Towards Remote Memory Access Programming for Data Analytics

With R. Gerstenberger, M. Besta, R. Belli @ SPCL
presented at Lawrence Berkeley National Laboratory, Berkeley, CA, June 2015
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

MPI-1 MESSAGE PASSING – SIMPLE EAGER

MPI-1 MESSAGE PASSING – SIMPLE EAGER

**MPI-1 MESSAGE PASSING – SIMPLE EAGER**

1. Data transfer to intermediate buffer
2. Acknowledgement
3. Message matching and copy

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

**Critical path: 1 latency + 1 copy**

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

Producer

1. Transfer of communication parameters

Consumer

Mailbox

Send

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

1. Transfer of communication parameters

2. Message matching


MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

1. Transfer of communication parameters
2. Message matching
3. Request

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

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
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") [1] was born
  - 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

"Traditionally, HPC programming models are following hardware developments” (IPDPS’15)

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

Put

Non-atomic communication calls (put, get)

MPI window

Atomic

Get

Process B (active)

Memory

Process C (active)

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

Process D (active)

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

Get

Process C (active)

Process D (active)

Atomic communication calls (Acc, Get & Acc, CAS, FAO)
MPI-3 RMA COMMUNICATION OVERVIEW

Process A (passive)

Process B (active)

Process C (active)

Process D (active)

Memory

MPI window

Put

Get

Non-atomic communication calls (put, get)

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

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

- **Process A (passive)**
  - Memory

- **Process B (active)**
  - Memory
  - MPI window

- **Process C (active)**
  - MPI window

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

Non-atomic communication calls (put, get)

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

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

Process A (passive) — Memory

Process B (active)

Process C (active)

Process D (active)

Memory

 MPI window

Put

Non-atomic communication calls (put, get)

Get

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

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, Hoefler, Thakur, Lusk: Using Advanced MPI, MIT Press, 2014
**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, MIT Press, 2014
MPI-3 RMA SYNCHRONIZATION OVERVIEW

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

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

**Communication**
- Synchronization

Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI, MIT Press, 2014
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, MIT Press, 2014
MPI-3 RMA SYNCHRONIZATION OVERVIEW

Active Target Mode

- Fence
- Post/Start/Complete/Wait

Passive Target Mode

- Lock
- Synchronization
- Communication

Active process
Passive process

Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI, MIT Press, 2014
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**
- **Transport Layer**
  - FOMPI MPI-3.0
  - Cray UPC
  - Cray MPI-2.2
  - Cray MPI-1
  - Cray CAF

- **Size [Bytes]**
  - 8
  - 64
  - 512
  - 4096
  - 32768
  - 262144

- **Latency [μs]**
  - 0.1
  - 1.0
  - 10.0
  - 100.0

- **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: MESSAGE RATE

Inter-Node

Intra-Node

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

Time bound: $O(\log p)$
Memory bound: $O(1)$

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></th>
<th>Time bound</th>
<th>Memory bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>counter: 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inc(counter)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inc(counter)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inc(counter)</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

<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
APPLICATION PERFORMANCE

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


### NAS 3D FFT [1] Performance

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

![Graph showing performance of NAS 3D FFT with different transport layers](image)

**Applications**

- Scale to 65k procs

### MILC [2] Application Execution Time

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

![Graph showing application execution time for MILC](image)

**Applications**

- 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
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

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

**Producer**
- Put
- Flush
- Explicit Synch

**Consumer**
- Explicit Synch

1. Data transfer
2. Producer waits for remote completion
3. Producer reports completion to consumer

★ : target aware of completion
★ : origin aware of completion

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**

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

**One Sided**
- 3 latencies

**Notified Access**
- 1 latency

But **how to notify?**

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

- **Message Passing**
  - 1. Transfer of communication parameters
  - 2. Message matching
  - 3. Request
  - 4. Data transfer
  - 5. Acknowledgement

- **Consumer**
  - Send
  - Recv

- **Producer**
  - Mailbox

- **RMA Put + Synchronization**
  - 1. Data transfer
  - 2. Acknowledgement
  - 3. Producer reports completion

