TORSTEN HOEFLER

Active RDMA - new tricks for an old dog

with M. Besta, R. Belli, S. di Girolamo @ SPCL

presented at Salishan Meeting, Gleneden Beach, OR, USA, April 2016
TORSTEN HOEFLER

Active RDMA - new tricks for an old dog

with M. Besta, R. Belli, S. di Girolamo @ SPCL

presented at Salishan Meeting, Salishan, OR, USA, April 2016

Alternative (better) title: Beyond RDMA
Remote Operations
- put, get, atomics

Remote Matching
- partial control at target

Lossy Networks
- Ethernet
- 1980’s

Lossless Networks
- RDMA
- 2000’s

Full Device Programs
- Offload
- 2020’s
Remote Operations
- put, get, atomics

Remote Matching
- partial control at target

Remote Synchronization
[IPDPS’15]
- Extend RMA semantics
- Fully one-sided (in HW)
- Synchronization

Remote Transactions
[HPDC’15]
- Similar to HTM
- Extend across nodes
- Think active messages?

Remote Invocation
[ICS’15]
- Utilizes IOMMUs
- Control transfer
- Active memory
RDMA

**IN CASE YOU WANT TO LEARN MORE ABOUT RMA**

- PGAS and RMA are programming abstractions
  - PGAS as language extension (e.g., UPC, CAF)
  - RMA as library (integrated in MPI)
- Offer abstraction for
  - Data placement, read, write, some atomic operations
  - Target has very little control
- RDMA is a hardware mechanism
  - Often accessible through a library (OFED, uGNI, DMAPP, libfabric, …)
  - Specific to a (set of) hardware implementation(s)
  - Offers varying levels of functionality
    - Most common: read, write
    - Address-space management
    - Common denominator is often virtual address access

**RDMA vs. RMA vs. PGAS?**

- How to implement producer/consumer in passive mode?

**Using Advanced MPI**

*Modern Features of the Message-Passing Interface*

- 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?
Remote Synchronization
ONE SIDED – PUT + SYNCHRONIZATION

Producer

Consumer
ONE SIDED – PUT + SYNCHRONIZATION

Producer

Put

1. Data transfer

Consumer
**ONE SIDED – PUT + SYNCHRONIZATION**

![Diagram showing producer and consumer with operations and synchronization points.]

- **Put**: Producer initiates data transfer.
- **Flush**: Producer waits for remote completion.
- **1. Data transfer**: Initial data transfer from producer to consumer.
- **2. Producer waits for remote completion**: Producer waits for completion of the data transfer.

Diamond symbolizes origin aware of completion.
ONE SIDED – PUT + SYNCHRONIZATION

Critical path: 3 latencies

Produced

Consumer

Put

1. Data transfer

Flush

2. Producer waits for remote completion

Explicit Synch

3. Producer reports completion to consumer

Star: target aware of completion

Diamond: origin aware of completion

IPDPS'15
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

PING PONG PERFORMANCE (INTER-NODE)

- 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

![Graph showing normalized completion time vs. number of processes](image)

- MPI Message Passing
- MPI One Sided
- Notified Access

(lower is better)

CHOLESKY – MANY-TO-MANY SYNCHRONIZATION

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

[1]: J. Kurzak, H. Ltaief, J. Dongarra, R. Badia: "Scheduling dense linear algebra operations on multicore processors", CCPE 2010
(Remote) Transactions
**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
...

M. Kulkarni et al., Optimistic Parallelism Benefits from Data Partitioning, ASPLOS’08
SYNCHRONIZATION MECHANISMS
FINE LOCKS

Higher performance possible

Complex protocols

Risk of deadlocks

J. Yan et al., Exploiting fine-grained parallelism in graph traversal algorithms via lock virtualization on multi-core architecture, Journ. of Supercomp.
SYNCHRONIZATION MECHANISMS

ATOMIC OPERATIONS

- High performance (may be challenging to get)
- Complex protocols
- Subtle issues (ABA, ...)

