Summary of Last Week

- GPU Programming
- Multiprocessor Basics
- Memory Ordering (Consistency)
- Cache Coherence
Interconnection Networks
Readings

- **Required**

- **Recommended**
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
Terminology

- **Network interface**
  - Module that connects endpoints (e.g. processors) to network
  - Decouples computation/communication

- **Link**
  - Bundle of wires that carry a signal

- **Switch/router**
  - Connects fixed number of input channels to fixed number of output channels

- **Channel**
  - A single logical connection between routers/switches
More Terminology

- **Node**
  - A router/switch within a network

- **Message**
  - Unit of transfer for network’s clients (processors, memory)

- **Packet**
  - Unit of transfer for network

- **Flit**
  - Flow control digit
  - Unit of flow control within network
Some More Terminology

- **Direct or Indirect Networks**
  - Endpoints sit “inside” (direct) or “outside” (indirect) the network
  - E.g. mesh is direct; every node is both endpoint and switch

![Diagram of Direct and Indirect Networks]

- Router (switch), Radix of 2 (2 inputs, 2 outputs)
  - Abbreviation: Radix-ary
  - These routers are 2-ary
Interconnection Network
Topology
Properties of a Topology/Network

- **Regular or Irregular**
  - Regular if topology is regular graph (e.g. ring, mesh).

- **Routing Distance**
  - number of links/hops along a route

- **Diameter**
  - maximum routing distance within the network

- **Average Distance**
  - Average number of hops across all valid routes
Properties of a Topology/Network

- **Bisection Bandwidth**
  - Often used to describe network performance
  - Cut network in half and sum bandwidth of links severed
    - \((\text{Min # channels spanning two halves}) \times (\text{BW of each channel})\)
  - Meaningful only for recursive topologies
  - Can be misleading, because does not account for switch and routing efficiency (and certainly not execution time)

- **Blocking vs. Non-Blocking**
  - If connecting any permutation of sources & destinations is possible, network is non-blocking; otherwise network is blocking.
  - **Rearrangeable non-blocking**: Same as non-blocking but might require rearranging connections when switching from one permutation to another.
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
Bus

All nodes connected to a single link
+ 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 → reduced frequency)
- High contention → fast saturation
Point-to-Point

Every node connected to every other with direct/isolated links

+ Lowest contention
+ Potentially lowest latency
+ Ideal, if cost is no 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 with a shared link for each destination
- 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
```

**Blocking or Non-blocking?**
A multistage network has more restrictions on feasible concurrent Tx-Rx pairs vs a crossbar.

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 before transmission
  - Establish route then send data
  - No one else can use those links while “circuit” is set
  - Faster arbitration
  - No buffering
  -- setting up and bringing down “path” takes time

- **Packet switching** routes per packet in each router
  - Route each packet individually (possibly via different paths)
  - If link is free, any packet can use it
  -- potentially slower --- must dynamically switch
  -- need buffering
  + 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
Combining Operations in the Network

- **Idea:** Combine multiple operations on a shared memory location
- **Example:** Omega network switches combine fetch-and-add operations in NYU Ultracomputer
- **Fetch-and-add**(\(M, I\)): return \(M\), replace \(M\) with \(M+I\)
  - Common when parallel processors modify a shared variable, e.g. obtain a chunk of the array
- Combining reduces synchronization latency

![Diagram of combining fetch-and-adds](image-url)

