TORSTEN HOEFLER

Efficient networking and programming of large-scale computing systems

with R. Gerstenberger, M. Besta, R. Belli @ SPCL
presented at HP Labs, Palo Alto, CA, USA
Networks, Limits, and Design Space

- Networks cost 25-30% of a large supercomputer

- Hard limits:
  - Router radix
  - Cable length

- Soft limits:
  - Cost
  - Performance
A BRIEF HISTORY OF NETWORK TOPOLOGIES

copper cables, small radix switches

Mesh

Butterfly

Clos/Benes

Kautz

~2005

1980’s

2000’s

2008

2014

Hypercube

Fat Trees

Flat Fly

Random

1980’s

2000’s

2008

~2005

2007

???

fiber, high-radix switches

Dragonfly

Slim Fly

Torus

Trees

Fat Trees

Trees

???
A BRIEF HISTORY OF NETWORK TOPOLOGIES

copper cables, small radix switches
fiber, high-radix switches

Bandwidth \( = 2^{d\sqrt{N^{d-1}}} \)
Latency \( = \frac{d}{2}d\sqrt{N} \)
Radix \( = 2d \)
**A BRIEF HISTORY OF NETWORK TOPOLOGIES**

- **1980’s**: Copper cables, small radix switches
- **~2005**: Fiber, high-radix switches
- **2007**: Copper cables, small radix switches
- **2008**: Flat Fly
- **2008**: Fat Trees
- **2014**: Random

---

- **Mesh**: 1980’s
- **Torus**: 2007
- **Butterfly**: ~2005
- **Kautz**: 2008
- **Clos/Benes**: 2008
- **Hypercube**: 2014
- **Flat Fly**: 2008
- **Fat Trees**: 2008
- **Random**: ????

---

Bandwidth: \( \frac{N}{2} \)  
Latency: \( \log_2 N \)  
Radix: \( \log_2 N \)
A BRIEF HISTORY OF NETWORK TOPOLOGIES

- Mesh
- Butterfly
- Clos/Benes
- Kautz
- Hypercube
- Torus
- Flat Fly
- Fat Trees
- Random
- Blew Fly
- Slim Fly

1980’s
2000’s
~2005
2007
2008
2014

- Bandwidth = 1
- Latency = 2 \log_2 N
- Radix = 2
- Copper cables, small radix switches
- Fiber, high-radix switches
A BRIEF HISTORY OF NETWORK TOPOLOGIES

- **Mesh**
- **Torus**
- **Butterfly**
- **Clos/Benes**
- **Kautz**
- **Dragonfly**
- **Slim Fly**
- **Hypercube**
- **Fat Trees**

Bandwidth = $\frac{N}{2}$
Latency = $2 \log_2 N$
Radix = 4

- **1980’s:** Copper cables, small radix switches
- **2000’s:** Fiber, high-radix switches
- **~2005:**
- **2007:**
- **2008:**
- **2014:**

---

spcl.inf.ethz.ch
@spcl_eth
A BRIEF HISTORY OF NETWORK TOPOLOGIES

copper cables, small radix switches

Mesh

1980’s

Butterfly

Clos/Benes

2000’s

Kautz

~2005

Hypercube

Slim Fly

2007

2008

2008

2014

Fat Trees

Bandwidth

Latency

Radix

⇒ \frac{N}{4}

= \log_k N

= k
A BRIEF HISTORY OF NETWORK TOPOLOGIES

copper cables, small radix switches

fiber, high-radix switches

Bandwidth $\approx \frac{N}{4}$
Latency $= 3 - 5$
Radix $= 48 - 64$
A BRIEF HISTORY OF NETWORK TOPOLOGIES

- **Copper cables, small radix switches**
- **Fiber, high-radix switches**

### Topologies:

- **Mesh**
- **Torus**
- **Butterfly**
- **Clos/Benes**
- **Kautz**
- **Dragonfly**
- **Slim Fly**
- **Hypercube**
- **Trees**
- **Fat Trees**
- **Flat Fly**
- **Random**

### Timeline:

- **1980’s**
- **2000’s**
- **~2005**
- **2007**
- **2008**
- **2014**

### Formulas:

- **Bandwidth** \( \approx \frac{N}{4} \)
- **Latency** \( = 2 - 4 \)
- **Radix** \( = k \)
DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS

- Intuition: lower average distance $\rightarrow$ lower resource needs
  - A new view as primary optimization target!
- Moore Bound [1]: upper bound on the number of routers in a graph with given diameter $(D)$ and network radix $(k)$.

$$MB(D, k) = 1 + k + k(k-1) + k(k-1)^2 + \cdots$$

$$MB(D, k) = 1 + k \sum_{i=0}^{D-1} (k-1)^i$$

DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS: DIAMETER 2

- Example Slim Fly design for diameter = 2: MMS graphs [1] (utilizing graph covering)

DESIGNING AN EFFICIENT NETWORK TOPOLOGY
CONNECTING ROUTERS: DIAMETER 2

Groups form a fully-connected bipartite graph
DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS: DIAMETER 2

1. Select a prime power \( q \)
\[
q = 4w + \delta;
\]
where \( w \in \mathbb{N} \) and \( \delta \in \{-1,0,1\} \).

- A Slim Fly based on \( q \):
  - Number of routers: \( 2q^2 \)
  - Network radix: \( (3q - \delta)/2 \)

2. Construct a finite field \( \mathbb{F}_q \).

- Assuming \( q \) is prime:
  - \( \mathbb{F}_q = \mathbb{Z}/q\mathbb{Z} = \{0,1,\ldots,q-1\} \) with modular arithmetic.

