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
1980's  2000's  ~2005

fiber, high-radix switches

Dragonfly  Slim Fly
2007  2008  2014

Torus  Hypercube  Fat Trees  Flat Fly  Random

Trees
A BRIEF HISTORY OF NETWORK TOPOLOGIES

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

1980's
2000's
~2005
2008
2014

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

Mesh
Torus
Butterfly
Kautz
Hypercube
Fat Trees
Flat Fly
Random

1980's
~2005
2007
2008
2014

Bandwidth
Latency
Radix

\[ \frac{N}{2} \]

\[ \log_2 N \]
A BRIEF HISTORY OF NETWORK TOPOLOGIES

copper cables, small radix switches

1980’s

Mesh

2000’s

Butterfly

Clos/Benes

~2005

Kautz

2007

Hypercube

Fat Trees

Bandwidth = 1

Latency = 2 \log_2 N

Radix \equiv 2

2008

Flat Fly

2014

Random

???

fiber, high-radix switches
A BRIEF HISTORY OF NETWORK TOPOLOGIES

copper cables, small radix switches

Mesh

Butterfly

Clos/Benes

Kautz

Bandwidth \( = \frac{N}{2} \)

Latency \( = 2 \log_2 N \)

Radix \( = 4 \)

1980’s

1980’s

2000’s

~2005

2008

2014

Slim Fly

Dragonfly

Fat Trees

Torus

Hypercube

1980's

2000's

~2005

2008

2014
A BRIEF HISTORY OF NETWORK TOPOLOGIES

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

1980's  |  ~2005

Mesh  |  Butterfly  |  Kautz  |  Clos/Benes  |  Hypercube  |  Torus

1990's  |  ~2005  |  ~2008

Fat Trees  |  Flat Fly  |  Random

Bandwidth  = \rightarrow \frac{N}{4}
Latency  = \log_k N
Radix  = 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

1980’s

Mesh

Torus

1980’s

2000’s

~2005

Butterfly

Clos/Benel

Kautz

1980’s

2007

2008

2014

Hypercube

Trees

Fat Trees Flat Fly Random

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 $(k)$.

\[
MB(D, k) = 1 + k + k(k - 1) + k(k - 1)^2 + \ldots + k(k - 1)^{D-1}
\]

DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS: DIAMETER 2

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

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

**CONNECTING ROUTERS: DIAMETER 2**

1. **Select a prime power** \( q \)
   \[
   q = 4w + \delta; \\
   w \in \mathbb{N} \quad \delta \in \{-1,0,1\},
   \]
   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:
\[
\{0,1\} \times \mathbb{F}_q \times \mathbb{F}_q
\]

Example: \( q = 5 \)

...
DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS: DIAMETER 2

4. Find primitive element $\xi$

$\xi \in \mathbb{F}_q$ generates $\mathbb{F}_q$:
All non-zero elements of $\mathbb{F}_q$ 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: $q = 5$

$\mathbb{F}_5 = \{0, 1, 2, 3, 4\}$
$\xi = 2$
$1 = \xi^4 \mod 5 = 2^4 \mod 5 = 16 \mod 5$
$X = \{1, 4\}$
$X' = \{2, 3\}$
**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:

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

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

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

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

\[ X = \{1, 4\} \]

---

Graphical representation of the network topology showing the connections between routers based on the given conditions.
Intra-group connections

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

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

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: \(q = 5\)

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

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

\(m = 0, c = 0\)
\(m = 1, c = 0\)
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

\[
\begin{align*}
\text{network radix} &= 67\% \text{ of router radix} \\
\text{concentration} &= 33\% \text{ of router radix}
\end{align*}
\]
COMPARISON TO OPTIMALITY

- How close is the presented Slim Fly network to the Moore Bound?

![Graph showing network performance comparison]

Each number shows the fraction of the upper bound of a data point (higher is better).