```
TestAndSet(V)
{Temp ← V
  V ← TRUE}
RETURN Temp.
```

```
Fetch&OR(V, TRUE).
```

![Formula](image-url)
Butterfly

- Equivalent to Omega Network
- Indirect
- Used in BBN Butterfly
- Conflicts can cause “tree saturation”
  - Randomization of route selection helps
# Review: Topologies

<table>
<thead>
<tr>
<th>Topology</th>
<th>Crossbar</th>
<th>Multistage Logarith.</th>
<th>Mesh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct/Indirect</td>
<td>Indirect</td>
<td>Indirect</td>
<td>Direct</td>
</tr>
<tr>
<td>Blocking/Non-blocking</td>
<td>Non-blocking</td>
<td>Blocking</td>
<td>Blocking</td>
</tr>
<tr>
<td>Cost</td>
<td>O(N^2)</td>
<td>O(NlogN)</td>
<td>O(N)</td>
</tr>
<tr>
<td>Latency</td>
<td>O(1)</td>
<td>O(logN)</td>
<td>O(sqrt(N))</td>
</tr>
</tbody>
</table>
Ring

Each node connected to exactly two other nodes. Nodes form a continuous pathway such that packets can reach any node.

- 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

- Single directional pathway
- 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
Bidirectional Rings

Multi-directional pathways, or multiple rings

+ Reduces latency
+ Improves scalability

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

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

+ More scalable
+ Lower latency

- More complex

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


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

Design and Evaluation of Hierarchical Rings with Deflection Routing

Rachata Ausavarungnirun  Chris Fallin  Xiangyao Yu†  Kevin Kai-Wei Chang
Greg Nazario  Reetuparna Das§  Gabriel H. Loh‡  Onur Mutlu
Carnegie Mellon University  §University of Michigan  †MIT  ‡Advanced Micro Devices, Inc.
Mesh

- Each node connected to 4 neighbors (N, E, S, W)
- 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, continued

- Weave nodes to make inter-node latencies ~ constant
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

- “N-dimensional cube” or “N-cube”

- 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

Routing
Routing Mechanism

- **Arithmetic**
  - Simple arithmetic to determine route in regular topologies
  - Dimension order routing in meshes/tori

- **Source Based**
  - Source specifies output port for each switch in route
    + Simple switches
      - no control state: strip output port off header
    - Large header

- **Table Lookup Based**
  - Index into table for output port
    + Small header
    - More complex switches
Routing Algorithm

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

  + 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

Oblivious Routing: Valiant’s 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
More on Adaptive Routing

- Can avoid faulty links/routers

- Idea: Route around faults

+ Deterministic routing cannot handle faulty components
- Need to change the routing table to disable faulty routes
  - Assuming the faulty link/router is detected

One recent example:

Fattah et al., "A Low-Overhead, Fully-Distributed, Guaranteed-Delivery Routing Algorithm for Faulty Network-on-Chips", NOCS 2015.
Buffering and Flow Control
Recall: Circuit vs. Packet Switching

- **Circuit switching** sets up full path before transmission
  - Establish route then send data
  - No one else can use those links while “circuit” is set
  + faster arbitration
  -- setting up and bringing down “path” takes time

- **Packet switching** routes per packet in each router
  - 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
Packet Switched Networks: Packet Format

- Header
  - routing and control information
- Payload
  - carries data (non HW specific information)
  - can be further divided (framing, protocol stacks...)
- Error Code
  - generally at tail of packet so it can be generated on the way out

![Packet Format Diagram]
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?
Flow Control Methods

- Circuit switching
- Bufferless (Packet/flit based)
- Store and forward (Packet based)
- Virtual cut through (Packet based)
- Wormhole (Flit based)
Circuit Switching Revisited

- Resource allocation granularity is high

- Idea: Pre-allocate resources across multiple switches for a given “flow”

- Need to send a probe to set up the path for pre-allocation

  + No need for buffering
  + No contention (flow’s performance is isolated)
  + Can handle arbitrary message sizes

- Lower link utilization: two flows cannot use the same link
- Handshake overhead to set up a “circuit”
**Bufferless Deflection Routing**

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

---

Bufferless Deflection Routing

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

Issues In Bufferless Deflection Routing

- Livelock

- Resulting Router Complexity

- Performance & Congestion at High Loads

Store and Forward Flow Control

- Packet-based flow control
- Store and Forward
  - Packet copied entirely into network router before moving to the next node
  - Flow control unit is the entire packet
- Leads to high per-packet latency
- Requires buffering for entire packet in each node

Can we do better?

![Network Diagram]
Cut through Flow Control

- Another form of packet-based flow control
- Start forwarding as soon as header is received and resources (buffer, channel, etc) allocated
  - Dramatic reduction in latency
- Still allocate buffers and channel bandwidth for full packets

What if packets are large?
Cut through Flow Control

- What to do if output port is blocked?
- Lets the tail continue when the head is blocked, absorbing the whole message into a single switch.
  - Requires a buffer large enough to hold the largest packet.
- Degenerates to store-and-forward with high contention

- Can we do better?
Wormhole Flow Control

- Packets broken into (potentially) smaller flits (buffer/bw allocation unit)
- Flits are sent across the fabric in a wormhole fashion
  - Body follows head, tail follows body
  - Pipelined
  - If head blocked, rest of packet stops
  - Routing (src/dest) information only in head
- How does body/tail know where to go?
- Latency almost independent of distance for long messages
Wormhole Flow Control

- Advantages over “store and forward” flow control
  + Lower latency
  + More efficient buffer utilization

- Limitations
  - Suffers from **head of line blocking**
    - If head flit cannot move due to contention, another worm cannot proceed even though links may be idle
Head of Line Blocking

- A worm can be before another in the router input buffer
- Due to FIFO nature, the second worm cannot be scheduled even though it may need to access another output port
Head of Line Blocking

Red holds this channel: channel remains idle until read proceeds

Channel idle but red packet blocked behind blue

Buffer full: blue cannot proceed

Blocked by other packets
Virtual Channel Flow Control

- **Idea:** Multiplex multiple channels over one physical channel
- Divide up the input buffer into multiple buffers sharing a single physical channel
Virtual Channel Flow Control

- **Idea:** Multiplex multiple channels over one physical channel
- Divide up the input buffer into multiple buffers sharing a single physical channel

Figure 5: (A) Conventional nodes organize their buffers into FIFO queues restricting routing. (B) A network using virtual-channel flow control organizes its buffers into several independent lanes.
Virtual Channel Flow Control

Buffer full: blue cannot proceed

Blocked by other packets
A Modern Virtual Channel Based Router
Other Uses of Virtual Channels

- **Deadlock avoidance**
  - Enforcing switching to a different set of virtual channels on some “turns” can break the cyclic dependency of resources
  - Enforce order on VCs
  - **Escape VCs**: Have at least one VC that uses deadlock-free routing. Ensure each flit has fair access to that VC.
  - **Protocol level deadlock**: Ensure address and data packets use different VCs → prevent cycles due to intermixing of different packet classes

- **Prioritization of traffic classes**
  - Some virtual channels can have higher priority than others
Review: Flow Control

Store and Forward

Any other issues?

Head-of-Line Blocking

Use Virtual Channels
Review: Flow Control

**Store and Forward**
- Shrink Buffers
- Reduce latency

**Cut Through / Wormhole**

Any other issues?
- Head-of-Line Blocking
- Use Virtual Channels

Buffer full: blue cannot proceed
Blocked by other packets
We did not cover the following slides in lecture. These are for your preparation for the next lecture.
Communicating Buffer Availability

- Credit-based flow control
  - Upstream knows how many buffers are downstream
  - Downstream passes back credits to upstream
  - Significant upstream signaling (esp. for small flits)

- On/Off (XON/XOFF) flow control
  - Downstream has on/off signal to upstream

- Ack/Nack flow control
  - Upstream optimistically sends downstream
  - Buffer cannot be deallocated until ACK/NACK received
  - Inefficiently utilizes buffer space
Credit-based Flow Control

- Round-trip credit delay:
  - Time between when buffer empties and when next flit can be processed from that buffer entry
- Significant throughput degradation if there are few buffers
- Important to size buffers to tolerate credit turn-around
On/Off (XON/XOFF) Flow Control

- Downstream has on/off signal to upstream

- \( F_{off} \) threshold reached

- \( F_{on} \) threshold reached

- \( F_{off} \) set to prevent flits arriving before \( t_4 \) from overflowing

- \( F_{on} \) set so that Node 2 does not run out of flits between \( t_5 \) and \( t_8 \)
Interconnection Network Performance
Interconnection Network Performance

Latency

Injection rate into network

Throughput given by flow control
Throughput given by routing
Throughput given by topology

Zero load latency (topology + routing + flow control)
Min latency given by routing algorithm
Min latency given by topology
Ideal Latency

- Ideal latency
  - Solely due to wire delay between source and destination

\[ T_{\text{ideal}} = \frac{D}{v} + \frac{L}{b} \]

- D = Manhattan distance
  - The distance between two points measured along axes at right angles.
- v = propagation velocity
- L = packet size
- b = channel bandwidth
Actual Latency

- Dedicated wiring impractical
  - Long wires segmented with insertion of routers

\[ T_{\text{actual}} = \frac{D}{v} + \frac{L}{b} + H \cdot T_{\text{router}} + T_c \]

- D = Manhattan distance
- v = propagation velocity