Example: \( q = 5 \)

- 50 routers
- Network radix: 7
- \( \mathbb{F}_5 = \{0,1,2,3,4\} \)
DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS: DIAMETER 2

3 Label the routers

Set of routers:

\[ \{0,1\} \times F_q \times F_q \]

Example: \( q = 5 \)

...
DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS: DIAMETER 2

4. Find primitive element $\xi$

$\xi \in \mathcal{F}_q$ generates $\mathcal{F}_q$:
All non-zero elements of $\mathcal{F}_q$
can be written as $\xi^i$; $i \in \mathbb{N}$

5. Build Generator Sets

$X = \{1, \xi^2, \ldots, \xi^{q-3}\}$

$X' = \{\xi, \xi^3, \ldots, \xi^{q-2}\}$

Example: $q = 5$

$\mathcal{F}_5 = \{0,1,2,3,4\}$

$\xi = 2$

$1 = \xi^4 \mod 5 = 2^4 \mod 5 = 16 \mod 5$

$X = \{1,4\}$

$X' = \{2,3\}$
**DESIGNING AN EFFICIENT NETWORK TOPOLOGY**

**CONNECTING Routers: DIAMETER 2**

---

**6 Intra-group connections**

Two routers in one group are connected iff their “vertical Manhattan distance” is an element from:

\[
X = \{1, \xi^2, ..., \xi^{q-3}\} \text{ (for subgraph 0)}
\]

\[
X' = \{\xi, \xi^3, ..., \xi^{q-2}\} \text{ (for subgraph 1)}
\]

---

**Example: q = 5**

Take Routers \((0,0,.)\)

\[
X = \{1, 4\}
\]
**Designing an Efficient Network Topology**

**Connecting routers: Diameter 2**

6. *Intra-group connections*

Two routers in one group are connected iff their "vertical Manhattan distance" is an element from:

- \( X = \{1, \xi^2, ..., \xi^{q-3}\} \) (for subgraph 0)
- \( X' = \{\xi, \xi^3, ..., \xi^{q-2}\} \) (for subgraph 1)

**Example: \( q = 5 \)**

Take Routers (1,4,.)

\( X' = \{2,3\} \)
**DESIGNING AN EFFICIENT NETWORK TOPOLOGY**

**CONNECTING ROUTERS: DIAMETER 2**

7 *Inter-group connections*

Router \((0, x, y) \leftrightarrow (1, m, c)\)

iff \(y = mx + c\)

---

Example: \(q = 5\)

\(m = 0, c = 0\)

Take Router \((1,0,0)\)

\((1,0,0) \leftrightarrow (0, x, 0)\)

Take Router \((1,1,0)\)

\((1,1,0) \leftrightarrow (0, x, x)\)
DESIGNING AN EFFICIENT NETWORK TOPOLOGY

ATTACHING ENDPOINTS: DIAMETER 2

- How many endpoints do we attach to each router?
- As many to ensure *full global bandwidth*:
  - Global bandwidth: the theoretical cumulative throughput if all endpoints simultaneously communicate with all other endpoints in a steady state

\[ \text{network radix} = 67\% \text{ of router radix} \]
\[ \text{concentration} = 33\% \text{ of router radix} \]
COMPARISON TO OPTIMALITY

- How close is the presented Slim Fly network to the Moore Bound?
Overview of our research

- Topology design
- Attaching endpoints
- Optimizing towards Moore Bound
- Comparison of optimality

Cost, power, resilience analysis

- Physical layout
- Cost model
- Comparison targets
- Cost & power results
- Detailed case-study

Routing and performance

- Performance & routing
- Routing
- Resilience
- Latency, bandwidth
Mix (pairwise) groups with different cabling patterns to shorten inter-group cables
PHYSICAL LAYOUT
PHYSICAL LAYOUT
Merge groups pairwise to create racks
PHYSICAL LAYOUT
Racks form a fully-connected graph
**Physical Layout**

**SlimFly:**
- ~50% fewer intra-group cables
- 2(q-1) inter-group cables between two groups
- ~25% fewer routers

**Dragonfly:**
- ~33% higher endpoint density
- One inter-group cable between two groups

SlimFly: Dragonfly:
COST COMPARISON

RESULTS

Assuming COTS material costs and best known layout for each topology!
A Slim Fly with
- \( N = 10,830 \)
- \( k = 43 \)
- \( N_r = 722 \)
## COST & POWER COMPARISON

