positives are OK)
- Protocols and race conditions are more complex (for high-performance)
+ Does not require broadcast to all caches
+ Exactly as scalable as interconnect and directory storage
  (*much more scalable than bus*)
Revisiting Directory-Based Cache Coherence
Remember: Directory Based Coherence

- **Idea:** A logically-central directory keeps track of where the copies of each cache block reside. Caches consult this directory to ensure coherence.

- **An example mechanism:**
  - For each cache block in memory, store \( P+1 \) bits in directory
    - One bit for each cache, indicating whether the block is in cache
    - Exclusive bit: indicates that the cache that has the only copy of the block and can update it without notifying others
  - On a read: set the cache’s bit and arrange the supply of data
  - On a write: invalidate all caches that have the block and reset their bits
  - Have an “exclusive bit” associated with each block in each cache
Remember: Directory Based Coherence

Example directory based scheme

\[ P = 4 \]

Exclusion bit

No cache has the block

\[ \begin{array}{c}
000000 \\
010000
\end{array} \]

\( P_1 \) takes a read miss to block A

\[ 000000 \rightarrow 010000 \]

\( P_3 \) takes a read miss

\[ \begin{array}{c}
010100 \\
010100
\end{array} \]
Directory-Based Protocols

- Required when scaling past the capacity of a single bus
- Distributed, *but*:
  - Coherence still requires single point of serialization (for write serialization)
  - Serialization location can be different for every block (striped across nodes)

- We can reason about the protocol for a single block: one *server* (directory node), many *clients* (private caches)

- Directory receives *Read* and *ReadEx* requests, and sends *InvI* requests: invalidation is explicit (as opposed to snoopy buses)
Directory: Basic Operations

- Follow *semantics* of snoop-based system
  - but with explicit request, reply messages

- Directory:
  - Receives *Read, ReadEx, Upgrade* requests from nodes
  - Sends *Inval/Downgrade* messages to sharers if needed
  - Forwards request to memory if needed
  - Replies to requestor and updates sharing state

- Protocol design is flexible
  - Exact forwarding paths depend on implementation
  - For example, do cache-to-cache transfer?
Mlesi Directory Transaction: Read

P0 acquires an address for reading:

1. Read

```
P0
```

2. DatEx (DatShr)

```
P1
```

Home
RdEx with Former Owner

1. RdEx

P0 ➔ Home

2. Invl

Home ➔ Owner

3a. Rev

Owner ➔ Home

3b. DatEx

Contention Resolution (for Write)

1a. RdEx
2a. DatEx
3. RdEx
4. Invl
5a. Rev
5b. DatEx

1b. RdEx
2b. NACK
Issues with Contention Resolution

- Need to escape race conditions by:
  - NACKing requests to busy (pending invalidate) entries
    - Original requestor retries
  - OR, queuing requests and granting in sequence
  - Or some combination thereof

- Fairness
  - Which requestor should be preferred in a conflict?
  - Both interconnect delivery order and distance matter

- Preventing ping-ponging is important
  - Can be achieved with better synchronization mechanisms or better prediction mechanisms
Scaling the Directory: Some Questions

- How large is the directory?

- How can we reduce the access latency to the directory?

- How can we scale the system to thousands of nodes?

- Can we get the best of snooping and directory protocols?
  - Think heterogeneity
Directory: Data Structures

- Required to support invalidation and cache block requests
- Key operation to support is set inclusion test
  - False positives are OK: want to know which caches may contain a copy of a block, and spurious invalidations are ignored
  - False positive rate determines performance
- Most accurate (and expensive): full bit-vector
- Compressed representation, linked list, Bloom filters are all possible
Interconnection Network Basics
Where Is Interconnect Used?

- To connect components

- Many examples
  - Processors and processors
  - Processors and memories (banks)
  - Processors and caches (banks)
  - Caches and caches
  - I/O devices

Interconnection network
Why Is It Important?

- Affects the scalability of the system
  - How large of a system can you build?
  - How easily can you add more processors?

