Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages

Maciej Besta, Torsten Hoefler
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
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

Proc p

start transaction

accesses

commit transaction

Proc q

start transaction

accesses

commit transaction
HARDWARE TRANSACTIONAL MEMORY

- Azul Systems
- IBM BlueGene/Q
- Haswell
- POWER8
- Rock
- Vega
- Intel
- Sun Microsystems
Hardware Transactional Memory

They offer programmability, how about performance?
**SHARED- & DISTRIBUTED-MEMORY MACHINES**

- HTM works fine for single shared-memory domains
  - Most graphs fit in such machines [1]
- However, some do not:
  - Very large instances
  - Rich vertex/edge data
- Fat nodes with lots of RAM still expensive ($35K for a machine with 1TB of RAM [1])

How to apply HTM in such a setting?

OVERVIEW OF OUR RESEARCH

AAM Design

Active Messages + HTM

Coarsening & coalescing

Evaluation

Considered engines and graphs

Performance Modeling & Analysis

Haswell & BG/Q Analysis

Performance model

Accelerating state-of-the-art

Scalability
ACTIVE MESSAGES (AM)


AM + HTM = ATOMIC ACTIVE MESSAGES

AM handlers run as HTM transactions

Node A
Proc p

Node B
Proc q

start transaction

start transaction
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
EXECUTING TRANSACTIONS ON MULTIPLE NODES

Vertices must be appropriately relocated to enable execution of a hardware transaction
OVERVIEW OF OUR RESEARCH

AAM Design

Active Messages + HTM

Coarsening & coalescing

Evaluation

Considered engines and graphs

Performance Modeling & Analysis

Haswell & BG/Q Analysis

Performance model

Accelerating state-of-the-art

Scalability
**Performance Analysis**

**Research Questions**

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

**Types of Machines**

- Evaluation on 3 machines
  - Intel Haswell server
  - InfiniBand cluster
  - IBM BlueGene/Q

---

*Commodity machines*

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

Very few aborts

Lower contention
(10 racing accesses/vertex)

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

Atomics (CAS) slightly faster than HTM

Commit overheads dominate

Used in BFS, SSSP, ...

Numbers are total aborts per data point

Total time [ms]

Threads per node (T)

BG/Q HTM (long mode)

BG/Q HTM (short mode)

BG/Q atomics

Intel RTM / HLE

Intel atomics
SINGLE-VERTEX TRANSACTIONS
MARKING A VERTEX AS VISITED

Still very few aborts

Higher contention
(100 racing accesses/vertex)

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

RTM better than atomics

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

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

Used in BFS, SSSP, ...

Intel atomics
BG/Q HTM
BG/Q HTM (long mode)
BG/Q HTM (short mode)
Intel HLE
Intel RTM
Intel atomics
BG/Q atomics
Numbers are total aborts per data point
SINGLE-VERTEX TRANSACTIONS
INCREMENTING VERTEX RANK

Atomsics always outperform HTM

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

Used in PageRank

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

We predict that:

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

Transactions' cost grows slower

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

Yes, we can!

Indeed:

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

\[ A_{AT} > A_{HTM} \]
**MULTI-VERTEX TRANSACTIONS**

**MARKING VERTICES AS VISITED**

- **Startup and commit overheads**
- **The sweetspot! (144 vertices)**
- **Abort and rollback overheads**

![Graph showing BGQ mechanism with HTM-Long-Mode and HTM-Short-Mode]

**Total time [s]**

- **Transaction size (M) [vertices]**

- **Atomic CAS**

The graph illustrates the performance of BGQ mechanism in different transaction sizes, highlighting the optimal sweet spot at 144 vertices with minimal abort and rollback overheads.
**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 (large cache size & associativity)**

- **Startup and commit overheads**
- **The sweetspot! (2 vertices)**

The diagram shows the relationship between transaction size and total time, with annotations highlighting specific percentages of aborts due to HTM capacity overflows.
PERFORMANCE ANALYSIS
QUESTIONS ANSWERED

- What are the best transaction sizes?
- How can we implement AAM handlers most effectively?
- What are advantages of HTM over atomics for AAM?
- What are performance tradeoffs related to HTM?
- What are the best transaction sizes?
“It really depends” 😊. But... There are some regularities.

