Single-Cycle Collective Communication Over A Shared Network Fabric

Tushar Krishna \(^1\)  
\(^1\)Intel Corporation  
VSSAD Group,  
Hudson, MA, USA

Li-Shiuan Peh \(^2\)  
\(^2\)Massachusetts Institute of Technology  
Department of EECS,  
Cambridge, MA, USA

8th International Symposium on Networks-on-Chip (NOCS 2014)
Collective Communication

- **Shared Memory Cache Coherence**
  - 1-to-Many: snoopy and limited-directory
  - Many-to-1: ACKnowledgments

- **MPI**
  - 1-to-Many: *MPI_Bcast*
  - Many-to-1: *MPI_Reduce/MPI_BARRIER*
Collective Communication over a NoC

Throughput
× Up to $M$ times bandwidth consumption

Latency
× High network contention ($M$ packets at src/dst links)
NoC support for 1-to-M/M-to-1

Throughput
✓ Each link utilized exactly once per flow

Latency
✓ Less contention
✓ Single-Cycle Per-Hop fork and reduce [4, 5, 7]
  ✓ Runtime within 1% of magical contention-free network
× Always need to send to/receive from other end of chip → hops

[1] VCTM (ISCA ‘08)
[2] bLBDR (MICRO ‘08)
[3] MRR (HPCA ‘09)
[4] RPM (NOCS ‘09)
[6] BAM (HPCA ‘12)
[8] Com (HPCA ‘12)
Problem: Latency $\alpha$ Hops

Hop = tile-to-tile distance

This work: remove the dependence of latency on hops for 1-to-Many and Many-to-1 flows
Outline

- Motivation
- Background: SMART NoC
- SMART FanOut for 1-to-Many flows
- SMART FanIn for Many-to-1 flows
- Evaluation
- Conclusions
Outline

- Motivation
- **Background: SMART NoC**
- SMART FanOut for 1-to-Many flows
- SMART FanIn for Many-to-1 flows
- Evaluation
- Conclusions
What limits us from designing a 1-cycle network?

**Is it the wires? No!**

- *Global repeated wires* can cross 10-16 mm within 1ns
  - 10-16 hops (1mm tiles) within 1GHz cycle
  - $HPC_{max}$ (Hops Per Cycle max)

- *Global repeated wire delay expected to remain fairly constant*
  - *Chip dimensions expected to remain similar* (yield)
  - *Clock frequency expected to remain similar* (power wall)
Background: SMART NoC

What limits us from designing a 1-cycle network?

Is it the wires? No!

Is it the routers? Yes!

Routers force flits to stop (latch) at every hop, and incur 1-1.5ns delay (in the best case)

Solution: decouple wires from routers
**Background: SMART NoC**

**SMART**: replace clocked link drivers at each router by asynchronous repeaters. Send flits multiple hops within a cycle.

*Single-cycle Multi-hop Asynchronous Repeated Traversal [1, 2, 3]*

---

**Baseline: Single-cycle router at each hop**

- R0
- L0
- ...
- L4

Cycle 0
Cycle 1
Cycle 9

**SMART: Single-cycle across multi-hops**

- R0
- L0\|L1\|L2\|L3\|L4

Cycle 0
Cycle 1

---

Background: SMART NoC

Assume $HPC_{max} = 3$

**SSR**: SMART Setup Request

Router transmits SSR=$H$ if it wants to send a flit $H$-hops.
SMART Example : R0 → R3

Cycle 1: R0 sends SSR = 3

Assume $HPC_{max} = 3$

SSR: SMART Setup Request

Router transmits SSR=$H$ if it wants to send a flit $H$-hops.
SMART Example: R0 → R3

Cycle 1: R0 sends SSR = 3

All routers set buffer, mux and xbar for this request.

SSR: SMART Setup Request
In case of contention for a link segment, one of the flit waits, and one gets to go (local decision at each router based on SSRs that cycle).