- L = packet size
- b = channel bandwidth
- H = hops
- \( T_{\text{router}} \) = router latency
- \( T_c \) = latency due to contention
Latency and Throughput Curve

Latency (cycles)

Injected load (fraction of capacity)

Ideal
On-chip Network
Network Performance Metrics

- Packet latency
- Round trip latency
- Saturation throughput

- Application-level performance: system performance
  - Affected by interference among threads/applications
Buffering and Routing in On-Chip Networks
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

Processing Element
(Cores, L2 Banks, Memory Controllers, etc)
On-chip Networks

Router

Processing Element
(Cores, L2 Banks, Memory Controllers etc.)
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 → low latency
  - On chip RC lines → need repeaters every 1-2mm
    - Can put logic in repeaters

- **Workloads**
  - Multi-core cache traffic vs. supercomputer interconnect traffic
On-Chip vs. Off-Chip Tradeoffs

- George Nychis, Chris Fallin, Thomas Moscibroda, Onur Mutlu, and Srinivasan Seshan,

"On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-core Interconnects"

Proceedings of the 2012 ACM SIGCOMM Conference (SIGCOMM), Helsinki, Finland, August 2012. Slides (pptx)
Buffers in NoC Routers

- Buffers are necessary for high network throughput
  - Buffers increase total available bandwidth in network

![Graph showing the relationship between injection rate and average packet latency for small, medium, and large buffers.](image)
Buffers are necessary for high network throughput.
→ buffers increase total available bandwidth.

Buffers consume significant energy/power.

- Dynamic energy when read/write
- Static energy even when not occupied

Buffers add complexity and latency.

- Logic for buffer management
- Virtual channel allocation
- Credit-based flow control

Buffers require significant chip area.

E.g., in TRIPS prototype chip, input buffers occupy 75% of total on-chip network area [Gratz et al, ICCD ’06]
Going Bufferless...?

- How much throughput do we lose?
  → How is latency affected?

- Up to what injection rates can we use bufferless routing?
  → Are there realistic scenarios in which NoC is operated at injection rates below the threshold?

- Can we achieve energy reduction?
  → If so, how much...?

- Can we reduce area, complexity, etc...?

Answers in our paper!
BLESS: Bufferless Routing

- Always forward *all* incoming flits to some output port
- If no productive direction is available, send to another direction
- → packet is deflected
- → Hot-potato routing [Baran’ 64, etc]
BLESS: Bufferless Routing

1. Create a ranking over all incoming flits
2. For a given flit in this ranking, find the best free output-port
   Apply to each flit in order of ranking
FLIT-BLESS: Flit-Level Routing

- Each flit is routed independently.
- **Oldest-first arbitration** (other policies evaluated in paper)

<table>
<thead>
<tr>
<th>Flit-Ranking</th>
<th>Port-Prioritization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Oldest-first ranking</td>
<td>2. Assign flit to productive port, if possible. Otherwise, assign to non-productive port.</td>
</tr>
</tbody>
</table>

- **Network Topology:**
  - Can be applied to most topologies (Mesh, Torus, Hypercube, Trees, …)
    1) #output ports , #input ports at every router
    2) every router is reachable from every other router

- **Flow Control & Injection Policy:**
  - Completely local, inject whenever input port is free

- **Absence of Deadlocks:** every flit is always moving
- **Absence of Livelocks:** with oldest-first ranking
BLESS: Advantages & Disadvantages

**Advantages**
- No buffers
- Purely local flow control
- Simplicity
  - no credit-flows
  - no virtual channels
  - simplified router design
- No deadlocks, livelocks
- Adaptivity
  - packets are deflected around congested areas!
- Router latency reduction
- Area savings

**Disadvantages**
- Increased latency
- Reduced bandwidth
- Increased buffering at receiver
- Header information at each flit
- Oldest-first arbitration complex
- QoS becomes difficult

Impact on energy...?
Evaluation – Synthetic Traces

• First, the bad news 😊

• Uniform random injection

• BLESS has significantly lower saturation throughput compared to buffered baseline.
Evaluation – Homogenous Case Study

- milc benchmarks (moderately intensive)

- Perfect caches!

- Very little performance degradation with BLESS (less than 4% in dense network)

- With router latency 1, BLESS can even outperform baseline (by ~10%)

- Significant energy improvements (almost 40%)
Observations:

1) Injection rates not extremely high on average → self-throttling!

2) For bursts and temporary hotspots, use network links as buffers!

- Significant energy improvements (almost 40%)
BLESS Conclusions

• For a very wide range of applications and network settings, buffers are not needed in NoC
  • Significant energy savings (32% even in dense networks and perfect caches)
  • Area-savings of 60%
  • Simplified router and network design (flow control, etc…)
  • Performance slowdown is minimal (can even increase!)

➤ A strong case for a rethinking of NoC design!

• Future research:
  • Support for quality of service, different traffic classes, energy-management, etc…
Bufferless Routing in NoCs

  - https://users.ece.cmu.edu/~omutlu/pub/bless_isca09.pdf
Issues In Bufferless Deflection Routing

- Livelock
- Resulting Router Complexity
- Performance & Congestion at High Loads
- Quality of Service and Fairness

Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu,
"Bufferless and Minimally-Buffered Deflection Routing"
Low-Complexity Bufferless Routing

- Chris Fallin, Chris Craik, and Onur Mutlu,
  "CHIPPER: A Low-Complexity Bufferless Deflection Router"


CHIPPER: A Low-complexity Bufferless Deflection Router

Chris Fallin  
cfallin@cmu.edu

Chris Craik  
craik@cmu.edu

Onur Mutlu  
onur@cmu.edu

Computer Architecture Lab (CALCM)  
Carnegie Mellon University
CHIPPER: A Low-complexity Bufferless Deflection Router

Chris Fallin, Chris Craik, and Onur Mutlu,
"CHIPPER: A Low-Complexity Bufferless Deflection Router"
Motivation

- Recent work has proposed **bufferless deflection routing** (BLESS [Moscibroda, ISCA 2009])

  - **Energy savings**: ~40% in total NoC energy
  - **Area reduction**: ~40% in total NoC area
  - **Minimal performance loss**: ~4% on average

- Unfortunately: unaddressed complexities in router
  - long critical path, large reassembly buffers

- **Goal**: obtain these benefits while simplifying the router in order to **make bufferless NoCs practical**.
Problems that Bufferless Routers Must Solve

1. Must provide **livelock freedom**

   ➔ A packet should not be deflected forever

2. Must **reassemble packets** upon arrival

**Flit**: atomic routing unit

**Packet**: one or multiple flits

0 1 2 3
A Bufferless Router: A High-Level View

Problem 1: Livelock Freedom
Problem 2: Packet Reassembly

Deflection Routing Logic
Crossbar
Reassembly Buffers
Complexity in Bufferless Deflection Routers

1. Must provide livelock freedom

   Flits are sorted by age, then assigned in age order to output ports

   $\Rightarrow$ 43% longer critical path than buffered router

2. Must reassemble packets upon arrival

   Reassembly buffers must be sized for worst case

   $\Rightarrow$ 4KB per node
   
   (8x8, 64-byte cache block)
Problem 1: Livelock Freedom
Livelock Freedom in Previous Work

- What stops a flit from deflecting forever?
- All flits are *timestamped*
- *Oldest flits* are assigned their desired ports
- Total order among flits

New traffic is lowest priority

Guaranteed progress!

Flit age forms total order

- But what is the *cost* of this?
Age-Based Priorities are Expensive: Sorting

- Router must **sort flits by age**: long-latency sort network
  - Three comparator stages for 4 flits
Age-Based Priorities Are Expensive: Allocation

- After sorting, flits assigned to output ports in priority order
- Port assignment of younger flits depends on that of older flits
  - **sequential dependence** in the port allocator

<table>
<thead>
<tr>
<th>Age-Ordered Flits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>East?</td>
</tr>
</tbody>
</table>

**1.** GRANT: Flit 1 → East
\{N,S,W\}

**2.** DEFLECT: Flit 2 → North
\{S,W\}

**3.** GRANT: Flit 3 → South
\{W\}

**4.** DEFLECT: Flit 4 → West

SAFARI
Age-Based Priorities Are Expensive

- Overall, **deflection routing logic** based on **Oldest-First** has a **43% longer critical path** than a buffered router.

- Question: is there a cheaper way to route while guaranteeing livelock-freedom?
Solution: Golden Packet for Livelock Freedom

- What is *really necessary* for livelock freedom?

  **Key Insight:** No total order. It is enough to:
  1. Pick one flit to **prioritize until arrival**
  2. Ensure any flit is **eventually picked**

New traffic is lowest-priority, guaranteed progress!

Flit age forms total order, partial ordering is sufficient!

“Golden Flit”
What Does Golden Flit Routing Require?

- Only **need** to properly route the Golden Flit
- **First Insight:** no need for full sort
- **Second Insight:** no need for sequential allocation
Golden Flit Routing With Two Inputs

- Let’s route the Golden Flit in a two-input router first

- **Step 1**: pick a “winning” flit: Golden Flit, else random
- **Step 2**: steer the winning flit to its desired output and deflect other flit

→ Golden Flit is always routed toward its destination
Golden Flit Routing with Four Inputs

- Each block makes decisions independently!
- Deflection is a distributed decision

![Diagram of Golden Flit Routing with Four Inputs]
Permutation Network Operation

wins → swap!

wins → no swap!

Port Allocator

Priority Sort

SAFARI
Problem 2: Packet Reassembly
Reassembly Buffers are Large

- **Worst case**: every node sends a packet to one receiver
- Why can’t we make reassembly buffers smaller?

\[ N \text{ sending nodes} \]

\[ \text{Node 0} \rightarrow \text{Node 1} \rightarrow \ldots \rightarrow \text{Node } N-1 \]

one packet in flight per node

\[ \text{Receiver} \]

\( O(N) \) space!
Small Reassembly Buffers Cause Deadlock

- What happens when reassembly buffer is too small?

Many Senders

One Receiver

Remaining flits must be injected for forward progress

Network

cannot inject new traffic

network full

reassembly buffer

cannot eject: reassembly buffer full
Reserve Space to Avoid Deadlock?

