Hardware-Accelerated Memory Operations on Large-Scale NUMA Systems

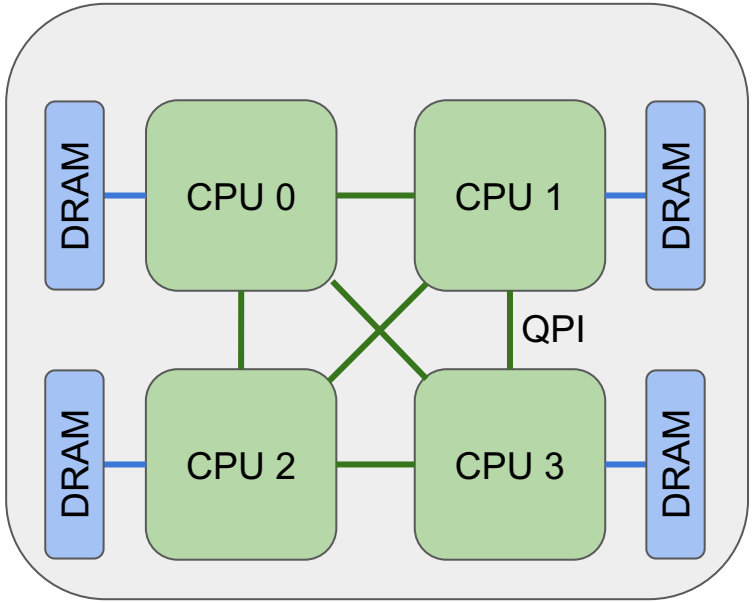
Markus Dreseler, Timo Djürken,
Matthias Uflacker, Hasso Plattner

Hasso Plattner Institut
Digital Engineering · Universität Potsdam

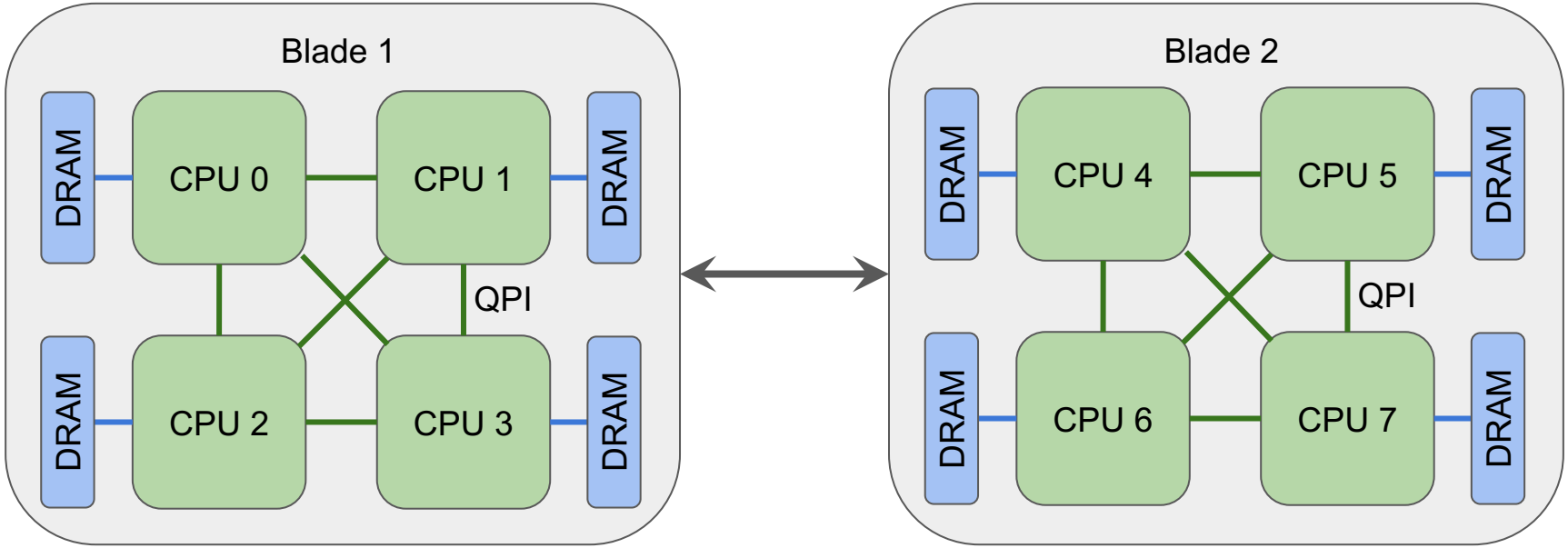
Thomas Kissinger, Eric Lübke,
Dirk Habich, Wolfgang Lehner