**DETAILED CASE-STUDY: high-radix topologies**

<table>
<thead>
<tr>
<th>Topology</th>
<th>Fat tree</th>
<th>Random</th>
<th>Flat. Butterfly</th>
<th>Dragonfly</th>
<th>Slim Fly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoints ($N$)</td>
<td>19,876</td>
<td>40,200</td>
<td>20,736</td>
<td>58,806</td>
<td>10,830</td>
</tr>
<tr>
<td>Routers ($N_r$)</td>
<td>2,311</td>
<td>4,020</td>
<td>1,728</td>
<td>5,346</td>
<td>722</td>
</tr>
<tr>
<td>Radix ($k$)</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>Electric cables</td>
<td>19,414</td>
<td>32,488</td>
<td>9,504</td>
<td>56,133</td>
<td>6,669</td>
</tr>
<tr>
<td>Fiber cables</td>
<td>40,215</td>
<td>33,842</td>
<td>20,736</td>
<td>29,524</td>
<td>6,869</td>
</tr>
<tr>
<td><strong>Cost per node [$$]</strong></td>
<td>2,346</td>
<td>1,743</td>
<td>1,570</td>
<td>1,438</td>
<td>1,033</td>
</tr>
<tr>
<td><strong>Power per node [W]</strong></td>
<td>14.0</td>
<td>12.04</td>
<td>10.8</td>
<td>10.9</td>
<td>8.02</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Topology</th>
<th>Fat tree</th>
<th>Random</th>
<th>Flat. Butterfly</th>
<th>Dragonfly</th>
<th>Slim Fly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoints ($N$)</td>
<td>10,718</td>
<td>9,702</td>
<td>10,000</td>
<td>9,702</td>
<td>10,830</td>
</tr>
<tr>
<td>Routers ($N_r$)</td>
<td>1,531</td>
<td>1,386</td>
<td>1,000</td>
<td>1,386</td>
<td>722</td>
</tr>
<tr>
<td>Radix ($k$)</td>
<td>35</td>
<td>28</td>
<td>33</td>
<td>27</td>
<td>43</td>
</tr>
<tr>
<td>Electric cables</td>
<td>7,350</td>
<td>6,837</td>
<td>4,500</td>
<td>9,009</td>
<td>6,669</td>
</tr>
<tr>
<td>Fiber cables</td>
<td>24,806</td>
<td>7,716</td>
<td>10,000</td>
<td>4,900</td>
<td>6,869</td>
</tr>
<tr>
<td><strong>Cost per node [$$]</strong></td>
<td>2,315</td>
<td>1,566</td>
<td>1,535</td>
<td>1,342</td>
<td>1,033</td>
</tr>
<tr>
<td><strong>Power per node [W]</strong></td>
<td>14.0</td>
<td>11.2</td>
<td>10.8</td>
<td>10.8</td>
<td>8.02</td>
</tr>
</tbody>
</table>
OVERVIEW OF OUR RESEARCH

- Topology design
- Attaching endpoints
- Optimizing towards Moore Bound
- Comparison of optimality

Cost, power, resilience analysis

- Physical layout
- Cost model
- Comparison targets

Cost & power results
- Detailed case-study
- Resilience

Routing and performance

- Routing
- Performance, latency, bandwidth
PERFORMANCE & ROUTING

- Cycle-accurate simulations [1]
- Routing protocols:
  - Minimum static routing
  - Valiant routing [2]
  - Universal Globally-Adaptive Load-Balancing routing [3]
    - **UGAL-L**: each router has access to its local output queues
    - **UGAL-G**: each router has access to the sizes of all router queues in the network

---

PERFORMANCE & ROUTING

RANDOM UNIFORM TRAFFIC

GRAPH 500
Intermediate conclusions

- We have:
  - The cheapest full-bandwidth topology (25% less than DF)
    Basing on group theory, large number of options (more than DF)
  - Requires advanced routing techniques (adaptive)
    Works somewhat with next-gen IB, we work on Ethernet solutions

- Is that all?
  - No – the endpoint is actually most (more?) important for performance!
  - So let’s see ….
COMMUNICATION IN TODAY’S HPC SYSTEMS

- The de-facto programming model: MPI-1
  - Using send/recv messages and collectives

- The de-facto network standard: RDMA, SHM
  - Zero-copy, user-level, os-bypass, fuzz-bang
MPI-1 MESSAGE PASSING – SIMPLE EAGER

MPI-1 MESSAGE PASSING – SIMPLE EAGER

Producer

Send

1. Data transfer to intermediate buffer

Consumer

Mailbox

MPI-1 MESSAGE PASSING – SIMPLE EAGER

Critical path: 1 latency + 1 copy

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

Producer

Consumer

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

1. Transfer of communication parameters

Producer

Send

Consumer

Mailbox

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

1. Transfer of communication parameters
2. Message matching

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

1. Transfer of communication parameters
2. Message matching
3. Request
4. Data transfer

Send

Producer

Consumer

Recv

*: target aware of completion

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

A Critique of RDMA
by Patrick Geoffray, Ph.D.

Do you remember VIA, the Virtual Interface Architecture? I do. In 1998, according to its promoters — Intel, Compaq, and Microsoft — VIA was supposed to change the face of high-performance networking. VIA was a buzzword at the time; Venture Capital was flowing, and startups multiplying. Many HPC pundits were rallying behind this low-level programming interface, which promised scalable, low-overhead, high-throughput communication, initially for HPC and eventually for the data center. The hype was on and doom was spelled for the non-believers.

It turned out that VIA, based on RDMA (Remote Direct Memory Access, or Remote DMA), was not an improvement on existing APIs to support widely used application-software interfaces such as MPI and Sockets. After a while, VIA faded away, overtaken by other developments.

VIA was eventually reborn into the RDMA programming model that is the basis of various InfiniBand Verbs implementations, as well as DAPL (Direct Access Provider Library) and iWARP (Internet Wide Area RDMA Protocol). The pundits have returned, VCs are spending their money, and RDMA is touted as an ideal solution for the efficiency of high-performance networks.

However, the evidence I'll present here shows that the revamped RDMA model is more a problem than a solution. What's more, the objective that RDMA pretends to address of efficient user-level communication between computing nodes is already solved by the two-sided Send/Recv model in products such as Quadrics QsNet, Cray SeaStar (implementing Sandia Portals), Qlogic InfiniPath, and Myricom's Myrinet Express (MX).