Complex access patterns

Proc p

Proc q

V. Agarwal et al., Scalable Graph Exploration on Multicore Processors, IEEE/ACM Supercomputing 2010 (SC10)
SYNCHRONIZATION MECHANISMS
TRANSACTIONAL MEMORY (CF. DB TRANSACTIONS)

Conflicts solved with rollbacks and/or serialization.

Software overheads

Simple protocols

N. Shavit and D. Touitou. Software transactional memory. PODC’95.
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
PERFORMANCE MODEL
ATOMICS VS TRANSACTIONS

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

![Graph showing performance model for Haswell and BlueGene/Q]

Besta, Hoeffer: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
MULTI-VERTEX TRANSACTIONS IN A BFS (GRAPH 500) MARKING VERTICES AS VISITED

Startup and commit overheads

Abort and rollback overheads

The sweetspot! (144 vertices)

Besta, Hoefer: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC’15
**REAL-GRAph PERFORMANCE**

<table>
<thead>
<tr>
<th>input graph properties</th>
<th>BiC/Q analysis</th>
<th>Hensell analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>Social networks (10K)</td>
<td>1</td>
<td>1.45</td>
</tr>
<tr>
<td>News networks (10K)</td>
<td>2</td>
<td>1.45</td>
</tr>
<tr>
<td>ORAAM networks (10K)</td>
<td>3</td>
<td>1.45</td>
</tr>
<tr>
<td>ORAAM networks (100K)</td>
<td>4</td>
<td>1.45</td>
</tr>
<tr>
<td>ORAAM graphs (10K)</td>
<td>5</td>
<td>1.45</td>
</tr>
<tr>
<td>ORAAM graphs (100K)</td>
<td>6</td>
<td>1.45</td>
</tr>
</tbody>
</table>

😊 No, you don’t have to read it. 😊 Here: just a summary.
REAL-GRAH PERFORMANCE

Average overall speedup (geomean) over Graph 500: 1.07,
Galois [1]: 1.40, HAMA: ~1000

1.85x on average, up to 4.3x

[1]: Satish et al.: Navigating the Maze of Graph Analytics Frameworks Using Massive Graph Datasets, SIGMOD’14
Remote Invocation
IMAGINE A SIMPLE DISTRIBUTED HASH-TABLE

No collision:
- 1 remote atomic
- Up to 5x speedup over MP [1]

A collision:
- 4 remote atomics + 2 remote puts
- Significant performance drops

**USE INPUT/OUTPUT MEMORY MANAGEMENT UNITS**

- **Main memory**
  - Physical addresses
  - IOMMU
  - IOTLB
  - I/O devices
  - Device addresses
  - Physical addresses
  - MMU
  - TLB
  - Virtual addresses
  - CPU
  - Physical addresses

M. Besta and T. Hoefler, Active Access: A Mechanism for High-Performance Distributed Data-Centric Computations, ICS’15
ACTIVE PUTS

Process p

1. Put(X)

Process q

2. Attempt to write(X)

Accessed page

3. Page fault! (W = 0)

Access log

4. Move(X)

Main memory

5. Process(X)

CPU

IOMMU

Do not modify the page

Log both the entry and the data of an incoming put

M. Besta and T. Hoefler, Active Access: A Mechanism for High-Performance Distributed Data-Centric Computations, ICS’15
ACTIVE GETS

Enable reading from the page

Log both the entry and the data accessed by a get

M. Besta and T. Hoefler, Active Access: A Mechanism for High-Performance Distributed Data-Centric Computations, ICS’15
INTERACTIONS WITH THE CPU

- Interrupts
- Polling
- Direct notifications via scratchpads

M. Besta and T. Hoefler, Active Access: A Mechanism for High-Performance Distributed Data-Centric Computations, ICS’15
PERFORMANCE: LARGE-SCALE CODES
DISTRIBUTED HASHTABLE