TECHNISCHE UNIVERSITÄT DRESDEN

Scaling beyond four sockets

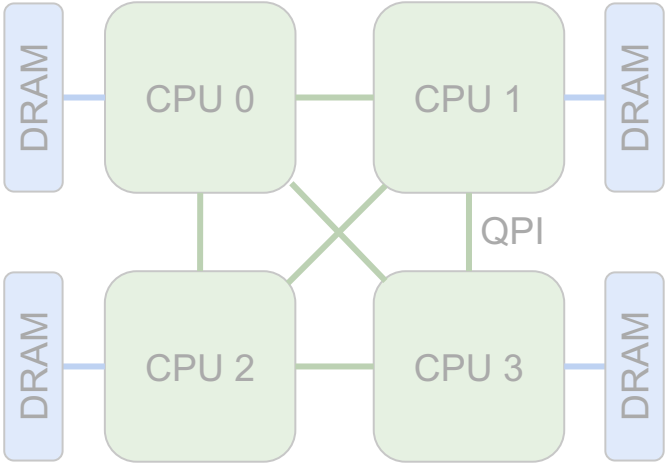


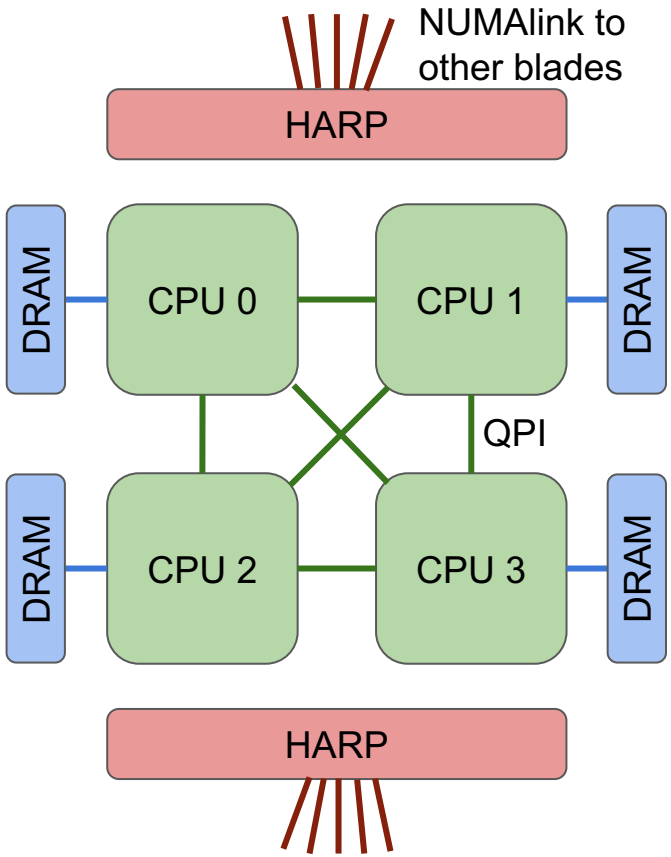
Scaling beyond four sockets