Send/Recv versus RDMA

The difference between these two paradigms, Send/Receive (Send/Recv) and RDMA, resides essentially in the...
REMOTE MEMORY ACCESS PROGRAMMING

- Why not use these RDMA features more directly?
  - A global address space may simplify programming
  - … and accelerate communication
  - … and there could be a widely accepted standard

- MPI-3 RMA ("MPI One Sided") was born ('13)
  - Just one among many others (UPC, CAF, …)
  - Designed to react to hardware trends, learn from others
  - Direct (hardware-supported) remote access
  - New way of thinking for programmers

MPI-3 RMA Summary

- MPI-3 updates RMA ("MPI One Sided")
  - Significant change from MPI-2
- Communication is „one sided” (no involvement of destination)
  - Utilize direct memory access
- RMA decouples communication & synchronization
  - Fundamentally different from message passing

MPI-3 RMA Communication Overview

- Process A (passive)
  - Memory
  - MPI window

- Process B (active)
  - Memory
  - MPI window
  - Non-atomic communication calls (put, get)

- Process C (active)
  - Atomic
  - Get

- Process D (active)
  - Atomic communication calls (Acc, Get & Acc, CAS, FAO)

Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI
MPI-3 RMA Communication Overview

- Process A (passive)
  - Memory
  - MPI window

- Process B (active)
  - Memory
  - MPI window
  - Non-atomic communication calls (put, get)

- Process C (active)
  - Atomic communication calls (Acc, Get & Acc, CAS, FAO)

- Process D (active)
  - Atomic
  - Get

Gropp, Hoelfer, Thakur, Lusk: Using Advanced MPI
MPI-3 RMA COMMUNICATION OVERVIEW

Process A (passive)

Process B (active)

Process C (active)

Process D (active)

Memory

MPI window

Non-atomic communication calls (put, get)

Atomic

Atomic communication calls (Acc, Get & Acc, CAS, FAO)

Put

Get
MPI-3 RMA COMMUNICATION OVERVIEW

Process A (passive)

Memory

Put

Non-atomic communication calls (put, get)

Process B (active)

Memory

MPI window

Process C (active)

Get

Process D (active)

Atomic communication calls (Acc, Get & Acc, CAS, FAO)

Gropp, Hoelfer, Thakur, Lusk: Using Advanced MPI
MPI-3 RMA Communication Overview

Process A (passive)

Memory

MPI window

Put

Non-atomic communication calls (put, get)

Atomic

Get

Process B (active)

Memory

MPI window

Atomic communication calls (Acc, Get & Acc, CAS, FAO)

Process C (active)

Process D (active)
MPI-3 RMA SYNCHRONIZATION OVERVIEW

Active Target Mode

- Fence
- Post/Start/Complete/Wait

Passive Target Mode

- Lock
- Lock All

Active process
Passive process
Synchronization
Communication

Active process
Passive process
Synchronization
Communication

Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI
MPI-3 RMA SYNCHRONIZATION OVERVIEW

Active Target Mode

- Fence
- Post/Start/Complete/Wait

Passive Target Mode

- Lock
- Lock All

Active process
Passive process
Synchronization
Communication

Gropp, Hoelfer, Thakur, Lusk: Using Advanced MPI
MPI-3 RMA SYNCHRONIZATION OVERVIEW

**Active Target Mode**
- Fence
- Post/Start/Complete/Wait

**Passive Target Mode**
- Lock
- Lock All

- Active process
- Passive process
- Synchronization
- Communication

Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI
MPI-3 RMA SYNCHRONIZATION OVERVIEW

Active Target Mode
- Fence
- Post/Start/Complete/Wait
- Active process
- Passive process
- Synchronization
- Communication

Passive Target Mode
- Lock
- Lock All

Gropp, Hoelfer, Thakur, Lusk: Using Advanced MPI
MPI-3 RMA SYNCHRONIZATION OVERVIEW

Active Target Mode

Passive Target Mode

Fence

Post/Start/Complete/Wait

Lock

Lock All

Active process
Passive process
Synchronization
Communication

Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI
SCALABLE PROTOCOLS & REFERENCE IMPLEMENTATION

- Scalable & generic protocols
  - Can be used on any RDMA network (e.g., OFED/IB)
  - Window creation, communication and synchronization

Window creation

Communication

Synchronization

Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13
Scalable & generic protocols
- Can be used on any RDMA network (e.g., OFED/IB)
- Window creation, communication and synchronization

foMPI, a fully functional MPI-3 RMA implementation
- DMAPP: lowest-level networking API for Cray Gemini/Aries systems
- XPMEM, a portable Linux kernel module

http://spcl.inf.ethz.ch/Research/Parallel_Programming/foMPI
SCALABLE PROTOCOLS & REFERENCE IMPLEMENTATION

- Scalable & generic protocols
  - Can be used on any RDMA network (e.g., OFED/IB)
  - Window creation, communication and synchronization

- foMPI, a fully functional MPI-3 RMA implementation
  - DMAPP: lowest-level networking API for Cray Gemini/Aries systems
  - XPMEM, a portable Linux kernel module

http://spcl.inf.ethz.ch/Research/Parallel_Programming/foMPI
SCALABLE PROTOCOLS & REFERENCE IMPLEMENTATION

- Scalable & generic protocols
  - Can be used on any RDMA network (e.g., OFED/IB)
  - Window creation, communication and synchronization

- foMPI, a fully functional MPI-3 RMA implementation
  - DMAPP: lowest-level networking API for Cray Gemini/Aries systems
  - XPMEM: a portable Linux kernel module

http://spcl.inf.ethz.ch/Research/Parallel_Programming/foMPI
**Performance Inter-node: Latency**

### Put Inter-Node

- 80% faster

### Get Inter-Node

- 20% faster

Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13
**Performance Intra-node: Latency**

Put/Get Intra-Node

![Graph showing latency vs size for different transport layers](image)