Collisions: 5%

Collisions: 25%

M. Besta and T. Hoefler, Active Access: A Mechanism for High-Performance Distributed Data-Centric Computations, ICS’15
Towards a Network Instruction Set Architecture (NISA)
An example for offloading
Communications
(non-blocking)

Computations

Dependencies

Offload Engine

recv

comp

send

L0: recv a from P1;
L1: b = compute f(buff, a);
L2: send b to P1;
L0 and CPU --> L1
L1 --> L2

S. di Girolamo, P. Jolivet, K. D. Underwood, T. Hoefler: Exploiting Offload Enabled Network Interfaces, HOTI'
Fully Offloaded Collectives

Collective communication: A communication that involves a group of processes

Non-blocking collective: Once initiated the operation may progress independently of any computation or other communication at participating processes
Fully Offloaded Collectives

Collective communication: A communication that involves a group of processes
Non-blocking collective: Once initiated the operation may progress independently of any computation or other communication at participating processes

Fully Offloading:
1. *No synchronization* is required in order to start the collective operation
2. Once a collective operation is started, *no further CPU intervention* is required in order to progress or complete it.

S. di Girolamo, P. Jolivet, K. D. Underwood, T. Hoefler: Exploiting Offload Enabled Network Interfaces, HOTI'
A Case Study: Portals 4

- Based on the one-sided communication model
- Matching/Non-Matching semantics can be adopted
A Case Study: Portals 4

**Communication primitives**
- Put/Get operations are natively supported by Portals 4
- One-sided + matching semantic

**Atomic operations**
- Operands are the data specified by the MD at the initiator and by the ME at the target
- Available operators: min, max, sum, prod, swap, and, or, …

**Counters**
- Associated with MDs or MEs
- Count specific events (e.g., operation completion)

**Triggered operations**
- Put/Get/Atomic associated with a counter
- Executed when the associated counter reaches the specified threshold
**FFlib: An Example**

Proof of concept library implemented on top of Portals 4

```c
ff_schedule_h sched = ff_schedule_create(...);

ff_op_h r1 = ff_op_create_recv(tmp + blocksize, blocksize, child1, tag);
ff_op_h r2 = ff_op_create_recv(tmp + 2*blocksize, blocksize, child2, tag);

ff_op_h c1 = ff_op_create_computation(rbuff, blocksize, tmp + blocksize, blocksize, operator, datatype, tag)
ff_op_h c2 = ff_op_create_computation(rbuff, blocksize, tmp + 2*blocksize, blocksize, operator, datatype, tag)

ff_op_h s = ff_op_create_send(rbuff, blocksize, parent, tag)

ff_op_hb(r1, c1)
ff_op_hb(r2, c2)
ff_op_hb(c1, s)
ff_op_hb(c2, s)

ff_schedule_add(sched, r1)
ff_schedule_add(sched, r2)
ff_schedule_add(sched, c1)
ff_schedule_add(sched, c2)
ff_schedule_add(sched, s)
```

S. di Girolamo, P. Jolivet, K. D. Underwood, T. Hoeffer: Exploiting Offload Enabled Network Interfaces, HOTI'
Experimental Results: Latency/Overhead

Target machine: Curie
5,040 nodes
2 eight-core Intel Sandy Bridge processors
Full fat-tree InfiniBand QDR

OMPI/P4: Open MPI 1.8.4 + Portals 4 RL
FFLIB: proof of concept library

More about FFLIB at: http://spcl.inf.ethz.ch/Research/Parallel_Programming/FFlib/
Active RDMA – what could it be?

Remote Synchronization
- Remote Synchronization
  - [I/O]DPS'15: Utilizes IOMMUs
  - [I/O]DPS'15: Control transfer
  - [I/O]DPS'15: Active memory

Remote Transactions
- Remote Transactions
  - [HPDC'15]: Similar to HTM
  - [HPDC'15]: Extend across nodes?

Network Instruction Set (NISA)
- NISA: Process the data while it moves!

Remote Invocation
- [ICS'15]: Utilizes IOMMUs
- [ICS'15]: Control transfer
- [ICS'15]: Active memory