SMART paths are opportunistic, not guaranteed.
Goal of this work

- Single-Cycle Multi-Hop NoC for Collective Communication
  - Dynamically create virtual 1-to-M and M-to-1 tree routes over a physical mesh
  - Traverse multiple nodes of the tree within a single-cycle (per-dimension)

- Challenge: fork/reduce while on a SMART path
  - SMART: Bypass router pipeline completely
  - NoC for Collective Communication: Stop at router to fork/reduce
Outline

- Motivation
- Background: SMART NoC
  - SMART FanOut for 1-to-Many flows
  - SMART FanIn for Many-to-1 flows
- Evaluation
- Conclusions
SMART FanOut (SFO): Routing

Shared Virtual Tree (SVT):
Contention among 1-to-M flows
SMART FanOut (SFO): Routing

- **Step 0:** Unicast to closest CR (Corner Router)
- **Step 1:** Broadcast along *straight dimension*
- **Step 2:** Broadcast along *turn dimension*
- **Step 3:** Deliver to NIC

September 17, 2014
**SMART FanOut (SFO): Routing**

- **Step 0:** Unicast to closest CR (Corner Router)
- **Step 1:** Broadcast along *straight dimension*
- **Step 2:** Broadcast along *turn dimension*
- **Step 3:** Deliver to NIC

**Private Virtual Tree (PVT):** No contention among 1-to-M flows

<table>
<thead>
<tr>
<th>CR</th>
<th>straight (Step 1)</th>
<th>turn (Step 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>E</td>
</tr>
<tr>
<td>W</td>
<td>W</td>
<td>S</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>N</td>
</tr>
</tbody>
</table>
SMART FanOut (SFO): Flow Control

- **SFO_Complete**
  - Guaranteed single-cycle broadcast per dimension
    - At fixed intervals [“broadcast interval (BI)’’], reserve 2 cycles for Step 1 and Step 2
    - SSRs (for unicasts) disabled during these 2 cycles
    - Path preset for single-cycle broadcast per-dimension
      - $buffer = 1$ and $mux = \text{bypass}$

- **SFO_Greedy**
  - Opportunistic SMART paths over a SVT or PVT
    - Extra $is\_bcast$ bit in SSR sets $buffer = 1$ and $mux = \text{bypass}$
  - Multi-cycle broadcast if contention
Outline

- Motivation
- Background: SMART NoC
- SMART FanOut for 1-to-Many flows
- SMART FanIn for Many-to-1 flows
- Evaluation
- Conclusions
**SMART FanIn (SFI)**

*Conceptually*, each ACK waits at a router till it has aggregated all ACKs from same flow that will pass through the router.

**ACKs from same M-to-1 flow carry same ACK_id**
- Assigned by the multicast (1-to-M) source (CR)
- Multicast flit setups up `num_ACK` according to ACK route (e.g., XY)
- ACKs injected with same ACK_id

---

**SSR Stage:**

```
if (ART[ACK_id].num_ACK != 1)
  drop Flit;
else
  buffer/SMART-bypass;
```

**Flit Stage:**

```
ART[ACK_id].num_ACK--;  
```

**Reduction Logic**

<table>
<thead>
<tr>
<th>ACK_id</th>
<th>reserved</th>
<th>num_ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Number of ACKs to wait for**

---

**Smart-hop Setup Request (SSR)**

- SSR carries extra `is_ACK` and `ACK_id` fields
- **ACK Reduction Table (ART)** – at every router
SFI Example: Step -1

**SSR Stage:**
if (ART[ACK_id].num_ACK != 1) drop Flit;
else buffer/SMART-bypass;

**Flit Stage:**
ART[ACK_id].num_ACK--;

- **Step -1:**
  1-to-M flit sets up ART
- **Step 0:**
  ACK injection
- **Step 1:**
  ACK X Traversal
- **Step 2:**
  ACK Y Traversal
- **Step 3:**
  Deliver to NIC
**SFI Example: Step 0 (SSR)**

**SSR Stage:**
if (ART[ACK_id].num_ACK != 1)
drop Flit;
else
buffer/SMART-bypass;

**Flit Stage:**
ART[ACK_id].num_ACK--;

- **Step -1:**
  1-to-M flit sets up ART

- **Step 0:**
  ACK injection

- **Step 1:**
  ACK X Traversal

- **Step 2:**
  ACK Y Traversal

- **Step 3:**
  Deliver to NIC
SFI Example: Step 0 (Flit)