- 3x faster
- Half ping-pong
- Proc 0
- Proc 1
- Put
- Sync memory

Transport Layer:
- FOMPI MPI-3.0
- Cray UPC
- Cray MPI-2.2
- Cray MPI-1
- Cray CAF

Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13
**Performance: Message Rate**

Inter-Node

![Inter-Node Diagram](image)

Intra-Node

![Intra-Node Diagram](image)

---

Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13
PART 3: SYNCHRONIZATION

Active Target Mode

- Fence
- Post/Start/Complete/Wait

Passive Target Mode

- Lock
- Lock All

Active process
Passive process
Synchronization
Communication
SCALABLE FENCE PERFORMANCE

Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13
Flush Synchronization

- Guarantees remote completion
- Performs a remote bulk synchronization and an x86 mfence
- One of the most performance critical functions, we add only 78 x86 CPU instructions to the critical path

<table>
<thead>
<tr>
<th>Time bound</th>
<th>( O(1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory bound</td>
<td>( O(1) )</td>
</tr>
</tbody>
</table>

Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13
Flush Synchronization

- Guarantees remote completion
- Performs a remote bulk synchronization and an x86 mfence
- One of the most performance critical functions, we add only 78 x86 CPU instructions to the critical path

<table>
<thead>
<tr>
<th>Time bound</th>
<th>$\mathcal{O}(1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory bound</td>
<td>$\mathcal{O}(1)$</td>
</tr>
</tbody>
</table>

Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13
## Performance Modeling

### Performance functions for synchronization protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fence</td>
<td>$P_{\text{fence}} = 2.9\mu s \cdot \log_2(p)$</td>
</tr>
</tbody>
</table>
| PSCW     | $P_{\text{start}} = 0.7\mu s, P_{\text{wait}} = 1.8\mu s$  
          | $P_{\text{post}} = P_{\text{complete}} = 350\text{ns} \cdot k$ |
| Locks    | $P_{\text{lock, excl}} = 5.4\mu s$  
          | $P_{\text{lock, shrd}} = P_{\text{lock, all}} = 2.7\mu s$  
          | $P_{\text{unlock}} = P_{\text{unlock, all}} = 0.4\mu s$  
          | $P_{\text{flush}} = 76\text{ns}$  
          | $P_{\text{sync}} = 17\text{ns}$ |

### Performance functions for communication protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Formula</th>
</tr>
</thead>
</table>
| Put/get  | $P_{\text{put}} = 0.16\text{ns} \cdot s + 1\mu s$  
          | $P_{\text{get}} = 0.17\text{ns} \cdot s + 1.9\mu s$ |
| Atomics  | $P_{\text{acc, sum}} = 28\text{ns} \cdot s + 2.4\mu s$  
          | $P_{\text{acc, min}} = 0.8\text{ns} \cdot s + 7.3\mu s$ |
APPLICATION PERFORMANCE

- Evaluation on Blue Waters System
  - 22,640 computing Cray XE6 nodes
  - 724,480 schedulable cores
- All microbenchmarks
- 4 applications
- One nearly full-scale run 😊
PERFORMANCE: MOTIF APPLICATIONS

Key/Value Store: Random Inserts per Second

<table>
<thead>
<tr>
<th>Transport Layer</th>
<th>FOMPI MPI-3.0</th>
<th>Cray UPC</th>
<th>Cray MPI-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>intra-node</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inter-node</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dynamic Sparse Data Exchange (DSDE) with 6 neighbors

<table>
<thead>
<tr>
<th>Transport Layer</th>
<th>FOMPI MPI-3.0</th>
<th>LibNBC</th>
<th>Cray MPI-2.2</th>
<th>Cray Reduce_scatter</th>
<th>Cray Alltoall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time [us]</td>
<td>8</td>
<td>32</td>
<td>128</td>
<td>512</td>
<td>2048</td>
</tr>
</tbody>
</table>

Number of Processes
PERFORMANCE: APPLICATIONS


NAS 3D FFT [1] Performance

MILC [2] Application Execution Time

[1] Nishtala et al.: Scaling communication-intensive applications on BlueGene/P using one-sided communication and overlap. IPDPS’09
[2] Shan et al.: Accelerating applications at scale using one-sided communication. PGAS’12
IN CASE YOU WANT TO LEARN MORE

- Available in most MPI libraries today
- Some are even fast!

Using Advanced MPI

Modern Features of the Message-Passing Interface

How to implement producer/consumer in passive mode?

William Gropp
Torsten Hoefler
Rajeev Thakur
Ewing Lusk
PRODUCER-CONSUMER RELATIONS

- Most important communication idiom
  - Some examples:

- Perfectly supported by MPI-1 Message Passing
  - But how does this actually work over RDMA?
ONE SIDED – PUT + SYNCHRONIZATION

Producer

Consumer
ONE SIDED – PUT + SYNCHRONIZATION

Producer

Put

Consumer

1. Data transfer

ONE SIDED – PUT + SYNCHRONIZATION

Producer

Put

Flush

Consumer

1. Data transfer

2. Producer waits for remote completion

: origin aware of completion
ONE SIDED – PUT + SYNCHRONIZATION

Critical path: 3 latencies

COMPARING APPROACHES

- **Message Passing**
  - 1 latency + copy / 3 latencies

- **One Sided**
  - 3 latencies
### IDEA: RMA Notifications

- First seen in Split-C (1992)
- Combine communication and synchronization using RDMA
- RDMA networks can provide various notifications
  - Flags
  - Counters
  - Event Queues

---

**Put + Notification**
- 1. Data transfer + notification
- 2. Acknowledgement

**Get + Notification**
- 1. Data transfer + notification

![Diagram showing Put and Get operations with notifications and statuses](image)