- **Explicit**
  - Put
  - Flush

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

- **Consumer**
  - Wait Notification

- **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_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);
```
EXPERIMENTAL SETTING

- Piz Daint [1]
  - 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

**CHOLESKY – MANY-TO-MANY SYNCHRONIZATION**

- 1,000 repetitions, each timed separately, RDTSC timer
- 95% confidence interval always within 10% of median

(Higher is better)

So what if we drive tasking to the extreme of data analytics?

[1]: J. Kurzak, H. Ltaief, J. Dongarra, R. Badia: "Scheduling dense linear algebra operations on multicore processors“, CCPE 2010
LARGE-SCALE IRREGULAR GRAPH PROCESSING

- Becoming more important [1]
  - Machine learning
  - Computational science
  - Social network analysis

SYNCHRONIZATION MECHANISMS
COARSE LOCKS

Simple protocols

Serialization

Detrimental performance

An example graph

Proc p

lock

accesses

unlock

Proc q

lock

accesses
Synchronization Mechanisms
Fine Locks

- Higher performance possible
- Complex protocols
- Risk of deadlocks

Complex access patterns 😊
SYNCHRONIZATION MECHANISMS

ATOMIC OPERATIONS

High performance (may be challenging to get)

Complex protocols

Subtle issues (ABA, ...)

Complex access patterns 😊
SYNCHRONIZATION MECHANISMS
SOFTWARE TRANSACTIONAL MEMORY (STM) [1]

Conflicts solved with rollbacks and/or serialization.

Software overheads
Simple protocols

SYNCHRONIZATION MECHANISMS
HARDWARE TRANSACTIONAL MEMORY (HTM)

Conflicts solved with rollbacks and/or HW serialization.

High performance? For graphs?

Simple protocols

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
They offer programmability, how about performance?
ACTIVE MESSAGES (AM)


AM + HTM = **Atomic Active Messages**

AM handlers run as HTM transactions

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
ACCESSING MULTIPLE VERTICES ATOMICALLY
Example: BFS

Transaction by thread A

Transaction by thread B

Size (the number of vertices) must be appropriate to minimize overheads from both commits and rollbacks

Besta, Hoefner: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
transactions must be appropriately coalesced to minimize communication overheads.
Vertices must be appropriately relocated to enable execution of a hardware transaction.

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
**Performance Analysis Research Questions**

- How can we implement AAM handlers to run most efficiently?
- What are the performance tradeoffs related to HTM?
- What are advantages of HTM over atomics for AAM?
- What are the optimal transaction sizes?

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
**Performance Analysis**

**Types of Machines**

- Evaluation on 3 machines
  - Intel Haswell server
  - InfiniBand cluster
  - IBM BlueGene/Q
Performance Analysis Considered Mechanisms

Haswell HTM
- 32KB per core
- Deployed in L1
- 8-way associative
- RTM (Restricted Transactional Memory)
- HLE (Hardware Lock Elision)

BlueGene/Q HTM
- 2MB per core
- Deployed in L2
- 16-way associative

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
**Single-Vertex Transactions**

**Marking a Vertex as Visited**

- **Very few aborts**
  - Lower contention (10 racing accesses/vertex)

```java
// start handler
if(!v.visited) {
  v.visited = 1;
}
// finish handler
```

- **Atomics (CAS) slightly faster than HTM**
- **Commit overheads dominate**

Used in BFS, SSSP, ...

- **Figure**: Graph showing the total time in ms against the number of threads per node (T). The graph compares Intel RTM / HLE with Intel atomics. The graph indicates that Intel RTM / HLE has lower commit overheads compared to Intel atomics, which is faster but has higher commit overheads.
**SINGLE-VERTEX TRANSACTIONS**
MARKING A VERTEX AS VISITED

Higher contention
(100 racing accesses/vertex)

Still very few aborts

BG/Q HTM still worse (L1 vs L2 matters!)

RTM better than atomics

// start handler
if(!v.visited) {
  v.visited = 1;
}
// finish handler

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC'15

Used in BFS, SSSP, ...
**Single-Vertex Transactions**

**Incrementing Vertex Rank**

- Used in PageRank