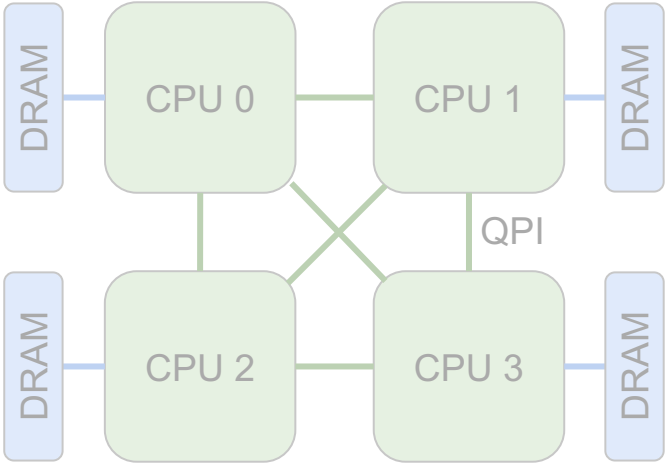
How can multiple blades form a scale-up system?

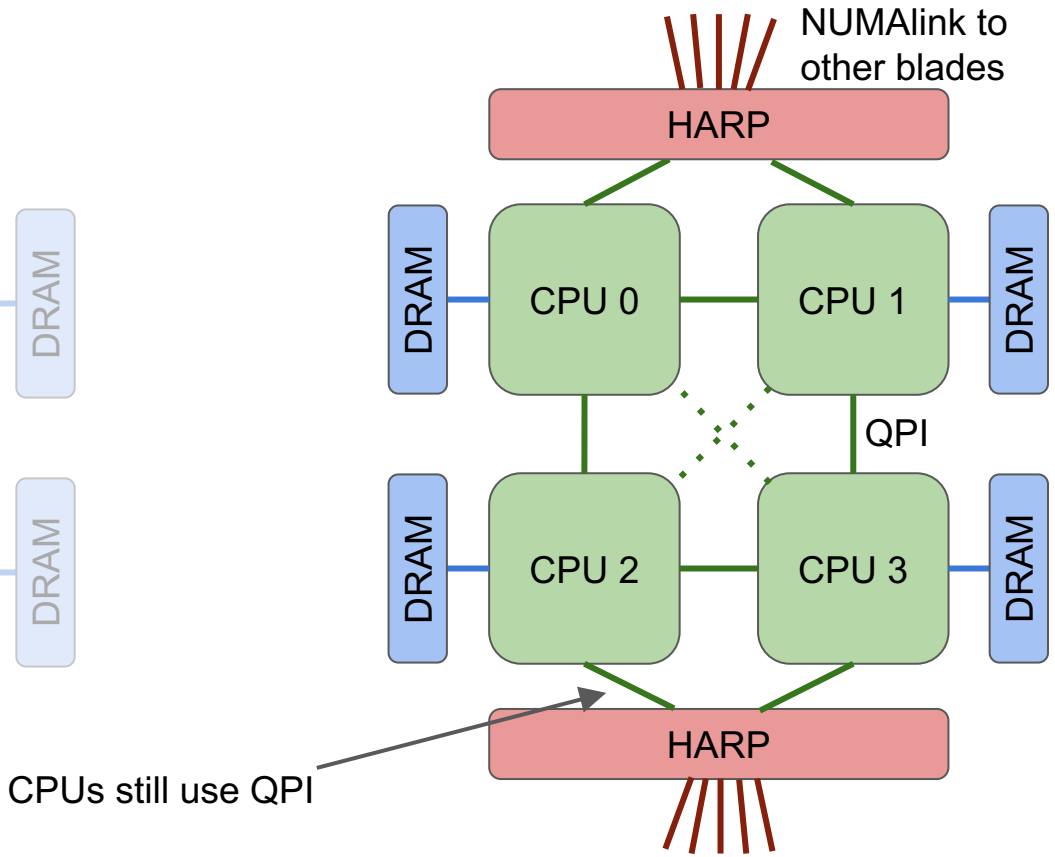
Scaling beyond four sockets - the SGI way