- : target aware of completion
- : origin aware of completion

COMPARING APPROACHES

Message Passing
1 latency + copy / 3 latencies

One Sided
3 latencies

Notified Access
1 latency

Comparing Approaches

Message Passing
1 latency + copy / 3 latencies

One Sided
3 latencies

Notified Access
1 latency

But how to notify?
PREVIOUS WORK: OVERWRITING INTERFACE

- Flags (polling at the remote side)
  - Used in GASPI, DMAPP, NEON

- Disadvantages
  - Location of the flag chosen at the sender side
  - Consumer needs at least one flag for every process
  - Polling a high number of flags is inefficient
**PREVIOUS WORK: COUNTING INTERFACE**

- Atomic counters (accumulate notifications → scalable)
  - Used in *Split-C, LAPI, SHMEM - Counting Puts, …*

- Disadvantages
  - Dataflow applications may require many counters
  - High polling overhead to identify accesses
  - Does not preserve order (may not be linearizable)
WHAT IS A GOOD NOTIFICATION INTERFACE?

- Scalable to yotta-scale
  - Does memory or polling overhead grow with # of processes?

- Computation/communication overlap
  - Do we support maximum asynchrony? (better than MPI-1)

- Complex data flow graphs
  - Can we distinguish between different accesses locally?
  - Can we avoid starvation?
  - What about load balancing?

- Ease-of-use
  - Does it use standard mechanisms?
**OUR APPROACH: NOTIFIED ACCESS**

- Notifications with MPI-1 (queue-based) matching
  - Retains benefits of previous notification schemes
  - Poll only head of queue
  - Provides linearizable semantics

---

## Notified Access – An MPI Interface

- Minor interface evolution
  - Leverages MPI two sided `<source, tag>` matching
  - Wildcards matching with FIFO semantics

### Example Communication Primitives

```c
int MPI_Put  (void *origin_addr, int origin_count, MPI_Datatype origin_type, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_type, MPI_Win win);
int MPI_Get   (void *origin_addr, int origin_count, MPI_Datatype origin_type, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_type, MPI_Win win);
```

### Example Synchronization Primitives

```c
/*Functions already available in MPI*/
int MPI_Start(MPI_Request *request);
int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status);
int MPI_Wait(MPI_Request *request, MPI_Status *status);
```
NOTIFIED ACCESS – AN MPI INTERFACE

- Minor interface evolution
  - Leverages MPI two sided <source, tag> matching
  - Wildcards matching with FIFO semantics

Example Communication Primitives

```c
int MPI_Put_notify(void *origin_addr, int origin_count, MPI_Datatype origin_type, int target_rank,
                   MPI_Aint target_disp, int target_count, MPI_Datatype target_type, MPI_Win win,
                   int tag);

int MPI_Get_notify(void *origin_addr, int origin_count, MPI_Datatype origin_type, int target_rank,
                    MPI_Aint target_disp, int target_count, MPI_Datatype target_type, MPI_Win win,
                    int tag);
```

Example Synchronization Primitives

```c
int MPI_Notify_init(MPI_Win win, int src_rank, int tag, int expected_count, MPI_Request *request);
/*Functions already available in MPI*/
int MPI_Start(MPI_Request *request);
int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status);
int MPI_Wait(MPI_Request *request, MPI_Status *status);
```
NOTIFIED ACCESS - IMPLEMENTATION

- **foMPI** – a fully functional MPI-3 RMA implementation
  - Runs on newer Cray machines (Aries, Gemini)
  - **DMAPP**: low-level networking API for Cray systems
  - **XPMEM**: a portable Linux kernel module
- **Implementation of Notified Access via uGNI** [1]
  - Leverages uGNI queue semantics
  - Adds unexpected queue
  - Uses 32-bit immediate value to encode source and tag

**EXPERIMENTAL SETTING**

- **Piz Daint**
  - Cray XC30, Aries interconnect
  - 5'272 computing nodes (Intel Xeon E5-2670 + NVIDIA Tesla K20X)
  - Theoretical Peak Performance 7.787 Petaflops
  - Peak Network Bisection Bandwidth 33 TB/s

PING PONG PERFORMANCE (INTER-NODE)

- 1000 repetitions, each timed separately, RDTSC timer
- 95% confidence interval always within 1% of median
PING PONG PERFORMANCE (INTRA-NODE)

- 1000 repetitions, each timed separately, RDTSC timer
- 95% confidence interval always within 1% of median
**COMPUTATION/COMMUNICATION OVERLAP**

- 1000 repetitions, each timed separately, RDTSC timer
- 95% confidence interval always within 1% of median

---

**Graph:**

- **Notified Access**
- **MPI Message Passing**
- **MPI One Sided**

- **Protocol Switch**
  - FMA $\rightarrow$ BTE

- **Protocol Switch**
  - Eager $\rightarrow$ Rendezvous

- **Uses communication progression thread**

**Legend:**

- Comp/Comm Overlap %
- Number of Transferred Bytes

(lower is better)
**Pipeline – One-to-One Synchronization**

- 1000 repetitions, each timed separately, RDTSC timer
- 95% confidence interval always within 1% of median

[Image of a graph showing normalized completion time for different synchronization methods across various numbers of processes. The graph indicates that notified access is consistently lower, indicating a lower completion time.]