- Affects performance and energy efficiency
  - How fast can processors, caches, and memory communicate?
  - How long are the latencies to memory?
  - How much energy is spent on communication?
Interconnection Network Basics

- **Topology**
  - Specifies the way switches are wired
  - Affects routing, reliability, throughput, latency, building ease

- **Routing (algorithm)**
  - How does a message get from source to destination
  - Static or adaptive

- **Buffering and Flow Control**
  - What do we store within the network?
    - Entire packets, parts of packets, etc?
  - How do we throttle during oversubscription?
  - Tightly coupled with routing strategy
Topology

- Bus (simplest)
- Point-to-point connections (ideal and most costly)
- Crossbar (less costly)
- Ring
- Tree
- Omega
- Hypercube
- Mesh
- Torus
- Butterfly
- …
Metrics to Evaluate Interconnect Topology

- Cost
- Latency (in hops, in nanoseconds)
- Contention

- Many others exist you should think about
  - Energy
  - Bandwidth
  - Overall system performance
+ Simple
+ Cost effective for a small number of nodes
+ Easy to implement coherence (snooping and serialization)
- Not scalable to large number of nodes (limited bandwidth, electrical loading $\rightarrow$ reduced frequency)
- High contention $\rightarrow$ fast saturation
Point-to-Point

Every node connected to every other

+ Lowest contention
+ Potentially lowest latency
+ Ideal, if cost is not an issue

-- Highest cost
  O(N) connections/ports per node
  O(N^2) links
-- Not scalable
-- How to lay out on chip?
**Crossbar**

- Every node connected to every other (non-blocking) except one can be using the connection at any given time
- Enables concurrent transfers to non-conflicting destinations
- Could be cost-effective for small number of nodes

+ Low latency and high throughput
- Expensive
- Not scalable $\rightarrow O(N^2)$ cost
- Difficult to arbitrate as $N$ increases

Used in core-to-cache-bank networks in
- IBM POWER5
- Sun Niagara I/II
Another Crossbar Design
Sun UltraSPARC T2 Core-to-Cache Crossbar

- High bandwidth interface between 8 cores and 8 L2 banks & NCU
- 4-stage pipeline: req, arbitration, selection, transmission
- 2-deep queue for each src/dest pair to hold data transfer request
Bufferless and Buffered Crossbars

+ Simpler arbitration/scheduling

+ Efficient support for variable-size packets

- Requires $N^2$ buffers
Can We Get Lower Cost than A Crossbar?

- Yet still have low contention compared to a bus?
- Idea: Multistage networks
Multistage Logarithmic Networks

- Idea: Indirect networks with multiple layers of switches between terminals/nodes
- Cost: $O(N \log N)$, Latency: $O(\log N)$
- Many variations (Omega, Butterfly, Benes, Banyan, ...)
- **Omega Network:**

```
000 001 010 011 100 101 110 111
```

```
000 001 010 011 100 101 110 111
```

```
conflict
```
Multistage Networks (Circuit Switched)

- Multistage has more restrictions on feasible concurrent Tx-Rx pairs
- But more scalable than crossbar in cost, e.g., $O(N \log N)$ for Butterfly
Multistage Networks (Packet Switched)

- Packets “hop” from router to router, pending availability of the next-required switch and buffer

2-by-2 router
Aside: Circuit vs. Packet Switching

- **Circuit switching** sets up full path
  - Establish route then send data
  - No one else can use those links
  + faster arbitration
  -- setting up and bringing down links takes time

- **Packet switching** routes per packet
  - Route each packet individually (possibly via different paths)
  - If link is free, any packet can use it
  -- potentially slower --- must dynamically switch
  + no setup, bring down time
  + more flexible, does not underutilize links
Switching vs. Topology

- Circuit/packet switching choice independent of topology
- It is a higher-level protocol on how a message gets sent to a destination

- However, some topologies are more amenable to circuit vs. packet switching
Another Example: Delta Network

- Single path from source to destination
- Each stage has different routers
- Proposed to replace costly crossbars as processor-memory interconnect
Another Example: Omega Network

- Single path from source to destination
- All stages are the same
- Used in NYU Ultracomputer
Ring