```c
// start handler
v.rank++;
// finish handler
```

- Atomics always outperform HTM

- The reason: each transaction always modifies some memory cell, increasing the number of conflicts

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
**Performance Model**

**Atomics vs Transactions**

Time to modify $N$ vertices with atomics:

$$T_{AT}(N) = A_{AT}N + B_{AT}$$

Time to modify $N$ vertices with a transaction:

$$T_{HTM}(N) = A_{HTM}N + B_{HTM}$$

We predict that:

- $B_{AT} < B_{HTM}$
- $A_{AT} > A_{HTM}$

Transactions’ cost grows slower

Transaction startup overheads dominate

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
**Performance Model**

Atomic vs Transactions

- Can we amortize HTM startup/commit overheads with larger transaction sizes?

Indeed:

\[
B_{AT} < B_{HTM} \quad \text{and} \quad A_{AT} > A_{HTM}
\]

Yes, we can!
**MULTI-VERTEX TRANSACTIONS**

**MARKING VERTICES AS VISITED**

Startup and commit overheads

Abort and rollback overheads

The sweetspot! (144 vertices)

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
**MULTI-VERTEX TRANSACTIONS**

**MARKING VERTICES AS VISITED**

Numbers: % of aborts due to HTM capacity overflows

Abort and rollback overheads

Majority of aborts are due to HTM capacity overflows (small cache size & associativity)

The sweetspot! (2 vertices)

Startup and commit overheads

---

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
**PERFORMANCE ANALYSIS QUESTIONS ANSWERED**

- "It really depends" 😊. But... There are some regularities. How can we implement AAM handlers most effectively?
- "May fail". For some algorithms (BFS) HTM is better. For others (PageRank) atomics give more performance.
- "Always succeed". AAM establishes a whole hierarchy of algorithms; check the paper 😊.
- Larger cache & associativity $\rightarrow$ fewer aborts & more coarsening. Larger (L2) cache $\rightarrow$ higher latency.
- Same for other graphs. Size for BG/Q $\sim$100 $>$ Size for Haswell $\sim$10.

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
Evaluation Considered Engines

- **PBGL [4]**: Distributed HPC libraries
  - Runtimes that exploit amorphous data-parallelism

- **AAM +**
  - Improving Graph500 design

- **HAMA [3]**: Hadoop-based BSP engines

---

**Evaluation**

**Considered Types of Graphs**

**Synthetic graphs**
- Kronecker [1]
- Erdös-Rényi [2]

**Real-world SNAP graphs** [3]
- Social networks
- Road networks
- Comm. graphs
- Citation graphs
- Web graphs
- Purchase networks

---

ACCELERATING STATE-OF-THE-ART
GRAPH500 + AAM (BLUEGENE/Q) IBM

Fill the whole memory

Graphs with 32M vertices:
- 0.94
- 1.19
- 1.28
- 1.32

Graphs with 8M vertices:
- 1.31
- 1.39
- 1.55
- 1.38

Graphs with 2M vertices:
- 1.76
- 2.02
- 1.56
- 1.38

Numbers are speedups of AAM over Graph500 for a given data point

Implementation:
- Graph500-BGQ
- AAM-BGQ

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
ACCELERATING STATE-OF-THE-ART
GRAPH500 + AAM (HASWELL)

Numbers are speedups of AAM over Graph500 for a given data point

Implementation
- Graph500–Haswell
- AAM–Haswell

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC'15
OUTPERFORMING STATE-OF-THE-ART

<table>
<thead>
<tr>
<th></th>
<th>Input graph properties</th>
<th>BG/Q analysis</th>