REDUCE – ONE-TO-MANY SYNCHRONIZATION

- Reduce as an example (same for FMM, BH, etc.)
  - Small data (8 Bytes), 16-ary tree
  - 1000 repetitions, each timed separately with RDTSC

![Diagram showing synchronization patterns and completion times](attachment:diagram.png)

- Notified Access
- MPI Message Passing
- MPI One Sided PSCW
- MPI Reduce

(lower is better)
**CHOLESKY – MANY-TO-MANY SYNCHRONIZATION**

- 1000 repetitions, each timed separately, RDTSC timer
- 95% confidence interval always within 10% of median

![Graph showing performance of Notified Access, MPI Message Passing, and MPI One Sided across different numbers of processes.](image)

(Higher is better)

- Notified Access
- MPI Message Passing
- MPI One Sided

Number of Processes: 4, 8, 16, 32, 64, 128

GMOPS: 0.0, 0.2, 0.4, 0.6, 0.8, 1.0

[1]: J. Kurzak, H. Ltaief, J. Dongarra, R. Badia: "Scheduling dense linear algebra operations on multicore processors", CCPE 2010
DISCUSSION AND CONCLUSIONS

- We develop a close-to-optimal network topology
  - Spawns new research on adaptive routing
- RDMA+SHM are de-facto hardware mechanisms
  - Gives rise to RMA programming
- MPI-3 RMA standardizes clear semantics
  - Builds on existing practice (UPC, CAF, ARMCI etc.)
  - Rich set of synchronization mechanisms
- Notified Access can support producer/consumer
  - Maintains benefits of RDMA
- Fully parameterized LogGP-like performance model
  - Aids algorithm development and reasoning

<table>
<thead>
<tr>
<th>Shared Memory</th>
<th>uGNI FMA</th>
<th>uGNI BTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>0.25(\mu s)</td>
<td>1.02(\mu s)</td>
</tr>
<tr>
<td>G</td>
<td>0.08ns</td>
<td>0.105ns</td>
</tr>
</tbody>
</table>

**Function**

- MPI\_Notify\_init
- MPI\_Request\_free
- MPI\_Start
- MPI\_{Put\|Get}\_notify

**Time**

- \(t_{init} = 0.07\mu s\)
- \(t_{free} = 0.04\mu s\)
- \(t_{start} = 0.008\mu s\)
- \(t_{na} = 0.29\mu s\)
COST COMPARISON

COST MODEL

- Electric cables, avg length: 1m
- Top-of-rack routers

*Most cables skipped for clarity

2m of overhead for each global link

Optic cables, length: Manhattan distance

Racks arranged as close to a square as possible

1 meter
COST COMPARISON

CABLE COST MODEL

- Cable cost as a function of distance
  - The functions obtained using linear regression*
  - Cables used:
    Mellanox IB FDR10 40Gb/s QSFP

- Other used cables:
  Mellanox IB QDR 56Gb/s QSFP
  Mellanox Ethernet 40Gb/s QSFP
  Mellanox Ethernet 10Gb/s SFP+
  Elpeus Ethernet 10Gb/s SFP+

*Prices based on:

\[
\begin{align*}
  f(x) &= 0.4079x + 0.5771 \\
  f(x) &= 0.0919x + 2.7452
\end{align*}
\]

Cables used: Mellanox IB FDR10 QSFP 40 Gb/s
COST COMPARISON

ROUTER COST MODEL

- Router cost as a function of radix
  - The function obtained using linear regression*
  - Routers used:
    - Mellanox IB FDR10
    - Mellanox Ethernet 10/40 Gb

\[ f(k) = 350.4k - 892.3 \]

*Prices based on:
## Structure Analysis

**Resiliency**

- Disconnection metrics*
- Other studied metrics:
  - Average path length (increase by 2);
  - SF is 10% more resilient than DF

*Missing values indicate the inadequacy of a balanced topology variant for a given N

<table>
<thead>
<tr>
<th>≈ N</th>
<th>Torus3D</th>
<th>Torus5D</th>
<th>Hypercube</th>
<th>Long Hop</th>
<th>Fat tree</th>
<th>Dragonfly</th>
<th>Flat. Butterfly</th>
<th>Random</th>
<th>Slim Fly</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>30%</td>
<td>-</td>
<td>40%</td>
<td>55%</td>
<td>35%</td>
<td>-</td>
<td>55%</td>
<td>60%</td>
<td>60%</td>
</tr>
<tr>
<td>1024</td>
<td>25%</td>
<td>40%</td>
<td>40%</td>
<td>55%</td>
<td>40%</td>
<td>50%</td>
<td>60%</td>
<td>-</td>
<td>65%</td>
</tr>
<tr>
<td>2048</td>
<td>20%</td>
<td>-</td>
<td>40%</td>
<td>55%</td>
<td>40%</td>
<td>55%</td>
<td>65%</td>
<td>65%</td>
<td>65%</td>
</tr>
<tr>
<td>4096</td>
<td>15%</td>
<td>-</td>
<td>45%</td>
<td>55%</td>
<td>55%</td>
<td>60%</td>
<td>70%</td>
<td>70%</td>
<td>70%</td>
</tr>
<tr>
<td>8192</td>
<td>10%</td>
<td>35%</td>
<td>45%</td>
<td>55%</td>
<td>60%</td>
<td>65%</td>
<td>-</td>
<td>75%</td>
<td>75%</td>
</tr>
</tbody>
</table>

*Missing values indicate the inadequacy of a balanced topology variant for a given N
OTHER RESULTS

Bisection b.

