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

MPI-1 MESSAGE PASSING – SIMPLE EAGER

Producer

Send

1. Data transfer to intermediate buffer

Consumer

Mailbox

MPI-1 MESSAGE PASSING – SIMPLE EAGER

Producer

Send

1. Data transfer to intermediate buffer

2. Acknowledgement

Consumer

Mailbox

Diamond: 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

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

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

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

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

*: target aware of completion
*: origin aware of completion

Critical path: 3 latencies

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

http://www.hpcwire.com/2006/08/18/a_critique_of_rdma-1/
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)

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

Process B (active)

Memory

MPI window

Get

Process C (active)

Process D (active)

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

Process A (passive)

Memory

Put

Process B (active)

MPI window

Non-atomic communication calls (put, get)

Get

Process C (active)

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

Process D (active)

Atomic

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

Process A (passive)

Memory

Put

Non-atomic communication calls (put, get)

Process B (active)

Memory

MPI window

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)

Process B (active)

Memory

 MPI window

Get

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

Process C (active)

Process D (active)

Atomic

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

Non-atomic communication calls (put, get)

Put

Atomic

Get

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
- 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

Passive Target Mode

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

Active process
Passive process
Synchronization
Communication

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**

<table>
<thead>
<tr>
<th>Size [Bytes]</th>
<th>8</th>
<th>64</th>
<th>512</th>
<th>4096</th>
<th>32768</th>
<th>262144</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency [µs]</td>
<td>1.6</td>
<td>1.2</td>
<td>0.8</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
</tr>
</tbody>
</table>

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

3x faster

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

Intra-Node

Inter-Node

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

DMAPP protocol change

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
- Active process
- Passive process
- Synchronization
- Communication

**Passive Target Mode**
- Lock
- Lock All
SCALABLE FENCE PERFORMANCE

~½ latency

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Time bound</td>
<td>$O(\log p)$</td>
</tr>
<tr>
<td>Memory bound</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
**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>$O(1)$</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
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

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**

![Diagram showing the process of one-sided put and synchronization]

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

- Star: target aware of completion
- Diamond: origin aware of completion

**Critical path: 3 latencies**

COMPARING APPROACHES

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

**One Sided**
3 latencies

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

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

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?
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
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

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

Node A
Proc p

Node B
Proc q

start transaction

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

Size (the number of vertices) must be appropriate to minimize overheads from both commits and rollbacks
Transactions must be appropriately coalesced to minimize communication overheads.
EXECUTING TRANSACTIONS ON MULTIPLE NODES

Vertices must be appropriately relocated to enable execution of a hardware transaction.
**Performance Analysis**

**Research Questions**

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

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

Commodity machines

Supercomputing machines

HPC clusters

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

CONSIDERED MECHANISMS

Haswell HTM

- Deployed in L1
- 32KB per core
- 8-way associative

RTM (Restricted Transactional Memory)
HLE (Hardware Lock Elision)

BlueGene/Q HTM

- Deployed in L2
- 2MB per core
- 16-way associative

The Long Running Mode
The Short Running Mode
SINGLE-VERTEX TRANSACTIONS
MARKING A VERTEX AS VISITED

Lower contention
(10 racing accesses/vertex)

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

Very few aborts

Atomics (CAS) slightly faster than HTM

Commit overheads dominate

Used in BFS, SSSP, ...
SINGLE-VERTEX TRANSACTIONS
MARKING A VERTEX AS VISITED

Still very few aborts

Higher contention
(100 racing accesses/vertex)

Intel atomics
BG/Q HTM
Better than atomics

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

Atomsics always outperform HTM

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

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

Used in PageRank

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

Overhead per vertex
Startup overheads
Overhead per vertex
Startup overheads

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

ATOMICS VS TRANSACTIONS

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

Indeed:

\[ B_{AT} < B_{HTM} \]
\[ A_{AT} > A_{HTM} \]

Yes, we can!
MULTI-VERTEX TRANSACTIONS
MARKING VERTICES AS VISITED

The sweetspot! (144 vertices)

 Abort and rollback overheads

Startup and commit overheads

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

MARKING VERTICES AS VISITED

- **The sweetspot!** (2 vertices)
- **Startup and commit overheads**
- **Abort and rollback overheads**
- **Majority of aborts are due to HTM capacity overflows (small cache size & associativity)**

Numbers: % of aborts due to HTM capacity overflows

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 → fewer aborts & more coarsening.

- Larger (L2) cache → higher latency.

- "It really depends" 😊. But... There are some regularities.
  - What are the optimal transaction sizes?
  - What are the performance tradeoffs related to HTM?

- Size for BG/Q ~100
  - Size for Haswell ~10

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

CONSIDERED ENGINES


PBGL [4]
Distributed HPC libraries

[2] 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

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.

- Total time in seconds
- Edges per vertex (\(\bar{d}\))
- Graphs with 8M vertices
- Graphs with 2M vertices
- Implementation:
  - Graph500–Haswell
  - AAM–Haswell

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

😊 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 does not support threading, thus we run more than 1 process/node

PBGL, 1 process/node
PBGL, 4 processes/node
AAM, 1 thread/node
AAM, 4 threads/node

The whole node memory filled

PBGL, 128 nodes
PBGL, 16 nodes
AAM, 128 nodes
AAM, 16 nodes

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

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