- What if every sender asks permission from the receiver before it sends?

→ **adds additional delay** to every request

1. Reserve Slot
2. ACK
3. Send Packet
Escaping Deadlock with Retransmissions

- Sender is optimistic instead: assume buffer is free
  - If not, receiver drops and NACKs; sender retransmits

→ no additional delay in best case
→ transmit buffering overhead for all packets
→ potentially many retransmits
Solution: Retransmitting Only Once

- **Key Idea:** Retransmit only *when space becomes available.*
  - Receiver *drops packet* if full; *notes* which packet it drops
  - When space frees up, receiver *reserves space* so retransmit is successful
  - Receiver notifies sender to retransmit
Using MSHRs as Reassembly Buffers

Inject/Eject Reassembly Buffers

Miss Buffers (MSHRs)

Using miss buffers for reassembly makes this a truly bufferless network.
CHIPPER: Cheap Interconnect Partially-Permuting Router

Baseline Bufferless Deflection Router

Long critical path:
1. Sort by age
2. Allocate ports sequentially

→ Golden Packet
→ Permutation Network

Large buffers for worst case

→ Retransmit-Once
→ Cache miss buffers

Reassembly Buffers

Eject

Inject

Deflection

Routing

Logic

Crossbar

SAFARI
CHIPPER: Cheap Interconnect Partially-Permuting Router

Inject/Eject

Miss Buffers (MSHRs)
EVALUATION
Methodology

- **Multiprogrammed** workloads: CPU2006, server, desktop
  - 8x8 (64 cores), 39 homogeneous and 10 mixed sets

- **Multithreaded** workloads: SPLASH-2, 16 threads
  - 4x4 (16 cores), 5 applications

- **System configuration**
  - **Buffered** baseline: 2-cycle router, 4 VCs/channel, 8 flits/VC
  - **Bufferless** baseline: 2-cycle latency, FLIT-BLESS
    - Instruction-trace driven, closed-loop, 128-entry OoO window
    - 64KB L1, **perfect L2 (stresses interconnect)**, XOR mapping
Methodology

- **Hardware modeling**
  - Verilog models for CHIPPER, BLESS, buffered logic
    - Synthesized with commercial 65nm library
  - **ORION** for crossbar, buffers and links

- **Power**
  - Static and dynamic power from hardware models
  - Based on event counts in cycle-accurate simulations
Results: Performance Degradation

Minimal loss for low-to-medium-intensity workloads

3.6%  49.8%
Results: Power Reduction

Multiprogrammed (subset of 49 total)

- Buffered
- BLESS
- CHIPPER

Network Power (W)

- Results: Power Reduction
- Removing buffers ➔ majority of power savings
- Slight savings from BLESS to CHIPPER

Multithreaded

- 54.9%
- 73.4%
CHIPPER maintains area savings of BLESS

Critical path becomes competitive to buffered
Conclusions

- Two key issues in bufferless deflection routing
  - livelock freedom and packet reassembly

- Bufferless deflection routers were high-complexity and impractical
  - Oldest-first prioritization → long critical path in router
  - No end-to-end flow control for reassembly → prone to deadlock with reasonably-sized reassembly buffers

- CHIPPER is a new, practical bufferless deflection router
  - Golden packet prioritization → short critical path in router
  - Retransmit-once protocol → deadlock-free packet reassembly
  - Cache miss buffers as reassembly buffers → truly bufferless network

- CHIPPER frequency comparable to buffered routers at much lower area and power cost, and minimal performance loss
Chris Fallin, Chris Craik, and Onur Mutlu, "CHIPPER: A Low-Complexity Bufferless Deflection Router"


CHIPPER: A Low-complexity Bufferless Deflection Router

Chris Fallin  cfallin@cmu.edu  Chris Craik  craik@cmu.edu  Onur Mutlu  onur@cmu.edu

Computer Architecture Lab (CALCM)
Carnegie Mellon University
Minimally-Buffered Deflection Routing

- Bufferless deflection routing offers **reduced power & area**
- But, high deflection rate hurts **performance at high load**

- **MinBD** (Minimally-Buffered Deflection Router) introduces:
  - Side buffer to hold **only** flits that would have been deflected
  - Dual-width ejection to address ejection bottleneck
  - Two-level prioritization to avoid unnecessary deflections

- MinBD yields **reduced power (31%) & reduced area (36%)** relative to **buffered** routers
- MinBD yields **improved performance (8.1% at high load)** relative to **bufferless** routers \(\rightarrow\) closes half of perf. gap

- MinBD has the **best energy efficiency** of all evaluated designs with **competitive performance**
Minimally-Buffered Deflection Routing

- Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungrunr, and Onur Mutlu,
"MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect"
“Bufferless” Hierarchical Rings


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

Design and Evaluation of Hierarchical Rings with Deflection Routing

Rachata Ausavarungnirun  Chris Fallin  Xiangyao Yu†  Kevin Kai-Wei Chang
Greg Nazario  Reetuparna Das§  Gabriel H. Loh‡  Onur Mutlu

Carnegie Mellon University  §University of Michigan  †MIT  ‡Advanced Micro Devices, Inc.
“Bufferless” Hierarchical Rings (II)


Achieving both High Energy Efficiency and High Performance in On-Chip Communication using Hierarchical Rings with Deflection Routing

Rachata Ausavarungnirun  Chris Fallin  Xiangyao Yu†  Kevin Kai-Wei Chang
Greg Nazario  Reetuparna Das§  Gabriel H. Loh‡  Onur Mutlu
Carnegie Mellon University  §University of Michigan  †MIT  ‡AMD
Summary of Six Years of Research

- Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungrunrun, and Onur Mutlu,
  "Bufferless and Minimally-Buffered Deflection Routing"

Chapter 1
Bufferless and Minimally-Buffered Deflection Routing

Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungrunrun, Onur Mutlu
More Readings

- Studies of congestion and congestion control in on-chip vs. internet-like networks


- George Nychis, Chris Fallin, Thomas Moscibroda, and Onur Mutlu, "Next Generation On-Chip Networks: What Kind of Congestion Control Do We Need?" Proceedings of the 9th ACM Workshop on Hot Topics in Networks (HOTNETS), Monterey, CA, October 2010. Slides (ppt) (key)
On-Chip vs. Off-Chip Tradeoffs

- George Nychis, Chris Fallin, Thomas Moscibroda, Onur Mutlu, and Srinivasan Seshan,
"On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-core Interconnects"
Proceedings of the 2012 ACM SIGCOMM Conference (SIGCOMM), Helsinki, Finland, August 2012. Slides (pptx)

On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-Core Interconnects

George Nychis†, Chris Fallin†, Thomas Moscibroda§, Onur Mutlu†, Srinivasan Seshan†
† Carnegie Mellon University
§ Microsoft Research Asia
{gnychis,cfallin,onur,srini}@cmu.edu
moscitho@microsoft.com
HAT: Heterogeneous Adaptive Throttling for On-Chip Networks

Kevin Chang, Rachata Ausavarungnirun, Chris Fallin, and Onur Mutlu,
"HAT: Heterogeneous Adaptive Throttling for On-Chip Networks"
Executive Summary

• **Problem:** Packets contend in on-chip networks (NoCs), causing congestion, thus reducing performance

• **Observations:**
  1) Some applications are more sensitive to network latency than others
  2) Applications must be throttled differently to achieve peak performance

• **Key Idea:** Heterogeneous Adaptive Throttling (HAT)
  1) Application-aware source throttling
  2) Network-load-aware throttling rate adjustment

• **Result:** Improves performance and energy efficiency over state-of-the-art source throttling policies
Throttling Based Fairness in NoCs

- Kevin Chang, Rachata Ausavarungnirun, Chris Fallin, and Onur Mutlu, "HAT: Heterogeneous Adaptive Throttling for On-Chip Networks"

HAT: Heterogeneous Adaptive Throttling for On-Chip Networks

Kevin Kai-Wei Chang, Rachata Ausavarungnirun, Chris Fallin, Onur Mutlu
Carnegie Mellon University
{kevincha, rachata, cfallin, onur}@cmu.edu
MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect

Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu,
"MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect"
Bufferless Deflection Routing

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

- Removing **buffers** yields significant benefits
  - Reduces **power** (CHIPPER: reduces NoC power by 55%)
  - Reduces **die area** (CHIPPER: reduces NoC area by 36%)

- But, at **high network utilization** (load), bufferless deflection routing causes **unnecessary link & router traversals**
  - Reduces network throughput and application performance
  - Increases dynamic power

- **Goal**: Improve high-load performance of low-cost deflection networks by reducing the deflection rate.
Outline: This Talk

- Motivation

- **Background**: Bufferless Deflection Routing

- **MinBD**: Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration

- Results

- Conclusions
Outline: This Talk

- Motivation

- **Background**: Bufferless Deflection Routing
  - MinBD: Reducing Deflections
    - Addressing Link Contention
    - Addressing the Ejection Bottleneck
    - Improving Deflection Arbitration

- Results
- Conclusions
Issues in Bufferless Deflection Routing

- **Correctness**: Deliver all packets without livelock
  - CHIPPER\(^1\): Golden Packet
  - Globally prioritize one packet until delivered

- **Correctness**: Reassemble packets without deadlock
  - CHIPPER\(^1\): Retransmit-Once

- **Performance**: Avoid performance degradation at high load
  - MinBD

\(^1\) Fallin et al., “CHIPPER: A Low-complexity Bufferless Deflection Router”, HPCA
Key Performance Issues

1. **Link contention**: no buffers to hold traffic → any link contention causes a deflection → use side buffers

2. **Ejection bottleneck**: only one flit can eject per router per cycle → simultaneous arrival causes deflection → eject up to 2 flits/cycle

3. **Deflection arbitration**: practical (fast) deflection arbiters deflect unnecessarily → new priority scheme (silver flit)
Outline: This Talk

- **Motivation**

- **Background:** Bufferless Deflection Routing

- **MinBD:** Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration

- **Results**

- **Conclusions**
Outline: This Talk

- Motivation

**Background**: Bufferless Deflection Routing

**MinBD**: Reducing Deflections
- Addressing Link Contention
- Addressing the Ejection Bottleneck
- Improving Deflection Arbitration

- Results
- Conclusions
Addressing Link Contention

- **Problem 1**: Any link contention causes a deflection

- **Buffering** a flit can avoid deflection on contention

- But, **input buffers** are expensive:
  - All flits are buffered on every hop → high dynamic energy
  - Large buffers necessary → high static energy and large area