- **SSR Stage:**
  - if (ART[ACK_id].num_ACK != 1)
    - drop Flit;
  - else
    - buffer/SMART-bypass;

- **Flit Stage:**
  - ART[ACK_id].num_ACK--;

- **Step 1:**
  - 1-to-M flit sets up ART

- **Step 0:**
  - ACK injection

- **Step 1:**
  - ACK X Traversal

- **Step 2:**
  - ACK Y Traversal

- **Step 3:**
  - Deliver to NIC

Only one active ACK per row ➞ lesser SSR conflicts
**SFI Example: Step 1 (SSR)**

---

**SSR Stage:**
if (ART[ACK_id].num_ACK != 1)
drop Flit;
else
buffer/SMART-bypass;

**Flit Stage:**
ART[ACK_id].num_ACK--;

---

- **Step -1:** 1-to-M flit sets up ART
- **Step 0:** ACK injection
- **Step 1:** ACK X Traversal
- **Step 2:** ACK Y Traversal
- **Step 3:** Deliver to NIC
**SFI Example: Step 1 (Flit)**

**SSR Stage:**

\[
\text{if} \ (\text{ART}[\text{ACK_id}].\text{num_ACK} \neq 1) \ \text{drop Flit;}
\]

\[
\text{else} \quad \text{buffer/SMART-bypass;}
\]

**Flit Stage:**

\[
\text{ART}[\text{ACK_id}].\text{num_ACK}--; \]

- **Step -1:** 1-to-M flit sets up ART
- **Step 0:** ACK injection
- **Step 1:** ACK X Traversal
- **Step 2:** ACK Y Traversal
- **Step 3:** Deliver to NIC
**SFI Example: Step 2 (SSR)**

SSR Stage:
\[
\text{if (ART[ACK_id].num_ACK != 1)} \\
\text{drop Flit;}
\]
else
\[
\text{buffer/SMART-bypass;}
\]

Flit Stage:
\[
\text{ART[ACK_id].num_ACK--;}
\]

- **Step -1:**
  1-to-M flit sets up ART
- **Step 0:**
  ACK injection
- **Step 1:**
  ACK X Traversal
- **Step 2:**
  ACK Y Traversal
- **Step 3:**
  Deliver to NIC
SFI Example: Step 2 (Flit)

- **Step 1:**
  1-to-M flit sets up ART
- **Step 0:**
  ACK injection
- **Step 1:**
  ACK X Traversal
- **Step 2:**
  ACK Y Traversal
- **Step 3:**
  Deliver to NIC

**SSR Stage:**
\[
\text{if} \ (\text{ART}[\text{ACK}_id].\text{num ACK} \neq 1) \quad \text{drop Flit}; \\
\text{else} \quad \text{buffer/SMART-bypass;}
\]

**Flit Stage:**
\[
\text{ART}[\text{ACK}_id].\text{num ACK}--; \\
\]

![Diagram](image)
SFI Example: Step 3 (Flit)

SSR Stage:
\[
\text{if (ART[ACK_id].num_ACK != 1)} \quad \text{drop Flit} ;
\]
\[
\text{else} \quad \text{buffer/SMART-bypass} ;
\]

Flit Stage:
\[
\text{ART[ACK_id].num_ACK} --;
\]

- **Step -1**: 1-to-M flit sets up ART
- **Step 0**: ACK injection
- **Step 1**: ACK X Traversal
- **Step 2**: ACK Y Traversal
- **Step 3**: Deliver to NIC

One ACK representing 24 SSR Flit
Handling Arbitrary Arrival Times

SSR Stage:
if (ART[ACK_id].num_ACK != 1)
drop Flit;
else
buffer/SMART-bypass;

Flit Stage:
ART[ACK_id].num_ACK--;