Avg. distance

Oversubscription analysis

Bit reverse

Bit complement

Shuffle

Shift

Adversarial

Buffer size analysis

Other cost & power results

-- 25% less expensive than in SF due to fewer routers and thus Server

<table>
<thead>
<tr>
<th>Topology</th>
<th>Dragonfly</th>
<th>Slim Fly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoints (N)</td>
<td>10,890</td>
<td>10,830</td>
</tr>
<tr>
<td>Routers (N_r)</td>
<td>990</td>
<td>722</td>
</tr>
<tr>
<td>Radix (k)</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>Electric cables</td>
<td>6,885</td>
<td>6,669</td>
</tr>
<tr>
<td>Fiber cables</td>
<td>1,012</td>
<td>6,869</td>
</tr>
<tr>
<td>Cost per node [+]</td>
<td>1,365</td>
<td>1,033</td>
</tr>
<tr>
<td>Power per node [W]</td>
<td>10.9</td>
<td>8.02</td>
</tr>
</tbody>
</table>
**SUMMARY**

**Topology design**

Optimizing towards the Moore Bound reduces expensive network resources.

**Advantages of SlimFly**

Cost & power  
Resilience  
Performance  
Diameter  
Avg. distance  
Bandwidth

**Optimization approach**

Combining mathematical optimization and current technology trends effectively tackles challenges in networking.

**Credits**

Maciej Besta  
(PhD Student @SPCL)
A LOWEST-DIAMETER TOPOLOGY
→ Viable set of configurations
→ Resilient

A COST & POWER EFFECTIVE TOPOLOGY
→ 25% less expensive than Dragonfly,
→ 26% less power-hungry than Dragonfly

A HIGH-PERFORMANCE TOPOLOGY
→ Lowest latency
→ Full global bandwidth

http://spcl.inf.ethz.ch/Research/Scalable_Networking/SlimFly

Thank you for your attention
DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS: DIAMETER 2

- Viable set of configurations
  - 10 SF networks with the number of endpoints < 11,000 (compared to 6 balanced Dragonflies [1])
- Let’s pick network radix = 7...
  - ... We get the Hoffman-Singleton graph (attains the Moore Bound)
**DESIGNING AN EFFICIENT NETWORK TOPOLOGY**

**ATTACHING ENDPOINTS: DIAMETER 2**

1. Get load $l$ per router-router channel (average number of routes per channel)

   $$l = \frac{\text{total number of routes}}{\text{total number of channels}}$$

2. Make the network balanced, i.e.,:
   - each endpoint can inject at full capacity
   - local uplink load = number of endpoints = $l$

   concentration = 33% of router radix

   network radix = 67% of router radix
COMPARISON TARGETS
LOW-RADIX TOPOLOGIES

Torus 3D
Cray XE6
IBM BG/Q
Torus 5D

Hypercube
NASA Pleiades
Infinetics
Long Hop [1]

COMPARISON TARGETS
HIGH-RADIX TOPOLOGIES

Fat tree [1]

Flattened Butterfly [2]

Cray Cascade

Random Topologies [4,5]

PERFORMANCE & ROUTING

MINIMUM ROUTING

1. **Intra-group connections**
   - Path of length 1 or 2 between two routers

2. **Inter-group connections (different types of groups)**
   - Path of length 1 or 2 between two routers

3. **Inter-group connections (identical types of groups)**
   - Path of length 2 between two routers
DESIGNING AN EFFICIENT NETWORK TOPOLOGY

GENERAL CONSTRUCTION SCHEME

- We split the problem into two pieces

Connect routers:
select diameter
select network radix
maximize number of routers

Attach endpoints
Derive concentration that provides full global bandwidth
OPTIMIZING NETWORK TOPOLOGIES

- Optimize for (bad-case) random uniform traffic
  - Can often be generated by randomization of allocations
  - Important for permutations, transpose, graph computations …

- Discrete optimization problem
  - \( \min(\text{# of routers}) \)
  - constraints:
    - \( \text{Router radix } k \)
    - \( \text{Full “global” bandwidth (“guarantee cable capacity”) } \)
  - Implemented as SAT problem: \( \binom{N-1}{k} \) options for neighbors alone!
    - \( \text{Maximum size solved was } N=8 \)

- Intuition: lower average distance \( \rightarrow \) lower resource needs
  - A new view as primary optimization target!
# COST & POWER COMPARISON

**Detailed Case-Study: high-radix topologies**

<table>
<thead>
<tr>
<th>Topology</th>
<th>Fat tree</th>
<th>Random</th>
<th>Flat. Butterfly</th>
<th>Dragonfly</th>
<th>Slim Fly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoints ($N$)</td>
<td>10,718</td>
<td>9,702</td>
<td>10,000</td>
<td>9,702</td>
<td>10,830</td>
</tr>
<tr>
<td>Routers ($N_r$)</td>
<td>1,531</td>
<td>1,386</td>
<td>1,000</td>
<td>1,386</td>
<td>722</td>
</tr>
<tr>
<td>Radix ($k$)</td>
<td>35</td>
<td>28</td>
<td>33</td>
<td>27</td>
<td>43</td>
</tr>
<tr>
<td>Electric cables</td>
<td>7,350</td>
<td>6,837</td>
<td>4,500</td>
<td>9,009</td>
<td>6,669</td>
</tr>
<tr>
<td>Fiber cables</td>
<td>24,806</td>
<td>7,716</td>
<td>10,000</td>
<td>4,900</td>
<td>6,869</td>
</tr>
<tr>
<td>Cost per node [$]</td>
<td>2,315</td>
<td>1,566</td>
<td>1,535</td>
<td>1,342</td>
<td>1,033</td>
</tr>
<tr>
<td>Power per node [W]</td>
<td>14.0</td>
<td>11.2</td>
<td>10.8</td>
<td>10.8</td>
<td>8.02</td>
</tr>
</tbody>
</table>