- **Key Idea 1**: add a small buffer to a bufferless deflection router to buffer only flits that would have been deflected
How to Buffer Deflected Flits

Baseline Router

How to Buffer Deflected Flits

Step 1. Remove up to one deflected flit per cycle from the outputs.

Step 2. Buffer this flit in a small FIFO “side buffer.”

Step 3. Re-inject this flit into pipeline when a slot is available.
Why Could A Side Buffer Work Well?

- Buffer some flits and deflect other flits at per-flit level

  - Relative to **bufferless routers**, deflection rate reduces (need not deflect all contending flits)
    - 4-flit buffer reduces deflection rate by 39%

  - Relative to **buffered routers**, buffer is more efficiently used (need not buffer all flits)
    - similar performance with 25% of buffer space
Outline: This Talk

- Motivation

- **Background**: Bufferless Deflection Routing

- **MinBD**: Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration

- Results

- Conclusions
Addressing the Ejection Bottleneck

**Problem 2:** Flits deflect unnecessarily because only one flit can **eject** per router per cycle.

- In 20% of all ejections, $\geq 2$ flits could have ejected, $\rightarrow$ all but one flit must **deflect** and try again.
  $\rightarrow$ these deflected flits cause additional contention.

- Ejection width of 2 flits/cycle reduces **deflection rate 21%**.

**Key idea 2:** Reduce deflections due to a single-flit ejection port by allowing **two flits** to eject per cycle.
Addressing the Ejection Bottleneck

Single-Width Ejection
Addressing the Ejection Bottleneck

For fair comparison, **baseline routers** have dual-width ejection for perf. (not power/area)
Outline: This Talk

- Motivation
- **Background**: Bufferless Deflection Routing
- **MinBD**: Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration
- Results
- Conclusions
Improving Deflection Arbitration

Problem 3: Deflections occur unnecessarily because fast arbiters must use simple priority schemes.

- Age-based priorities (several past works): full priority order gives fewer deflections, but requires slow arbiters.

- State-of-the-art deflection arbitration (Golden Packet & two-stage permutation network):
  - Prioritize one packet globally (**ensure forward progress**)
  - Arbitrate other flits randomly (**fast critical path**)

- Random common case leads to **uncoordinated arbitration**
Let’s route in a two-input router first:

- **Step 1:** pick a “winning” flit (Golden Packet, else random)
- **Step 2:** steer the winning flit to its desired output and deflect other flit

→ Highest-priority flit always routes to destination
Fast Deflection Routing with Four Inputs

- Each block makes decisions independently
- Deflection is a distributed decision
Unnecessary Deflections in Fast Arbiters

- How does lack of coordination cause unnecessary deflections?
  1. No flit is golden (pseudorandom arbitration)
  2. Red flit wins at first stage
  3. Green flit loses at first stage (must be deflected now)
  4. Red flit loses at second stage; Red and Green are deflected
Key idea 3: Add a priority level and prioritize one flit to ensure at least one flit is not deflected in each cycle

Highest priority: one Golden Packet in network
- Chosen in static round-robin schedule
- Ensures correctness

Next-highest priority: one silver flit per router per cycle
- Chosen pseudo-randomly & local to one router
- Enhances performance
Adding A Silver Flit

- Randomly picking a silver flit ensures **one flit is not deflected**
  1. No flit is golden but **Red** flit is silver
  2. **Red** flit wins at first stage (silver)
  3. **Green** flit is deflected at first stage
  4. **Red** flit wins at second stage (silver); not deflected

![Diagram of network with flits and priorities]

- At least one flit is not deflected
- Red flits have higher priority
- All flits have equal priority
Minimally-Buffered Deflection Router

**Problem 1:** Link Contention
**Solution 1:** Side Buffer

**Problem 2:** Ejection Bottleneck
**Solution 2:** Dual-Width Ejection

**Problem 3:** Unnecessary Deflections
**Solution 3:** Two-level priority scheme
Outline: This Talk

- Motivation

- Background: Bufferless Deflection Routing

- MinBD: Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration
Outline: This Talk

- Motivation

- Background: Bufferless Deflection Routing

- MinBD: Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration

- Results
- Conclusions
Methodology: Simulated System

- **Chip Multiprocessor Simulation**
  - **64-core** and **16-core** models
  - **Closed-loop** core/cache/NoC cycle-level model
  - Directory cache coherence protocol (SGI Origin-based)
  - 64KB L1, perfect L2 (stresses interconnect), XOR-mapping
  - Performance metric: **Weighted Speedup**
    (similar conclusions from network-level latency)
  - Workloads: multiprogrammed SPEC CPU2006
    - 75 randomly-chosen workloads
    - Binned into network-load categories by average injection rate
Methodology: Routers and Network

- **Input-buffered** virtual-channel router
  - 8 VCs, 8 flits/VC [Buffered(8,8)]: large buffered router
  - 4 VCs, 4 flits/VC [Buffered(4,4)]: typical buffered router
  - 4 VCs, 1 flit/VC [Buffered(4,1)]: smallest deadlock-free router
  - All power-of-2 buffer sizes up to (8, 8) for perf/power sweep

- **Bufferless deflection** router: CHIPPER\(^1\)

- **Bufferless-buffered hybrid** router: AFC\(^2\)
  - Has input buffers and deflection routing logic
  - Performs coarse-grained (multi-cycle) mode switching

- **Common parameters**
  - 2-cycle router latency, 1-cycle link latency
  - 2D-mesh topology (16-node: 4x4; 64-node: 8x8)
  - Dual ejection assumed for baseline routers (for perf. only)

---

\(^1\)Fallin et al., “CHIPPER: A Low-complexity Bufferless Deflection Router”, HPCA 2011.

Methodology: Power, Die Area, Crit. Path

- **Hardware modeling**
  - **Verilog models** for CHIPPER, MinBD, buffered control logic
    - Synthesized with commercial 65nm library
  - **ORION 2.0** for datapath: crossbar, muxes, buffers and links

- **Power**
  - Static and dynamic power from hardware models
  - Based on event counts in cycle-accurate simulations
  - Broken down into buffer, link, other
1. All mechanisms individually reduce deflections.
2. Side buffer alone is not sufficient for performance (ejection bottleneck remains).
3. Overall, **5.8%** over baseline, **2.7%** over dual-eject by reducing deflections **64% / 54%**.
Overall Performance Results

- Similar perf. to Buffered (4,1) @ 25% of buffering space
- Within 2.7% of Buffered (4,4) (8.3% at high load)

Weighted Speedup

0.0 - 0.15
0.15 - 0.30
0.30 - 0.40
0.40 - 0.50
> 0.50
AVG

Buffered (4,1)
MinBD-4
Overall Power Results

- Dynamic power increases with deflection routing.
- There are significant reductions in dynamic power in baseline routers.
- Dynamic power reduces in MinBD relative to CHIPPER.
Performance-Power Spectrum

- **Most energy-efficient** (perf/watt) of any evaluated network router design

MinBD  Buf (4,1)  AFC  CHIPPER

Buf (1,1)  Buf (4,4)  Buf (8,8)
Die Area and Critical Path

- Only **3%** area increase over CHIPPER (4-flit buffer)
- Increases by **7%** over CHIPPER, **8%** over Buffered (4,4)
Conclusions

- Bufferless deflection routing offers **reduced power & area**
- But, high deflection rate hurts **performance at high load**

- **MinBD** (Minimally-Buffered Deflection Router) introduces:
  - Side buffer to hold **only** flits that would have been deflected
  - Dual-width ejection to address ejection bottleneck
  - Two-level prioritization to avoid unnecessary deflections

- MinBD yields **reduced power (31%) & reduced area (36%)** relative to **buffered** routers
- MinBD yields **improved performance (8.1% at high load)** relative to **bufferless** routers → closes half of perf. gap

- MinBD has the **best energy efficiency** of all evaluated designs with **competitive performance**
Minimally-Buffered Deflection Routing

Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu,
"MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect"
Packet Scheduling
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
Application-Aware Packet Scheduling

The Problem: Packet Scheduling

Network-on-Chip is a critical resource shared by multiple applications
The Problem: Packet Scheduling

- **Routers** (R)
- **Processing Element** (PE) (Cores, L2 Banks, Memory Controllers etc)

Diagram:
- Crossbar *(5 x 5)*
- Input Port with Buffers
- Control Logic
  - Routing Unit (RC)
  - VC Allocator (VA)
  - Switch Allocator (SA)
- From East, West, North, South, From PE to To East, West, North, South, PE
The Problem: Packet Scheduling
The Problem: Packet Scheduling

Routing Unit (RC)

VC Allocator (VA)

Switch Allocator (SA)
The Problem: Packet Scheduling

Which packet to choose?
The Problem: Packet Scheduling

- Existing scheduling policies
  - Round Robin
  - Age
- Problem 1: Local to a router
  - Lead to contradictory decision making between routers: packets from one application may be prioritized at one router, to be delayed at next.
- Problem 2: Application oblivious
  - Treat all applications packets equally
  - But applications are heterogeneous
- Solution: Application-aware global scheduling policies.
STC Scheduling Example

Packet Injection Order at Processor

Batching interval length = 3 cycles

Ranking order =

Core1   Core2   Core3

Batch 0

Batch 1

Batch 2
STC Scheduling Example

Applications

Batch 0

- 1
- 2
- 3

Batch 1

- 3
- 4
- 5
- 6

Batch 2

- 7
- 8

Injection Cycles

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8

Router

Scheduler

1

2

3

4

5

6

7

8
STC Scheduling Example

**Router**

5
8 4
3
7 1
6 2
2
3 2

**Round Robin**

3 2 8 7 6

**STALL CYCLES**

<table>
<thead>
<tr>
<th>RR</th>
<th>8</th>
<th>6</th>
<th>11</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
STC Scheduling Example

Router

5
8
4

3
7
1

6
2

3
2

Round Robin

5
4
3
1
2
2
3
2
8
7
6

Age

3
3
5
4
6
7
8

Time

Time

STALL CYCLES | Avg
--- | ---
RR | 8 | 6 | 11 | 8.3
Age | 4 | 6 | 11 | 7.0
STC |
STC Scheduling Example

STALL CYCLES

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR</td>
<td>8 6 11</td>
<td>8.3</td>
</tr>
<tr>
<td>Age</td>
<td>4 6 11</td>
<td>7.0</td>
</tr>
<tr>
<td>STC</td>
<td>1 3 11</td>
<td>5.0</td>
</tr>
</tbody>
</table>
Application-Aware Prioritization in NoCs


Application-Aware Prioritization Mechanisms for On-Chip Networks