+ Cheap: \(O(N)\) cost
- High latency: \(O(N)\)
- Not easy to scale
  - Bisection bandwidth remains constant

Used in Intel Haswell, Intel Larrabee, IBM Cell, many commercial systems today
**Unidirectional Ring**

- Simple topology and implementation
  - Reasonable performance if N and performance needs (bandwidth & latency) still moderately low
  - $O(N)$ cost
  - $N/2$ average hops; latency depends on utilization

![Diagram of Unidirectional Ring with 2x2 routers](image-url)
Bidirectional Rings

+ Reduces latency
+ Improves scalability

- Slightly more complex injection policy (need to select which ring to inject a packet into)
Hierarchical Rings

+ More scalable
+ Lower latency
- More complex

(a) 4-, 8-, and 16-bridge hierarchical ring topologies.

(b) Three-level hierarchy (8x8).
More on Hierarchical Rings


- Discusses the design and implementation of a mostly-bufferless hierarchical ring
Mesh

- O(N) cost
- Average latency: O(sqrt(N))
- Easy to layout on-chip: regular and equal-length links
- Path diversity: many ways to get from one node to another

- Used in Tilera 100-core
- And many on-chip network prototypes
Torus

- Mesh is not symmetric on edges: performance very sensitive to placement of task on edge vs. middle
- Torus avoids this problem
  + Higher path diversity (and bisection bandwidth) than mesh
  - Higher cost
  - Harder to lay out on-chip
    - Unequal link lengths

![Torus diagram]
Torus, continued

- Weave nodes to make inter-node latencies $\sim$ constant
Trees

Planar, hierarchical topology
Latency: $O(\log N)$
Good for local traffic
+ Cheap: $O(N)$ cost
+ Easy to Layout
- Root can become a bottleneck
  Fat trees avoid this problem (CM-5)
CM-5 Fat Tree

- Fat tree based on 4x2 switches
- Randomized routing on the way up
- Combining, multicast, reduction operators supported in hardware
Hypercube

- Latency: $O(\log N)$
- Radix: $O(\log N)$
- #links: $O(N\log N)$

+ Low latency
- Hard to lay out in 2D/3D
Caltech Cosmic Cube

- 64-node message passing machine
Handling Contention

- Two packets trying to use the same link at the same time
- What do you do?
  - Buffer one
  - Drop one
  - Misroute one (deflection)
- Tradeoffs?
**Bufferless Deflection Routing**

- **Key idea:** Packets are never buffered in the network. When two packets contend for the same link, one is deflected.¹

> New traffic can be injected whenever there is a free output link.

Bufferless Deflection Routing

- Input buffers are eliminated: packets are buffered in **pipeline latches** and on **network links**
Routing Algorithm

- Types
  - **Deterministic**: always chooses the same path for a communicating source-destination pair
  - **Oblivious**: chooses different paths, without considering network state
  - **Adaptive**: can choose different paths, adapting to the state of the network

- How to adapt
  - Local/global feedback
  - Minimal or non-minimal paths
Deterministic Routing

- All packets between the same (source, dest) pair take the same path

- **Dimension-order routing**
  - E.g., XY routing (used in Cray T3D, and many on-chip networks)
  - First traverse dimension X, then traverse dimension Y

  + Simple
  + Deadlock freedom (no cycles in resource allocation)
  - Could lead to high contention
  - Does not exploit path diversity
Deadlock

- No forward progress
- Caused by circular dependencies on resources
- Each packet waits for a buffer occupied by another packet downstream
Handling Deadlock

- Avoid cycles in routing
  - Dimension order routing
    - Cannot build a circular dependency
  - Restrict the “turns” each packet can take

- Avoid deadlock by adding more buffering (escape paths)

- Detect and break deadlock
  - Preemption of buffers
Turn Model to Avoid Deadlock

- **Idea**
  - Analyze directions in which packets can turn in the network
  - Determine the cycles that such turns can form
  - Prohibit just enough turns to break possible cycles


![Diagram of possible turns and cycles](image1)

![Diagram of turns allowed by the xy routing algorithm](image2)