- **Step -1:**
  1-to-M flit sets up ART

- **Step 0:**
  ACK injection

- **Step 1:**
  ACK X Traversal

- **Step 2:**
  ACK Y Traversal

- **Step 3:**
  Deliver to NIC
Handling Arbitrary Arrival Times

SSR Stage:
if (ART[ACK_id].num_ACK != 1)
drop Flit;
else
buffer/SMART-bypass;

Flit Stage:
ART[ACK_id].num_ACK--;

- **Step -1:**
  1-to-M flit sets up ART

- **Step 0:**
  ACK injection

- **Step 1:**
  ACK X Traversal

- **Step 2:**
  ACK Y Traversal

- **Step 3:**
  Deliver to NIC
Handling Arbitrary Arrival Times

**SSR Stage:**
if (ART[ACK_id].num_ACK != 1)
drop Flit;
else
buffer/SMART-bypass;

**Flit Stage:**
ART[ACK_id].num_ACK--;

- **Step -1:**
  1-to-M flit sets up ART

- **Step 0:**
  ACK injection

- **Step 1:**
  ACK X Traversal

- **Step 2:**
  ACK Y Traversal

- **Step 3:**
  Deliver to NIC
Handling Arbitrary Arrival Times

SSR Stage:
if (ART[ACK_id].num_ACK != 1)
drop Flit;
else
buffer/SMART-bypass;

Flit Stage:
ART[ACK_id].num_ACK--;

- **Step -1:**
  1-to-M flit sets up ART

- **Step 0:***
  ACK injection

- **Step 1:**
  ACK X Traversal

- **Step 2:**
  ACK Y Traversal

- **Step 3:**
  Deliver to NIC
Handling Arbitrary Arrival Times

SSR Stage:
if (ART[ACK_id].num_ACK != 1)
    drop Flit;
else
    buffer/SMART-bypass;

Flit Stage:
ART[ACK_id].num_ACK--;

- **Step -1:** 1-to-M flit sets up ART
- **Step 0:** ACK injection
- **Step 1:** ACK X Traversal
- **Step 2:** ACK Y Traversal
- **Step 3:** Deliver to NIC
Handling Arbitrary Arrival Times

SSR Stage:
if (ART[ACK_id].num_ACK != 1)
drop Flit;
else
buffer/SMART-bypass;

Flit Stage:
ART[ACK_id].num_ACK--;

- **Step -1:**
  1-to-M flit sets up ART

- **Step 0:**
  ACK injection

- **Step 1:**
  ACK X Traversal

- **Step 2:**
  ACK Y Traversal

- **Step 3:**
  Deliver to NIC
Handling Arbitrary Arrival Times

SSR Stage:
\[ \text{if} \ (\text{ART}[\text{ACK}_id].\text{num}_\text{ACK} \neq 1) \ \text{drop Flit;} \]
\[ \text{else} \ \text{buffer/SMART-bypass;} \]

Flit Stage:
\[ \text{ART}[\text{ACK}_id].\text{num}_\text{ACK}--; \]

- **Step -1:**
  1-to-M flit sets up ART

- **Step 0:**
  ACK injection

- **Step 1:**
  ACK X Traversal

- **Step 2:**
  ACK Y Traversal

- **Step 3:**
  Deliver to NIC
Handling Arbitrary Arrival Times

SSR Stage:
if (ART[ACK_id].num_ACK != 1)
drop Flit;
else
buffer/SMART-bypass;

Flit Stage:
ART[ACK_id].num_ACK--;

- **Step -1:**
  1-to-M flit sets up ART

- **Step 0:**
  ACK injection

- **Step 1:**
  ACK X Traversal

- **Step 2:**
  ACK Y Traversal

- **Step 3:**
  Deliver to NIC
Handling Arbitrary Arrival Times

**SSR Stage:**
if (ART[ACK_id].num_ACK != 1)
drop Flit;
else
buffer/SMART-bypass;

**Flit Stage:**
ART[ACK_id].num_ACK--;

- **Step -1:**
  1-to-M flit sets up ART

- **Step 0:**
  ACK injection

- **Step 1:**
  ACK X Traversal

- **Step 2:**
  ACK Y Traversal

- **Step 3:**
  Deliver to NIC
Handling Arbitrary Arrival Times

SSR Stage:
if (ART[ACK_id].num_ACK != 1)
  drop Flit;
else
  buffer/SMART-bypass;