Reetuparna Das§  Onur Mutlu†  Thomas Moscibroda†  Chita R. Das§
§Pennsylvania State University  †Carnegie Mellon University  ‡Microsoft Research
{rdas,das}@cse.psu.edu  onur@cmu.edu  moscitho@microsoft.com
Slack-Based Packet Scheduling

Low-Cost QoS in On-Chip Networks (I)

- Boris Grot, Stephen W. Keckler, and Onur Mutlu, "Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QoS Scheme for Networks-on-Chip"


Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QoS Scheme for Networks-on-Chip

Boris Grot
Department of Computer Sciences
The University of Texas at Austin
{bgrot, skeckler}@cs.utexas.edu

Stephen W. Keckler

Onur Mutlu†
†Computer Architecture Laboratory (CALCM)
Carnegie Mellon University
onur@cmu.edu
Low-Cost QoS in On-Chip Networks (II)

- Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu, "Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees"
  Proceedings of the 38th International Symposium on Computer Architecture (ISCA), San Jose, CA, June 2011. Slides (pptx)

Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees

Boris Grot\textsuperscript{1} bgrot@cs.utexas.edu
Joel Hestness\textsuperscript{1} hestness@cs.utexas.edu
Stephen W. Keckler\textsuperscript{1,2} skeckler@nvidia.com
Onur Mutlu\textsuperscript{3} onur@cmu.edu

\textsuperscript{1}The University of Texas at Austin
Austin, TX
\textsuperscript{2}NVIDIA
Santa Clara, CA
\textsuperscript{3}Carnegie Mellon University
Pittsburgh, PA
Kilo-NoC: Topology-Aware QoS

Motivation

- Extreme-scale chip-level integration
  - Cores
  - Cache banks
  - Accelerators
  - I/O logic
  - Network-on-chip (NOC)
- 10-100 cores today
- 1000+ assets in the near future
Kilo-NOC requirements

- High efficiency
  - Area
  - Energy
- Good performance
- Strong service guarantees (QoS)
Problem: QoS support in each router is expensive (in terms of buffering, arbitration, bookkeeping)


Goal: Provide QoS guarantees at low area and power cost

Idea:

- Isolate shared resources in a region of the network, support QoS within that area
- Design the topology so that applications can access the region without interference
Baseline QOS-enabled CMP

Multiple VMs sharing a die

- Shared resources (e.g., memory controllers)
- VM-private resources (cores, caches)
- QOS-enabled router
Conventional NOC QOS

Contetion scenarios:
- Shared resources
  - memory access
- Intra-VM traffic
  - shared cache access
- Inter-VM traffic
  - VM page sharing
Conventional NOC QOS

Contention scenarios:

- Shared resources
  - memory access
- Intra-VM traffic
  - shared cache access
- Inter-VM traffic
  - VM page sharing

Network-wide guarantees *without* network-wide QOS support
Kilo-NOC QOS

- Insight: leverage rich network connectivity
  - Naturally reduce interference among flows
  - Limit the extent of hardware QOS support

- Requires a low-diameter topology
  - This work: Multidrop Express Channels (MECS)

Grot et al., HPCA 2009
**Topology-Aware QOS**

- Dedicated, QOS-enabled regions
  - Rest of die: QOS-free
- Richly-connected topology
  - Traffic isolation
- Special routing rules
  - Manage interference
Dedicated, QOS-enabled regions
- Rest of die: QOS-free

Richly-connected topology
- Traffic isolation

Special routing rules
- Manage interference
Topology-Aware QOS

- Dedicated, QOS-enabled regions
  - Rest of die: QOS-free
- Richly-connected topology
  - Traffic isolation
- Special routing rules
  - Manage interference
Topology-Aware QOS

- Dedicated, QOS-enabled regions
  - Rest of die: QOS-free
- Richly-connected topology
  - Traffic isolation
- Special routing rules
  - Manage interference
Kilo-NOC view

- Topology-aware QOS support
  - Limit QOS complexity to a fraction of the die
- Optimized flow control
  - Reduce buffer requirements in QOS-free regions
## Evaluation Methodology

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>15 nm</td>
</tr>
<tr>
<td>Vdd</td>
<td>0.7 V</td>
</tr>
<tr>
<td>System</td>
<td>1024 tiles:</td>
</tr>
<tr>
<td></td>
<td>256 concentrated nodes (64 shared resources)</td>
</tr>
<tr>
<td>Networks:</td>
<td></td>
</tr>
<tr>
<td>MECS+PVC</td>
<td>VC flow control, QOS support (PVC) at each node</td>
</tr>
<tr>
<td>MECS+TAQ</td>
<td>VC flow control, QOS support only in shared regions</td>
</tr>
<tr>
<td>MECS+TAQ+EB</td>
<td>EB flow control outside of SRs, Separate Request and Reply networks</td>
</tr>
<tr>
<td>K-MECS</td>
<td>Proposed organization: TAQ + hybrid flow control</td>
</tr>
</tbody>
</table>
Area comparison

The chart compares the area requirements of different configurations:

- **MECS+PVC**
- **MECS+TAQ**
- **MECS+TAQ+EB**
- **K-MECS**

The areas are categorized into:

- SR Routers
- Routers
- Link EBs
- Links

The area requirements are shown in square millimeters (mm²).
Energy comparison

![Graph showing energy comparison between different network configurations.](image)

- **MECS**
- **MECS EB**
- **MECS hybrid**

**Average packet latency (cycles)**

**Load (%)**

- MECS+PVC
- MECS+TAQ
- MECS+EB+TAQ
- K-MECS

**Legend**

- SR Routers
- Routers
- Link EBs
- Links
Summary

Kilo-NOC: a heterogeneous NOC architecture for kilo-node substrates

- Topology-aware QOS
  - Limits QOS support to a fraction of the die
  - Leverages low-diameter topologies
  - Improves NOC area- and energy-efficiency
  - Provides strong guarantees
Low-Cost QoS in On-Chip Networks (II)

- Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu, "Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees"
  Proceedings of the 38th International Symposium on Computer Architecture (ISCA), San Jose, CA, June 2011. Slides (pptx)

Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees

Boris Grot\textsuperscript{1}
bgrot@cs.utexas.edu

Joel Hestness\textsuperscript{1}
hestness@cs.utexas.edu

Stephen W. Keckler\textsuperscript{1,2}
skeckler@nvidia.com

Onur Mutlu\textsuperscript{3}
onur@cmu.edu

\textsuperscript{1}The University of Texas at Austin
Austin, TX

\textsuperscript{2}NVIDIA
Santa Clara, CA

\textsuperscript{3}Carnegie Mellon University
Pittsburgh, PA
Express-Cube Topologies

Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu,
"Express Cube Topologies for On-Chip Interconnects"
Slides (ppt)
2-D Mesh
2-D Mesh

- **Pros**
  - Low design & layout complexity
  - Simple, fast routers

- **Cons**
  - Large diameter
  - Energy & latency impact
Concentration *(Balfour & Dally, ICS ‘06)*

**Pros**
- Multiple *terminals* attached to a router node
- Fast nearest-neighbor communication via the crossbar
- Hop count reduction proportional to *concentration* degree

**Cons**
- Benefits limited by crossbar complexity
Concentration

- Side-effects
  - Fewer channels
  - Greater channel width
Replication

Benefits

- Restores bisection channel count
- Restores channel width
- Reduced crossbar complexity

CMesh-X2
Flattened Butterfly (Kim et al., Micro '07)

- Objectives:
  - Improve connectivity
  - Exploit the wire budget
Flattened Butterfly \cite{Kim07}(Kim et al., Micro 07)
Flattened Butterfly (Kim et al., Micro '07)
Flattened Butterfly *(Kim et al., Micro '07)*
Flattened Butterfly \textit{(Kim et al., Micro 07)}
Flattened Butterfly (Kim et al., Micro 07)

- **Pros**
  - Excellent connectivity
  - Low diameter: 2 hops

- **Cons**
  - High channel count: $k^2/2$ per row/column
  - Low channel utilization
  - Increased control (arbitration) complexity
Multidrop Express Channels (MECS)

- Objectives:
  - Connectivity
  - More scalable channel count
  - Better channel utilization
Multidrop Express Channels (MECS)
Multidrop Express Channels (MECS)
Multidrop Express Channels (MECS)
Multidrop Express Channels (MECS)
Multidrop Express Channels (MECS)
Multidrop Express Channels (MECS)

- **Pros**
  - One-to-many topology
  - Low diameter: 2 hops
  - $k$ channels row/column
  - Asymmetric

- **Cons**
  - Asymmetric
  - Increased control (arbitration) complexity
Partitioning: a GEC Example

**MECS**

**MECS-X2**

**Partitioned MECS**

**Flattened Butterfly**
## Analytical Comparison

<table>
<thead>
<tr>
<th></th>
<th>CMesh</th>
<th>FBfly</th>
<th>MECS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network Size</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>256</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td><strong>Radix (conctr’d)</strong></td>
<td>4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td><strong>Diameter</strong></td>
<td>6</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td><strong>Channel count</strong></td>
<td>2</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td><strong>Channel width</strong></td>
<td>576</td>
<td>1152</td>
<td>144</td>
</tr>
<tr>
<td><strong>Router inputs</strong></td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td><strong>Router outputs</strong></td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>
## Experimental Methodology

<table>
<thead>
<tr>
<th>Topologies</th>
<th>Mesh, CMesh, CMesh-X2, FBFly, MECS, MECS-X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network sizes</td>
<td>64 &amp; 256 terminals</td>
</tr>
<tr>
<td>Routing</td>
<td>DOR, adaptive</td>
</tr>
<tr>
<td>Messages</td>
<td>64 &amp; 576 bits</td>
</tr>
<tr>
<td>Synthetic traffic</td>
<td>Uniform random, bit complement, transpose, self-similar</td>
</tr>
<tr>
<td>PARSEC benchmarks</td>
<td>Blackscholes, Bodytrack, Canneal, Ferret, Fluidanimate, Freqmine, Vip, x264</td>
</tr>
<tr>
<td>Full-system config</td>
<td>M5 simulator, Alpha ISA, 64 OOO cores</td>
</tr>
<tr>
<td>Energy evaluation</td>
<td>Orion + CACTI 6</td>
</tr>
</tbody>
</table>
64 nodes: Uniform Random

Latency (cycles) vs. injection rate (%)

- mesh
- cmesh
- cmesh-x2
- fbfly
- mecs
- mecs-x2
256 nodes: Uniform Random

Latency (cycles)

Injection rate (%)

- mesh
- cmesh-x2
- fbfly
- mecs
- mecs-x2
Energy (100K pkts, Uniform Random)

Average packet energy (nJ)

- **Link Energy**
- **Router Energy**

### Comparison for different topologies:
- mesh
- cmesh
- cmesh-x2
- fbfly
- mecs
- mecs-x2

**For 64 nodes:**
- Mesh
- CMesh
- CMesh-x2
- FBfly
- MECS
- MECS-x2

**For 256 nodes:**
- Mesh
- CMesh
- CMesh-x2
- FBfly
- MECS
- MECS-x2
64 Nodes: PARSEC

- **Total network Energy (J)**
- **Avg packet latency (cycles)**

**Graph Legend:**
- Red: Router Energy
- Green: Link Energy
- Blue: latency

**Applications:**
- Blackscholes
- Canneal
- Vip
- x264
Summary

- **MECS**
  - A new one-to-many topology
  - Good fit for planar substrates
  - Excellent connectivity
  - Effective wire utilization