Networks with diameter = 2:
- 88%
- 21%
- 1.6%
- 1.2%

- Moore Bound 2
- Slim Fly MMS
- Flat. Butterfly
- Long Hop
- Fat tree
OVERVIEW OF OUR RESEARCH

Topology design
- Optimizing towards Moore Bound

Attaching endpoints
- Comparison of optimality

Cost, power, resilience analysis
- Physical layout
- Cost model
- Comparison targets

Routing and performance
- Cost & power results
- Detailed case-study
- Resilience

Routing

Performance, latency, bandwidth

Optimizing towards Moore Bound
Mix (pairwise) groups with different cabling patterns to shorten inter-group cables.
PHYSICAL LAYOUT
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
- ~25% fewer routers
- 2(q-1) inter-group cables between two groups

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

RESULTS

Assuming COTS material costs and best known layout for each topology!
COST & POWER COMPARISON
DETAILED CASE-STUDY

- 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>Cost per node ([$])</td>
<td>2,346</td>
<td>1,743</td>
<td>1,570</td>
<td>1,438</td>
<td>1,033</td>
</tr>
<tr>
<td>Power per node ([\text{W}])</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>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 ([\text{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>
OVERVIEW OF OUR RESEARCH

Topology design
- Optimizing towards Moore Bound

Attaching endpoints
- 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

Routing protocol
- Slim Fly (Valiant)
- Slim Fly (Minimum)
- Slim Fly (UGAL-G)
- Slim Fly (UGAL-L)
- Dragonfly (UGAL-L)
- Fat Tree (ANCA)
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

Producer

Consumer

1. Data transfer to intermediate buffer
MPI-1 MESSAGE PASSING – SIMPLE EAGER

1. Data transfer to intermediate buffer
2. Acknowledgement

: origin aware of completion

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  

Consumer  

Send

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

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

*: target aware of completion

Critical path: 3 latencies
COMMUNICATION IN TODAY’S HPC SYSTEMS

August 18, 2006

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
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

Process B (active)

Memory

Non-atomic communication calls (put, get)

Put

Atomic

Get

Process C (active)

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

Put

Non-atomic communication calls (put, get)

Atomic

Memory

Process B (active)

MPI window

Get

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

Process C (active)

Process D (active)
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 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

Put

Non-atomic communication calls (put, get)

Atomic

Process C (active)

Get

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

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

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

Active Target Mode
- Fence
  - Active process
  - Passive process

Post/Start/Complete/Wait

Passive Target Mode
- Lock
  - Synchronization
- Communication

- Lock All
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

Passive Target Mode
- Lock
- Lock All
- 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, Hoelfer, Thakur, Lusk: Using Advanced MPI
MPI-3 RMA SYNCHRONIZATION OVERVIEW

Active Target Mode

Passive Target Mode

Gropp, Hoelfer, Thakur, Lusk: Using Advanced MPI
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 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
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

Half ping-pong
Proc 0
Proc 1

put
sync memory

Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13
PERFORMANCE INTRA-NODE: LATENCY

Put/Get Intra-Node

3x faster

Latency [μs]

Size [Bytes]

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

Half ping-pong
Proc 0
put
sync memory
Proc 1

Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13
**PERFORMANCE: MESSAGE RATE**

**Inter-Node**

![Graph showing message rate for Inter-Node communication]

**Intra-Node**

![Graph showing message rate for Intra-Node communication]

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

- Active process
- Passive process
- Synchronization
- Communication
- Lock
- Lock All
SCALABLE FENCE PERFORMANCE

<table>
<thead>
<tr>
<th></th>
<th>( \mathcal{O}(\log p) )</th>
<th>( \mathcal{O}(1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time bound</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory bound</td>
<td></td>
<td></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
**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>Fence</th>
<th>( P_{fence} = 2.9\mu s \cdot \log_2(p) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSCW</td>
<td>( 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 )</td>
</tr>
<tr>
<td>Locks</td>
<td>( 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} )</td>
</tr>
</tbody>
</table>

Performance functions for communication protocols

<table>
<thead>
<tr>
<th>Put/get</th>
<th>( 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 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomics</td>
<td>( 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 )</td>
</tr>
</tbody>
</table>
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

Dynamic Sparse Data Exchange (DSDE)
with 6 neighbors
PERFORMANCE: APPLICATIONS


NAS 3D FFT [1] Performance

MILC [2] Application Execution Time

scale to 65k procs

scale to 512k procs

[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

1. Data transfer

Consumer
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

Comparing Approaches

Message Passing
1 latency + copy / 3 latencies

One Sided
3 latencies

Notified Access
1 latency

COMPARING APPROACHES

But how to notify?