<th>Haswell analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ID</td>
<td>Name</td>
<td>V</td>
</tr>
<tr>
<td>Copper networks</td>
<td>cWT</td>
<td>wiki-Talk</td>
<td>2.1M</td>
</tr>
<tr>
<td></td>
<td>eEU</td>
<td>email-EnAll</td>
<td>2.05k</td>
</tr>
<tr>
<td></td>
<td>sLV</td>
<td>soc-Livecl</td>
<td>4.8M</td>
</tr>
<tr>
<td></td>
<td>sO</td>
<td>soc-Orkut</td>
<td>1.1M</td>
</tr>
<tr>
<td></td>
<td>tCA</td>
<td>roadNet-CA</td>
<td>5.5M</td>
</tr>
<tr>
<td></td>
<td>tTX</td>
<td>roadNet-TX</td>
<td>3.8M</td>
</tr>
<tr>
<td></td>
<td>tPA</td>
<td>roadNet-PA</td>
<td>3.3M</td>
</tr>
<tr>
<td></td>
<td>eCP</td>
<td>cit-Patents</td>
<td>3.7M</td>
</tr>
<tr>
<td></td>
<td>wGO</td>
<td>web-Google</td>
<td>8.75k</td>
</tr>
<tr>
<td></td>
<td>wDS</td>
<td>web-DarkStarn</td>
<td>605k</td>
</tr>
<tr>
<td></td>
<td>wSF</td>
<td>web-Stanford</td>
<td>281k</td>
</tr>
</tbody>
</table>

😊 No, you don’t have to read it. All details are in the paper. Here: just a summary.
OUTPERFORMING STATE-OF-THE-ART

Average overall speedup (geometric mean) over Graph 500: 1.07, Galois: 1.40, HAMA ~1000

1.85x on average, up to 4.3x

Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
OUTPERFORMING STATE-OF-THE-ART
SCALABILITY ANALYSIS: DISTRIBUTED-MEMORY

PBGL, 1 process/node
PBGL, 4 processes/node
AAM, 1 thread/node
AAM, 4 threads/node
PBGL does not support threading, thus we run more than 1 process/node

The whole node memory filled
PBGL, 128 nodes
AAM, 128 nodes
PBGL, 16 nodes
AAM, 16 nodes

Nodes (N)
Threads/node (T)
OTHER ANALYSES
DISCUSSION AND CONCLUSIONS

- MPI-3 RMA [1] standardizes weak remote memory
  - Builds on existing practice (UPC, CAF, ARMCI etc.)
  - Rich set of synchronization mechanisms
- Notified Access [2] can support producer/consumer
  - Maintains benefits of RDMA
- HTM can accelerate parallel applications [3]
  - Uses optimistic coarsened irregular parallelism
- Atomic Active Messages use HTM on DM systems
  - First steps towards software-emulated TM over RDMA
  - Thinking about hardware support
- Significant speedups over highly-tuned graph frameworks
  - Haswell: 7% over Graph 500, 40% over Galois, 1000x over Hama
- What next? Discussions?
  - GraphBlas using RMA+HTM?

[3] Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC'15
ACKNOWLEDGMENTS
**MPI-1 Message Passing – Simple Rendezvous**

- Producer
  - Send
  - Mailbox
  - 1. Transfer of communication parameters
  - 2. Message matching
  - 3. Request
  - 4. Data transfer
  - 5. Acknowledgment
  - Target aware of completion
  - Origin aware of completion
- Consumer
  - Receive

**Performance Inter-node: Latency**

- Put Inter-node
  - 89% faster
- Get Inter-node
  - 20% faster

**Performance: Applications**

- Annotations represent performance gain of HAFiP over Cray MP-1 [3].

**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**

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

**Synchronization Mechanisms**

- Hardware Transactional Memory (HTM)
  - Conflicts solved with rollbacks and/or HIN sanitization
- Multi-Vertex Transactions
  - Marking vertices as visited

**Multi-Vertex Transactions**

- Abort and rollback overheads
- The sweet spot (144 vertices)

**Accelerating State-of-the-Art**

- Graph500 + AAM (BlueGene/Q)
  - Numbers are speedups of AAM over Graph500 for a given data point.