Hardware-Accelerated Memory Operations on Large-Scale NUMA Systems

Markus Dreseler, Timo Djürken, Matthias Uflacker, Hasso Plattner

Thomas Kissinger, Eric Lübke, Dirk Habich, Wolfgang Lehner

Hasso Plattner Institut

Technische Universität Dresden
Scaling beyond four sockets
Scaling beyond four sockets

How can multiple blades form a scale-up system?
Scaling beyond four sockets - the SGI way

CPU 0
DRAM

CPU 1
QPI
DRAM

CPU 2

CPU 3
QPI

CPU 0
HARP

CPU 1

CPU 2

CPU 3
HARP

NUMA link to other blades
Scaling beyond four sockets - the SGI way

CPUs still use QPI
Scaling beyond four sockets - the SGI way
Scaling beyond four sockets - the SGI way

An eight blade system has

- $8 \times 4 = 32$ sockets
- $32$ TB of cache-coherent memory
- $32 \times 15 = 480$ physical cores (with Ivy Bridge)

The NL7 link has a transfer rate of 7.47 GB/s (QPI here 16 GB/s)

<table>
<thead>
<tr>
<th>Latency</th>
<th>Local</th>
<th>1 hop (same blade)</th>
<th>2 hops (same blade)</th>
<th>Different blade</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100 ns</td>
<td>200 ns</td>
<td>250 ns</td>
<td>&lt; 500 ns</td>
</tr>
</tbody>
</table>

87.5% of memory is on a different blade
HARP and GRU

- HARP
- CPU 0
  - DRAM
- CPU 1
  - DRAM
  - QPI
- CPU 2
  - DRAM
- CPU 3
  - DRAM
- NUMAlink to other blades

Global Reference Unit (GRU)
- handles remote addresses
- translates QPI and NUMAlink
- maintains cache coherency
- can be programmed
GRU API Features

- Memory Transfer
- Atomic Memory Operations (AMO)
- Message Passing
- Operative
Research Questions

How can we use the GRU to hide the cost of accessing remote memory?

More general: How can special hardware help in optimizing memory accesses?
GRU API Features

- Memory Transfer
- Atomic Memory Operations (AMO)
- Message Passing
- Operative
gru_bcopy

gru_bcopy works similar to memcpy:

```c
void* memcpy(dest, src, n);

void gru_bcopy(control_block, src, dest, tri0,
                 element_type, num_elements, buffer_size, hints);
```
gru_bcopy

Other than memcpy, it is invoked asynchronously

```c
void* memcpy(dest, src, n);

void gru_bcopy(control_block, src, dest, tri0, element_type, num_elements, buffer_size, hints);

void gru_wait(control_block);
```
gru_bcopy vs memcpy

The graph compares the throughput (in GB/s) of `gru_bcopy` and `memcpy` as a function of data size (in bytes). The graph shows that `gru_bcopy` has a throughput that is more than 2 times greater than `memcpy` for large data sizes.

The x-axis represents the data size in bytes, ranging from $10^1$ to $10^9$. The y-axis represents the throughput in GB/s, ranging from 0 to 7.

The graph indicates that `gru_bcopy` surpasses `memcpy` in performance for larger datasets, with a notable advantage of more than 2 times for very large data sizes.
Using *gru_bcopy*

We have a method that transfers a *large, contiguous block of memory* and

- executes asynchronously
- has a higher throughput than regular CPU reads

How can we use that in an In-Memory Database?

➔ Where do we read and process large amounts of contiguous memory?
GRU-accelerated remote table scan

- We modified the existing table scan for Hyrise and a commercial database
- Instead of loading the remote data, we use GRU-based double buffering
- The existing scan code does not have to be modified
Benchmarks

![Graph showing time spent per row vs. number of 10 bit entries in column (log)].

The graph illustrates the time spent per row for both 'regular' and 'GRU' methods as the number of 10 bit entries in a column increases. The graph shows a significant improvement, with a 30% improvement indicated by the vertical arrow. The Break-Even Point is marked, indicating where the 'GRU' method becomes more efficient than the 'regular' method.
GRU API Features

- Memory Transfer
- Atomic Memory Operations (AMO)
- Message Passing
- Operative
Atomic Counter Increments

Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores

Xiangyao Yu
MIT CSAIL
xyy@csail.mit.edu
George Bezerra
MIT CSAIL
gbezerra@csail.mit.edu
Andrew Pavlo
Carnegie Mellon University
pavlo@cs.cmu.edu
Srinivas Devadas
MIT CSAIL
devadas@csail.mit.edu
Michael Stonebraker
MIT
sten@csail.mit.edu

<table>
<thead>
<tr>
<th></th>
<th>2PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL_DETECT</td>
<td>Scales under low-contention. Suffers from lock thrashing.</td>
</tr>
<tr>
<td>NO_WAIT</td>
<td>Has no centralized point of contention. Highly scalable. Very high abort rate.</td>
</tr>
<tr>
<td>WAIT_DIE</td>
<td>Suffers from lock thrashing and timestamp bottleneck.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>High overhead from copying data locally. Non-blocking writes. Suffers from timestamp bottleneck.</td>
</tr>
<tr>
<td>MVCC</td>
<td>Performs well w/ read-intensive workload. Non-blocking reads and writes. Suffers from timestamp bottleneck.</td>
</tr>
<tr>
<td>OCC</td>
<td>High overhead for copying data locally. High abort cost. Suffers from timestamp bottleneck.</td>
</tr>
</tbody>
</table>

Table 2: A summary of the bottlenecks for each concurrency control scheme evaluated in Section 5.
GRU-supported Counters

Problem 1: DoubleWord is cached by all processors

Problem 2: All blades have to communicate to keep counter in sync
GRU-supported Counters

Solution 1: DoubleWord is held in one place

Solution 2: Blades only talk to a single GRU
GRU-supported Counters

4x improvement, 10x in other configurations
GRU-supported Latches
Future Work

● Use bcopy for more database operations
  ○ NUMA Rebalancing
  ○ Joins

● Evaluate Atomic Memory Operations for
  ○ Transaction Counters
  ○ Latches

Please contact us if you have more ideas!
Conclusions

● Programming memory interconnects reduce the cost of high NUMA distances

● By using asynchronous memory transfers and double-buffering, we improved the throughput of remote table scans by up to 30%
  ○ Limitations apply, especially with regards to the table size

● By having the interconnect handle memory operations, we increased the throughput of atomic counters by up to 10x and of latches by up to 8x

Contact: markus.dreseler@hpi.de
Backup
Related Work

- The GRU has not yet been used in a DBMS context.

- Rödiger et al. [1] talk about offloading memory transfers in scale-out systems.

- So does Vaidyanathan et al. [2], reducing CPU costs using Intel’s I/OAT.

- Hardware Counter Increments, where the increments are done remotely, are proposed and simulated by Yu et al. [3].

People like scaling up for a number of reasons

- Lower latency
- Existing DBMS can be used
- Lower TCO, less administrative effort
- Performance advantages for some settings

There is a value to both scale-up and scale-out. Let’s focus on really big scale-up systems though.