![Diagram of turns that complete the cycles and allow deadlock](image3)
Oblivious Routing: Valiant’s Algorithm

- An example of oblivious algorithm
- Goal: Balance network load
- Idea: Randomly choose an intermediate destination, route to it first, then route from there to destination
  - Between source-intermediate and intermediate-dest, can use dimension order routing

+ Randomizes/balances network load
- Non minimal (packet latency can increase)

- Optimizations:
  - Do this on high load
  - Restrict the intermediate node to be close (in the same quadrant)
Adaptive Routing

- **Minimal adaptive**
  - Router uses network state (e.g., downstream buffer occupancy) to pick which “productive” output port to send a packet to
  - Productive output port: port that gets the packet closer to its destination
    + Aware of local congestion
    - Minimality restricts achievable link utilization (load balance)

- **Non-minimal (fully) adaptive**
  - “Misroute” packets to non-productive output ports based on network state
    + Can achieve better network utilization and load balance
    - Need to guarantee livelock freedom
On-Chip Networks

- Connect **cores, caches, memory controllers, etc**
  - Buses and crossbars are not scalable
- **Packet switched**
- **2D mesh**: Most commonly used topology
- Primarily serve **cache misses** and memory requests

**Diagram Notes**

- **PE** (Processing Element) (Cores, L2 Banks, Memory Controllers, etc)
- **R** (Router)
On-chip Networks

Router
Processing Element
(Cores, L2 Banks, Memory Controllers etc)

Crossbar (5 x 5)

Input Port with Buffers

Control Logic
Routing Unit (RQ)
VC Allocator (VA)
Switch Allocator (SA)

From East
From West
From North
From South
From PE

VC 0
VC 1
VC 2

Crossbar

To East
To West
To North
To South
To PE
On-Chip vs. Off-Chip Interconnects

- **On-chip advantages**
  - Low latency between cores
  - No pin constraints
  - Rich wiring resources
    - Very high bandwidth
    - Simpler coordination

- **On-chip constraints/disadvantages**
  - 2D substrate limits implementable topologies
  - Energy/power consumption a key concern
  - Complex algorithms undesirable
  - Logic area constrains use of wiring resources
On-Chip vs. Off-Chip Interconnects (II)

- **Cost**
  - Off-chip: Channels, pins, connectors, cables
  - On-chip: Cost is storage and switches (wires are plentiful)
  - Leads to networks with many wide channels, few buffers

- **Channel characteristics**
  - On chip short distance $\rightarrow$ low latency
  - On chip RC lines $\rightarrow$ need repeaters every 1-2mm
    - Can put logic in repeaters

- **Workloads**
  - Multi-core cache traffic vs. supercomputer interconnect traffic
Motivation for Efficient Interconnect

- In many-core chips, on-chip interconnect (NoC) consumes significant power
  - **Intel Terascale**: ~28% of chip power
  - **Intel SCC**: ~10%
  - **MIT RAW**: ~36%

- Recent work\(^1\) uses **bufferless deflection routing** to reduce power and die area

---

Research Topics in Interconnects

- Plenty of topics in interconnection networks. Examples:
  - **Energy/power** efficient and proportional design
  - **Reducing Complexity**: Simplified router and protocol designs
  - **Adaptivity**: Ability to adapt to different access patterns
  - **QoS and performance isolation**
    - Reducing and controlling interference, admission control
  - **Co-design of NoCs with other shared resources**
    - End-to-end performance, QoS, power/energy optimization
  - **Scalable topologies** to many cores, heterogeneous systems
  - **Fault tolerance**
  - **Request prioritization, priority inversion, coherence, ...**
  - **New technologies** (optical, 3D)
One Example: Packet Scheduling

- **Which packet to choose for a given output port?**
  - Router needs to prioritize between competing flits
  - Which input port?
  - Which virtual channel?
  - Which application’s packet?

- **Common strategies**
  - Round robin across virtual channels
  - Oldest packet first (or an approximation)
  - Prioritize some virtual channels over others

- **Better policies in a multi-core environment**
  - Use application characteristics
  - Minimize energy