- **Generalized Express Cubes**
  - Framework & taxonomy for NOC topologies
  - Extension of the k-ary n-cube model
  - Useful for understanding and exploring on-chip interconnect options
  - Future: expand & formalize
Scalability: Express Cube Topologies

- Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu, "Express Cube Topologies for On-Chip Interconnects"


Express Cube Topologies for On-Chip Interconnects

Boris Grot    Joel Hestness    Stephen W. Keckler    Onur Mutlu†

Department of Computer Sciences
The University of Texas at Austin
{bgrot, hestness, skeckler}@cs.utexas.edu

†Computer Architecture Laboratory (CALCM)
Carnegie Mellon University
onur@cmu.edu
Interconnect Readings
Application-Aware Prioritization in NoCs

Slack-Based Packet Scheduling

Low-Cost QoS in On-Chip Networks (I)

- Boris Grot, Stephen W. Keckler, and Onur Mutlu, "Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-Chip"


Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-Chip

Boris Grot
Department of Computer Sciences
The University of Texas at Austin
{bgrot, skeckler@cs.utexas.edu}

Stephen W. Keckler

Onur Mutlu†
†Computer Architecture Laboratory (CALCM)
Carnegie Mellon University
onur@cmu.edu
Low-Cost QoS in On-Chip Networks (II)

- Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu, "Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees"
  Proceedings of the 38th International Symposium on Computer Architecture (ISCA), San Jose, CA, June 2011. Slides (pptx)

Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees

Boris Grot¹
bgrot@cs.utexas.edu

Joel Hestness¹
hestness@cs.utexas.edu

Stephen W. Keckler¹,²
skeckler@nvidia.com

Onur Mutlu³
onur@cmu.edu

¹The University of Texas at Austin
Austin, TX

²NVIDIA
Santa Clara, CA

³Carnegie Mellon University
Pittsburgh, PA
Throttling Based Fairness in NoCs

- Kevin Chang, Rachata Ausavarungnirun, Chris Fallin, and Onur Mutlu, "HAT: Heterogeneous Adaptive Throttling for On-Chip Networks"
Scalability: Express Cube Topologies


Express Cube Topologies for On-Chip Interconnects

Boris Grot Joel Hestness Stephen W. Keckler Onur Mutlu†

Department of Computer Sciences The University of Texas at Austin \{bgrot, hestness, skeckler\}@cs.utexas.edu

†Computer Architecture Laboratory (CALCM) Carnegie Mellon University onur@cmu.edu
Scalability: Slim NoC

Maciej Besta, Syed Minhaj Hassan, Sudhakar Yalamanchili, Rachata Ausavarungnirun, Onur Mutlu, Torsten Hoefler,
"Slim NoC: A Low-Diameter On-Chip Network Topology for High Energy Efficiency and Scalability"
[Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pdf)]

Slim NoC: A Low-Diameter On-Chip Network Topology for High Energy Efficiency and Scalability

Maciej Besta\textsuperscript{1} Syed Minhaj Hassan\textsuperscript{2} Sudhakar Yalamanchili\textsuperscript{2} Rachata Ausavarungnirun\textsuperscript{3} Onur Mutlu\textsuperscript{1,3} Torsten Hoefler\textsuperscript{1}

\textsuperscript{1}ETH Zürich  \textsuperscript{2}Georgia Institute of Technology  \textsuperscript{3}Carnegie Mellon University
Bufferless Routing in NoCs

  - [https://users.ece.cmu.edu/~omutlu/pub/bless_isca09.pdf](https://users.ece.cmu.edu/~omutlu/pub/bless_isca09.pdf)

---

A Case for Bufferless Routing in On-Chip Networks

Thomas Moscibroda  
Microsoft Research  
moscitho@microsoft.com

Onur Mutlu  
Carnegie Mellon University  
onur@cmu.edu
CHIPPER: Low-Complexity Bufferless

- Chris Fallin, Chris Craik, and Onur Mutlu,
  "CHIPPER: A Low-Complexity Bufferless Deflection Router"

CHIPPER: A Low-complexity Bufferless Deflection Router

Chris Fallin  
cfallin@cmu.edu

Chris Craik  
craik@cmu.edu

Onur Mutlu  
onur@cmu.edu

Computer Architecture Lab (CALCM)  
Carnegie Mellon University
Minimally-Buffered Deflection Routing

- Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu,
  "MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect"


MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect

Chris Fallin, Greg Nazario, Xiangyao Yu†, Kevin Chang, Rachata Ausavarungnirun, Onur Mutlu

Carnegie Mellon University
{cfallin,gnazario,kevincha,rachata,onur}@cmu.edu

†Tsinghua University & Carnegie Mellon University
yxythu@gmail.com
“Bufferless” Hierarchical Rings


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

---

**Design and Evaluation of Hierarchical Rings with Deflection Routing**

Rachata Ausavarungnirun  Chris Fallin  Xiangyao Yu†  Kevin Kai-Wei Chang
Greg Nazario  Reetuparna Das§  Gabriel H. Loh‡  Onur Mutlu

Carnegie Mellon University  §University of Michigan  †MIT  ‡Advanced Micro Devices, Inc.
“Bufferless” Hierarchical Rings (II)


Achieving both High Energy Efficiency and High Performance in On-Chip Communication using Hierarchical Rings with Deflection Routing

Rachata Ausavarungnirun  Chris Fallin  Xiangyao Yu†  Kevin Kai-Wei Chang
Greg Nazario  Reetuparna Das§  Gabriel H. Loh‡  Onur Mutlu
Carnegie Mellon University  §University of Michigan  †MIT  ‡AMD
Summary of Six Years of Research

- Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu, "Bufferless and Minimally-Buffered Deflection Routing"

Chapter 1
Bufferless and Minimally-Buffered Deflection Routing

Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, Onur Mutlu
On-Chip vs. Off-Chip Tradeoffs

- George Nychis, Chris Fallin, Thomas Moscibroda, Onur Mutlu, and Srinivasan Seshan,

"On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-core Interconnects"

Proceedings of the 2012 ACM SIGCOMM Conference (SIGCOMM), Helsinki, Finland, August 2012. Slides (pptx)

On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-Core Interconnects

George Nychis†, Chris Fallin†, Thomas Moscibroda§, Onur Mutlu†, Srinivasan Seshan†

† Carnegie Mellon University
{gnychis,cfallin,onor,srini}@cmu.edu

§ Microsoft Research Asia
moscitho@microsoft.com
A Model for Application Slowdown Estimation in On-Chip Networks and Its Use for Improving System Fairness and Performance

Xiyue Xiang†, Saugata Ghose‡, Onur Mutlu§‡, Nian-Feng Tzeng†

†University of Louisiana at Lafayette  ‡Carnegie Mellon University  §ETH Zürich
Handling Multicast and Hotspot Issues

- Xiyue Xiang, Wentao Shi, Saugata Ghose, Lu Peng, Onur Mutlu, and Nian-Feng Tzeng,

"Carpool: A Bufferless On-Chip Network Supporting Adaptive Multicast and Hotspot Alleviation"

Proceedings of the International Conference on Supercomputing (ICS), Chicago, IL, USA, June 2017.
[Slides (pptx) (pdf)]
Computer Architecture
Lecture 23: Interconnects

Prof. Onur Mutlu
ETH Zürich
Fall 2018
12 December 2018
More Readings

- Studies of congestion and congestion control in on-chip vs. internet-like networks


- George Nychis, Chris Fallin, Thomas Moscibroda, and Onur Mutlu, "Next Generation On-Chip Networks: What Kind of Congestion Control Do We Need?" Proceedings of the 9th ACM Workshop on Hot Topics in Networks (HOTNETS), Monterey, CA, October 2010. Slides (ppt) (key)
HAT: Heterogeneous Adaptive Throttling for On-Chip Networks

Kevin Chang, Rachata Ausavarungnirun, Chris Fallin, and Onur Mutlu,
"HAT: Heterogeneous Adaptive Throttling for On-Chip Networks"
Executive Summary

• **Problem**: Packets contend in on-chip networks (NoCs), causing congestion, thus reducing performance

• **Observations**:
  1) Some applications are more sensitive to network latency than others
  2) Applications must be throttled differently to achieve peak performance

• **Key Idea**: Heterogeneous Adaptive Throttling (HAT)
  1) Application-aware source throttling
  2) Network-load-aware throttling rate adjustment

• **Result**: Improves performance and energy efficiency over state-of-the-art source throttling policies
Outline

• **Background and Motivation**
• Mechanism
• Prior Works
• Results
On-Chip Networks

- Connect **cores, caches, memory controllers, etc**
- **Packet switched**
- **2D mesh:** Most commonly used topology
- Primarily serve **cache misses** and **memory requests**
- **Router designs**
  - Buffered: **Input buffers** to hold contending packets
  - Bufferless: **Misroute (deflect)** contending packets

**Diagram:**

- **Router**
- **Processing Element**
  - (Cores, L2 Banks, Memory Controllers, etc)
Network Congestion Reduces Performance

Limited shared resources (buffers and links)
- Design constraints: power, chip area, and timing

Network congestion:
- Network throughput
- Application performance

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

- **Improve performance in a highly congested NoC**

- Reducing network load decreases network congestion, hence improves performance

- **Approach: source throttling to reduce network load**
  - Temporarily delay new traffic injection

- **Naïve mechanism: throttle every single node**
Key Observation #1

Different applications respond differently to changes in network latency

**gromacs**: network-non-intensive

**mcf**: network-intensive

Throttling network-intensive applications benefits system performance more
Key Observation #2

Different workloads achieve peak performance at different throttling rates

Dynamically adjusting throttling rate yields better performance than a single static rate
Outline

• Background and Motivation

• **Mechanism**

• Prior Works

• Results
Heterogeneous Adaptive Throttling (HAT)

1. **Application-aware throttling:**
   Throttle **network-intensive** applications that interfere with **network-non-intensive** applications

2. **Network-load-aware aware throttling rate adjustment:**
   Dynamically adjusts throttling rate to adapt to different workloads
Heterogeneous Adaptive Throttling (HAT)

1. **Application-aware throttling:**
   Throttle **network-intensive** applications that interfere with **network-non-intensive** applications

2. **Network-load-aware throttling rate adjustment:**
   Dynamically adjusts throttling rate to adapt to different workloads
Application-Aware Throttling

1. **Measure Network Intensity**
   Use **L1 MPKI** (misses per thousand instructions) to estimate network intensity

2. **Classify Application**
   Sort applications by **L1 MPKI**
   
   - **Network-non-intensive**
   - **Network-intensive**

   - **$\Sigma MPKI < \text{NonIntensiveCap}$**
   - Higher **L1 MPKI**

3. **Throttle network-intensive applications**
Heterogeneous Adaptive Throttling (HAT)

1. **Application-aware throttling:**
   Throttle network-intensive applications that interfere with network-non-intensive applications

2. **Network-load-aware throttling rate adjustment:**