Flit Stage:
ART[ACK_id].num_ACK--;

- Step 1:
  1-to-M flit sets up ART

- Step 0:
  ACK injection

- Step 1:
  ACK X Traversal

- Step 2:
  ACK Y Traversal

- Step 3:
  Deliver to NIC
Number of ART entries

- How do we guarantee unique ACK_ids?
  - Each source can assign out of a distinct set ACK_ids
  - Min size of ART = 4 for PVT

- What if source runs out of ART entries?
  - it marks ACK_id as invalid
    - ACKs for this flow not reduced
  - More ART entries are required for performance, not correctness
SMART FanIn (SFI): Flow Control

- SFI_Complete
  - Presented so far
  - ART guarantees only 1 ACK per flow will reach destination NIC

- SFI_Greedy
  - Opportunistically aggregate ACKs from same flow while waiting at a router trying to arbitrate for the switch
  - No ART required
Outline

- Motivation
- Background: SMART NoC
- SMART FanOut for 1-to-Many flows
- SMART FanIn for Many-to-1 flows
- Evaluation
- Conclusions
## Evaluation

<table>
<thead>
<tr>
<th>Simulation Infrastructure</th>
<th>Full-System GEMS + Garnet (NoC) + DSENT (Energy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>45nm, 1.0V $V_{DD}$, 1.0 GHz</td>
</tr>
</tbody>
</table>

### CPU and Memory

<table>
<thead>
<tr>
<th>Processors</th>
<th>64-core in-order SPARC</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Caches</td>
<td>Private I&amp;D 32kB</td>
</tr>
<tr>
<td>L2 Caches</td>
<td>Private 1MB per core</td>
</tr>
<tr>
<td>Coherence Protocol</td>
<td>Limited directory (modeled similar to AMD HT)</td>
</tr>
<tr>
<td>Memory Controllers</td>
<td>8 (each corner and center of each edge)</td>
</tr>
</tbody>
</table>

### On-chip Network

<table>
<thead>
<tr>
<th>Topology</th>
<th>8x8 Mesh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Router Ports</td>
<td>5</td>
</tr>
<tr>
<td>VCs/port</td>
<td>4 (req), 4 (fwd) [1-flit/VC], 4 (resp) [5-flit/VC]</td>
</tr>
<tr>
<td>Flit Size</td>
<td>128-bit</td>
</tr>
<tr>
<td>Link Length</td>
<td>1mm</td>
</tr>
</tbody>
</table>

**Baseline Collective:**
NoC with 1-cycle fork/reduce at every hop
SFO with synthetic 1-to-All Traffic

SFO_Greedy has lower latency and higher throughput than SFO_Complete

SVT has lower latency than PVT

PVT has higher throughput than SVT

SVT: Shared Virtual Tree
PVT: Private Virtual Tree
BI: Broadcast Interval, i.e., interval between reserved broadcast slots
SFI with synthetic All-to-1 Traffic

82% latency reduction

- Baseline+Collective
- Greedy (Prio=Local)
- Greedy (Prio=Bypass)
- Complete

SFI_Complete has lowest latency and highest throughput
Full-System Traffic

SFO + SFI reduces runtime by 14%, just 12% more than runtime with impractical fully-connected topology.
Outline

- Motivation
- Background: SMART NoC
- SMART FanOut for 1-to-Many flows
- SMART FanIn for Many-to-1 flows
- Evaluation

Conclusions
Conclusions

- Challenge notion that *collective communication is expensive*
  - Near single-cycle (per dimension) traversal for 1-to-Many (multicast) and Many-to-1 (reduction) flows using SFO and SFI
  - Best strategy: greedy forking + complete reduction
    - 76-82% reduction in latency

- SFO+SFI can pave way for *locality-oblivious* design
  - Scalability to snoopy/limited-directory protocols
    - Network Latency and Application Runtime close to that of a system with point-to-point network
  - Ease burden on OS/Compiler
Thank you!