Message Passing
1 latency + copy / 3 latencies

One Sided
3 latencies

Notified Access
1 latency
**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

---

**Pipeline — One-to-One Synchronization**

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

---


(lower is better)
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

![Graph showing performance of Notified Access, MPI Message Passing, MPI One Sided PSCW, and MPI Reduce across different numbers of processes. Lower values are better.](image)
**CHOLESKY – MANY-TO-MANY SYNCHRONIZATION**

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

(Higher is better)

- Notified Access
- MPI Message Passing
- MPI One Sided

[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>0.25µs</td>
<td>1.02µs</td>
</tr>
<tr>
<td>G</td>
<td>0.08ns</td>
<td>0.105ns</td>
</tr>
</tbody>
</table>

Function               | Time       
-----------------------|------------
MPI.Notify_init        | $t_{init} = 0.07µs$  
MPI.Request_free      | $t_{free} = 0.04µs$ 
MPI.Start             | $t_{start} = 0.008µs$ 
MPI.Put|Get|notify         | $t_{na} = 0.29µs$
ACKNOWLEDGMENTS
A BRIEF HISTORY OF NETWORK TOPOLOGIES

- Copper cables, small radio switches
- Fiber-optic switches

Topologies:
- Butterfly
- Mesh
- Ring
- Fat Trees
- Torus
- 2D Mesh
- 3D Torus

COST & POWER COMPARISON

<table>
<thead>
<tr>
<th>Topology</th>
<th>Fatree</th>
<th>Random</th>
<th>Flat Mercola</th>
<th>CogentFly</th>
<th>SimFly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoints (N)</td>
<td>18,786</td>
<td>40,200</td>
<td>20,736</td>
<td>58,800</td>
<td>10,830</td>
</tr>
<tr>
<td>Nodes (N)</td>
<td>2,311</td>
<td>4,070</td>
<td>1,728</td>
<td>5,546</td>
<td>773</td>
</tr>
<tr>
<td>Rads (R)</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>Nodes (N)</td>
<td>19,714</td>
<td>22,843</td>
<td>10,830</td>
<td>773</td>
<td></td>
</tr>
<tr>
<td>Fiber cables</td>
<td>46,215</td>
<td>33,842</td>
<td>20,736</td>
<td>29,324</td>
<td>6,869</td>
</tr>
<tr>
<td>Cost per node (€)</td>
<td>2,346</td>
<td>1,743</td>
<td>1,570</td>
<td>1,438</td>
<td>1,833</td>
</tr>
<tr>
<td>Power per node (W)</td>
<td>1,48</td>
<td>12,04</td>
<td>10,8</td>
<td>10,9</td>
<td>8,89</td>
</tr>
</tbody>
</table>

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS


Critical path: 3 latencies

PERFORMANCE MODELING

Performance functions for synchronization protocols
- Fence
  - $p_{sync} = 2.9 \times 10^{-3} \times log(p)
- PRSCW
  - $p_{ready} = 0.7 \times p_{wait} = 1.8 \times p_{wait} = 1 \times p_{wait} = 2 \times p_{wait} = 3 \times p_{wait}
- Locks
  - $p_{lock} = 5.6 \times p_{unlock} = 2.7 \times p_{unlock} = 6.4 \times p_{unlock} = 7 \times p_{unlock}
  - $p_{lock} = 17.8 \times p_{unlock}$

Performance functions for communication protocols
- Put get
  - $p_{put} = 0.15 \times p_{recv} + 1 \times p_{send}$
  - $p_{recv} = 0.17 \times p_{recv} + 1.9 \times p_{recv}$
- Atomics
  - $p_{recv} = 2.8 \times p_{recv} + 2.4 \times p_{recv} = 6.9 \times p_{recv}$

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

CHOLESKY – MANY-TO-MANY SYNCHRONIZATION

1,000 repetitions, each timed separately, RDTSC timer
95% confidence interval always within 10% of median delays
- Using Advanced MPI
  - Modern Features of the Message-Passing Interface

Using Advanced MPI

Modern Features of the Message-Passing Interface

- William Gropp
- taco@mpi-mpi
- Uwe Schäfer
- Ewing Lusk
**COST COMPARISON**

**COST MODEL**

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

- 2m of overhead for each global link

- Optic cables, length: Manhattan distance

- Racks arranged as close to a square as possible

*Most cables skipped for clarity*
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:

\[ f(x) = 0.4079x + 0.5771 \]
\[ f(x) = 0.0919x + 2.7452 \]
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

*Prices based on:
## Structure Analysis

### Resiliency

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

<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>-</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

<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 = \( l \)

network radix = 67\% of router radix

concentration = 33\% 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

Flattened Butterfly [2]

Random Topologies [4,5]

Fat tree [1]

TSUBAME2.0

Dragonfly [3]

Cray Cascade

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

- Intuition: lower average distance → 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>