   Dynamically adjusts throttling rate to adapt to different workloads
Dynamic Throttling Rate Adjustment

• For a given network design, peak performance tends to occur at a fixed network load point

• Dynamically adjust throttling rate to achieve that network load point
Dynamic Throttling Rate Adjustment

- **Goal**: maintain network load at a peak performance point

1. **Measure network load**
2. **Compare and adjust throttling rate**
   - If *network load* > *peak point*:
     - Increase throttling rate
   - *elif network load* ≤ *peak point*:
     - Decrease throttling rate
Epoch-Based Operation

- Continuous **HAT** operation is expensive
- **Solution:** performs **HAT** at epoch granularity

**During epoch:**
1) Measure **L1 MPKI** of each application
2) Measure **network load**

**Beginning of epoch:**
1) Classify applications
2) Adjust throttling rate
3) Reset measurements

---

**Current Epoch** *(100K cycles)*

**Next Epoch** *(100K cycles)*
Outline

• Background and Motivation
• Mechanism
• Prior Works
• Results
Prior Source Throttling Works

• **Source throttling for bufferless NoCs**
  [Nychis+ Hotnets’10, SIGCOMM’12]
  – Application-aware throttling based on starvation rate
  – Does not adaptively adjust throttling rate
  – “Heterogeneous Throttling”

• **Source throttling off-chip buffered networks**
  [Thottethodi+ HPCA’01]
  – Dynamically trigger throttling based on fraction of buffer occupancy
  – Not application-aware: fully block packet injections of every node
  – “Self-tuned Throttling”
Outline

• Background and Motivation
• Mechanism
• Prior Works
• Results
Methodology

• **Chip Multiprocessor Simulator**
  – 64-node multi-core systems with a **2D-mesh topology**
  – Closed-loop core/cache/NoC cycle-level model
  – 64KB L1, perfect L2 (always hits to stress NoC)

• **Router Designs**
  – Virtual-channel buffered router: 4 VCs, 4 flits/VC [Dally+ IEEE TPDS’92]
  – Bufferless deflection routers: BLESS [Moscibroda+ ISCA’09]

• **Workloads**
  – 60 multi-core workloads: SPEC CPU2006 benchmarks
  – Categorized based on their network intensity
    • Low/Medium/High intensity categories

• **Metrics:** Weighted Speedup (perf.), perf./Watt (energy eff.), and maximum slowdown (fairness)
HAT provides better performance improvement than past work

Highest improvement on heterogeneous workload mixes
- L and M are more sensitive to network latency
Performance: Buffered NoC

Congestion is much lower in Buffered NoC, but HAT still provides performance benefit
HAT provides better fairness than prior works.
Network Energy Efficiency

HAT increases energy efficiency by reducing congestion
Other Results in Paper

• Performance on CHIPPER

• Performance on multithreaded workloads

• Parameters sensitivity sweep of HAT
Conclusion

• **Problem**: Packets contend in on-chip networks (NoCs), causing congestion, thus reducing performance

• **Observations**:
  1) Some applications are more sensitive to network latency than others
  2) Applications must be throttled differently to achieve peak performance

• **Key Idea**: Heterogeneous Adaptive Throttling (HAT)
  1) Application-aware source throttling
  2) Network-load-aware throttling rate adjustment

• **Result**: Improves performance and energy efficiency over state-of-the-art source throttling policies
Throttling Based Fairness in NoCs

Kevin Chang, Rachata Ausavarungrunrun, Chris Fallin, and Onur Mutlu, "HAT: Heterogeneous Adaptive Throttling for On-Chip Networks"

HAT: Heterogeneous Adaptive Throttling for On-Chip Networks

Kevin Kai-Wei Chang, Rachata Ausavarungrunrun, Chris Fallin, Onur Mutlu
Carnegie Mellon University
{kevincha, rachata, cfallin, onur}@cmu.edu
Slack-Driven Packet Scheduling

Packet Scheduling in NoC

- Existing scheduling policies
  - Round robin
  - Age

- Problem
  - Treat all packets equally
  - Application-oblivious

- Packets have different criticality
  - Packet is critical if latency of a packet affects application’s performance
  - Different criticality due to memory level parallelism (MLP)
MLP Principle

Packet Latency $\neq$ Network Stall Time

Different Packets have different criticality due to MLP

Criticality(Compute) $>$ Criticality(Stall) $>$ Criticality(Network Stall Time)
Outline

- Introduction
  - Packet Scheduling
  - Memory Level Parallelism
- Aérgia
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion
What is Áégia?

- Áégia is the spirit of laziness in Greek mythology
- Some packets can afford to slack!
Outline

- Introduction
  - Packet Scheduling
  - Memory Level Parallelism
- Aergia
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion
Slack of Packets

- What is slack of a packet?
  - Slack of a packet is number of cycles it can be delayed in a router without (significantly) reducing application’s performance
  - **Local network slack**

- Source of slack: Memory-Level Parallelism (MLP)
  - Latency of an application’s packet hidden from application due to overlap with latency of pending cache miss requests

- Prioritize packets with lower slack
Concept of Slack

Slack \( (\text{slack}) \) = Latency \( (\text{latency}) \) – Latency \( (\text{latency}) \) = 26 – 6 = 20 hops

Packet\( (\text{packet}) \) can be delayed for available slack cycles without reducing performance!
Prioritizing using Slack

Core A
Load Miss Causes
Load Miss Causes

Core B
Load Miss Causes
Load Miss Causes

Packet | Latency | Slack
--- | --- | ---
13 hops | 0 hops
3 hops | 10 hops

- Interference at 3 hops
- Slack(■) > Slack (■)
- Prioritize

Causes
Load Miss
Load Miss
Slack in Applications

- 50% of packets have 350+ slack cycles
- 10% of packets have <50 slack cycles

Percentage of all Packets (%) vs. Slack in cycles

- Non-critical
- Critical
Slack in Applications

68% of packets have zero slack cycles
Diversity in Slack

Percentage of all Packets (%)

Slack in cycles

Gems
omnet
tpcw
mcf
bzip2
sjbb
sap
sphinx
deal
barnes
art
libquantum
sjeng
h264ref
Diversity in Slack

Slack varies \textbf{between} packets of \textit{different} applications

Slack varies \textbf{between} packets of \textit{a single} application
Outline

- Introduction
  - Packet Scheduling
  - Memory Level Parallelism
- Aérgia
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion
Estimating Slack Priority

\[ \text{Slack (P)} = \max (\text{Latencies of P’s Predecessors}) - \text{Latency of P} \]

**Predecessors(P)** are the packets of outstanding cache miss requests when P is issued

- Packet latencies not known when issued

- Predicting latency of any packet Q
  - Higher latency if Q corresponds to an L2 miss
  - Higher latency if Q has to travel farther number of hops
Estimating Slack Priority

- Slack of $P = \text{Maximum Predecessor Latency} - \text{Latency of } P$

- $\text{Slack}(P) = \text{PredL2 (2 bits)} - \text{MyL2 (1 bit)} + \text{HopEstimate (2 bits)}$

**PredL2**: Set if any predecessor packet is servicing L2 miss

**MyL2**: Set if $P$ is NOT servicing an L2 miss

**HopEstimate**: $\text{Max (Number of hops of Predecessors)} - \text{hops of } P$
Estimating Slack Priority

- How to predict L2 hit or miss at core?
  - Global Branch Predictor based L2 Miss Predictor
    - Use Pattern History Table and 2-bit saturating counters
  - Threshold based L2 Miss Predictor
    - If \#L2 misses in “M” misses >= “T” threshold then next load is a L2 miss.

- Number of miss predecessors?
  - List of outstanding L2 Misses

- Hops estimate?
  - Hops => \(\Delta X + \Delta Y\) distance
  - Use predecessor list to calculate slack hop estimate
Starvation Avoidance

- Problem: **Starvation**
  - Prioritizing packets can lead to starvation of lower priority packets

- Solution: **Time-Based Packet Batching**
  - New batches are formed at every $T$ cycles
  - Packets of older batches are prioritized over younger batches
Putting it all together

- Tag header of the packet with priority bits before injection

**Priority (P) =**

- **Priority(P)?**
  - P’s batch *(highest priority)*
  - P’s Slack
  - Local Round-Robin *(final tie breaker)*
Outline

- Introduction
  - Packet Scheduling
  - Memory Level Parallelism
- Aergia
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion
Evaluation Methodology

- 64-core system
  - x86 processor model based on Intel Pentium M
  - 2 GHz processor, 128-entry instruction window
  - 32KB private L1 and 1MB per core shared L2 caches, 32 miss buffers
  - 4GB DRAM, 320 cycle access latency, 4 on-chip DRAM controllers

- Detailed Network-on-Chip model
  - 2-stage routers (with speculation and look ahead routing)
  - Wormhole switching (8 flit data packets)
  - Virtual channel flow control (6 VCs, 5 flit buffer depth)
  - 8x8 Mesh (128 bit bi-directional channels)

- Benchmarks
  - Multiprogrammed scientific, server, desktop workloads (35 applications)
  - 96 workload combinations
Qualitative Comparison

- **Round Robin & Age**
  - Local and application oblivious
  - Age is biased towards heavy applications

- **Globally Synchronized Frames (GSF)**
  [Lee et al., ISCA 2008]
  - Provides *bandwidth fairness* at the expense of *system performance*
  - Penalizes heavy and bursty applications

- **Application-Aware Prioritization Policies (SJF)**
  [Das et al., MICRO 2009]
  - **Shortest-Job-First Principle**
  - Packet scheduling policies which prioritize network sensitive applications which inject lower load
System Performance

- SJF provides 8.9% improvement in weighted speedup
- Aergia improves system throughput by 10.3%
- Aergia+SJF improves system throughput by 16.1%
Network Unfairness

- SJF does not imbalance network fairness
- Aergia improves network unfairness by 1.5X
- SJF+Aergia improves network unfairness by 1.3X
Conclusions & Future Directions

- Packets have different criticality, yet existing packet scheduling policies treat all packets equally.
- We propose a new approach to packet scheduling in NoCs.
  - We define Slack as a key measure that characterizes the relative importance of a packet.
  - We propose Aer gia, a novel architecture to accelerate low slack critical packets.
- Result
  - Improves system performance: 16.1%
  - Improves network fairness: 30.8%
Slack-Based Packet Scheduling


Aérgia: Exploiting Packet Latency Slack in On-Chip Networks

Reetuparna Das§ Onur Mutlu† Thomas Moscibroda‡ Chita R. Das§
§Pennsylvania State University {rdas,das}@cse.psu.edu †Carnegie Mellon University onur@cmu.edu ‡Microsoft Research moscitho@microsoft.com