Larger cache & associativity $\rightarrow$ fewer aborts & more coarsening

Larger (L2) cache $\rightarrow$ higher latency

For some algorithms (BFS) HTM is better

For others (PageRank) atomics give more performance

AAM establishes a whole hierarchy of algorithms; check the paper 😊

Same for other graphs

Size for BG/Q $\sim$100

Size for Haswell $\sim$10
Overview of our Research

AAM Design

Coarsening & coalescing

Active Messages + HTM

Performance Modeling & Analysis

Haswell & BG/Q Analysis

Evaluation

Considered engines and graphs

Accelerating state-of-the-art

Scalability
EVALUATION
CONSIDERED ENGINES

PBGL [4]
Distributed HPC libraries

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)

Fill the whole memory

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

Implementation
- Graph500–BGQ
- AAM–BGQ
ACCELERATING STATE-OF-THE-ART
GRAPH500 + AAM (HASWELL)

Fill the whole memory

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

Implementation
- Graph500-Haswell
- AAM-Haswell

Edges per vertex ($\overline{d}$)

Total time [s]
## Outperforming State-of-the-Art

<table>
<thead>
<tr>
<th>Input graph properties</th>
<th>BG/Q analysis</th>
<th>Haswell analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comm. networks (Ns)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cWT wiki-Talk</td>
<td>2.4M 5M</td>
<td>2.82 48 3.35</td>
</tr>
<tr>
<td>cEU email-EuAll</td>
<td>265k 420k</td>
<td>3.67 32 4.36</td>
</tr>
<tr>
<td>Social networks (SnS)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sLV soc-LiveJ.</td>
<td>4.8M 69M</td>
<td>1.44 12 1.56</td>
</tr>
<tr>
<td>sOR com-orkut</td>
<td>3M 117M</td>
<td>1.22 20 1.27</td>
</tr>
<tr>
<td>sLJ com-lj</td>
<td>4M 34M</td>
<td>1.44 12 1.54</td>
</tr>
<tr>
<td>sYT com-youtube</td>
<td>1.1M 2.9M</td>
<td>1.67 8 1.84</td>
</tr>
<tr>
<td>sDB com-dblp</td>
<td>317k 1M</td>
<td>1.33 8 1.80</td>
</tr>
<tr>
<td>sAM com-amazon</td>
<td>334k 925k</td>
<td>1.14 8 1.62</td>
</tr>
<tr>
<td>Purchase network (PNs)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pAM amazon0601</td>
<td>403k 3.3M</td>
<td>1.45 8 1.91</td>
</tr>
<tr>
<td>Road networks (RNs)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rCA roadNet-CA</td>
<td>1.9M 5.5M</td>
<td>≈1 2 1.59</td>
</tr>
<tr>
<td>rTX roadNet-TX</td>
<td>1.3M 3.8M</td>
<td>≈1 2 1.53</td>
</tr>
<tr>
<td>rPA roadNet-PA</td>
<td>1M 3M</td>
<td>≈1 2 1.52</td>
</tr>
<tr>
<td>Citation graphs (CsGs)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ciP cit-Patents</td>
<td>3.7M 16.5M</td>
<td>1.16 8 1.57</td>
</tr>
<tr>
<td>Web graphs (WGs)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wGL web-Google</td>
<td>875k 5.1M</td>
<td>1.78 12 2.08</td>
</tr>
<tr>
<td>wBS web-BerkStan</td>
<td>685k 7.6M</td>
<td>1.91 24 1.91</td>
</tr>
<tr>
<td>wSF web-Stanford</td>
<td>281k 2.3M</td>
<td>1.89 24 1.89</td>
</tr>
</tbody>
</table>
OUTPERFORMING STATE-OF-THE-ART

 уникалу табліці.

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

Average overall speedup (geometric mean) over Graph500: 1.07, Galois: 1.40, HAMA: ~1000

1.85x on average, up to 4.3x
OUTPERFORMING STATE-OF-THE-ART
SCALABILITY ANALYSIS: DISTRIBUTED-MEMORY

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
OTHER ANALYSES
CONCLUSIONS

Atomic Active Messages

Combine the advantages of Active Messages and HTM

Illustrate HTM’s advantages in performance, next to programmability

Deliver the hierarchy of atomic messages that covers various graph algorithms

Detailed performance analysis

Derive close-to-optimal transaction sizes for Haswell & BG/Q

Model & analyze performance tradeoffs

Accelerating state-of-the-art

Average speedup 1.85x
Up to 4x
Thank you for your attention
DISTRIBUTED HTM TRANSACTIONS

Numbers are the local + remote marked vertices.
Can we amortize HTM transactions’ transfer overheads with coalescing?

Yes, we can!
**Single-Vertex Transactions**

**Incrementing Vertex Rank**

- More aborts
- Lower contention (10 accesses/vertex)
- Higher contention (100 accesses/vertex)
- Atomics always outperform HTM

The reason: each transaction always modifies some memory cell, increasing the number of conflicts.
OUTPERFORMING STATE-OF-THE-ART
BLUEGENE/Q IBM

Average speedup: 1

Average overall speedup over Graph500 (geometric mean): 1.51 (1.85)

Average speedup: 3.20

Average speedup: 1.85

The same transaction size for all graphs

The same transaction sizes for each graph separately

Best transaction size: ~24-100 vertices accessed
OUTPERFORMING STATE-OF-THE-ART
SCALABILITY ANALYSIS: SHARED-MEMORY

Contention from atomics dominates the runtime of Graph500

Implementation
- AAM
- Graph500

Framework
- HAMA
- Galois
- Graph500
- AAM