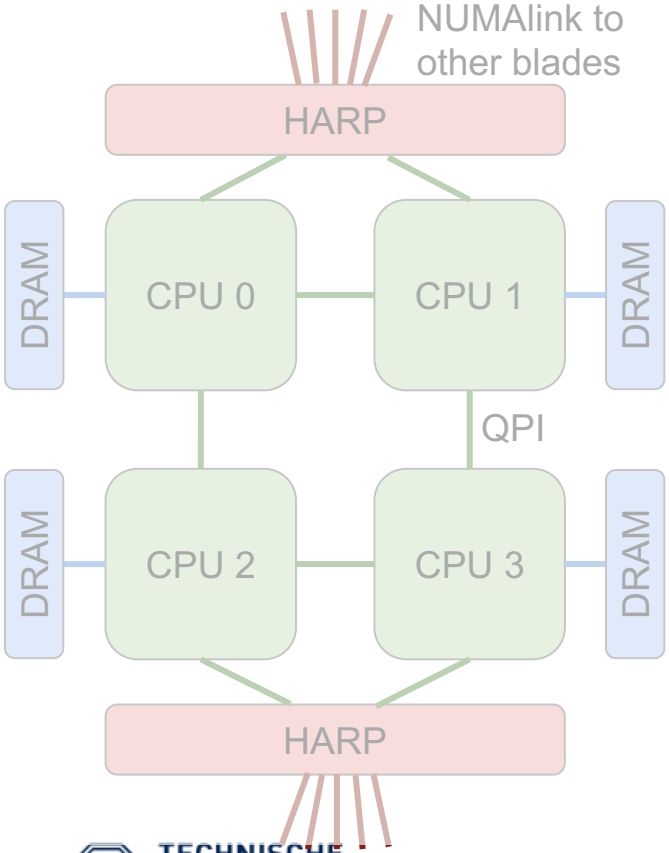


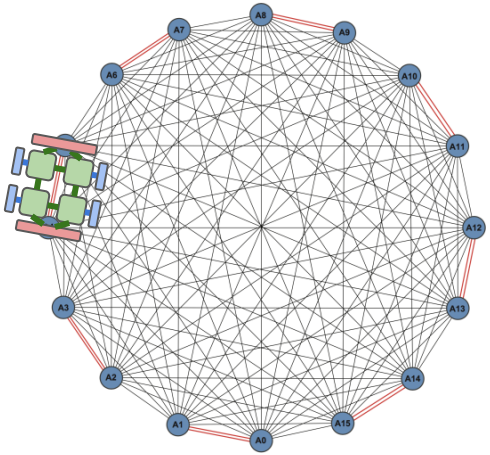
Scaling beyond four sockets - the SGI way





Scaling beyond four sockets - the SGI way



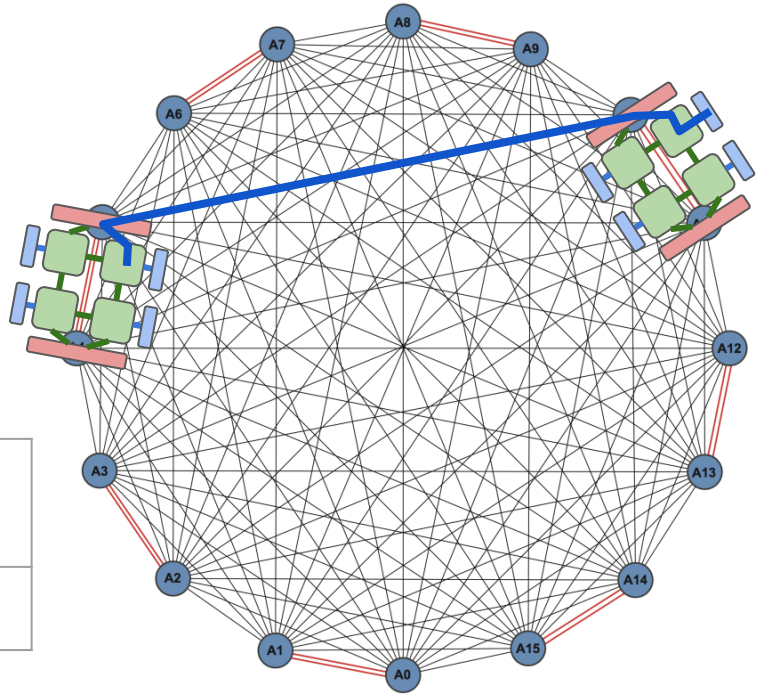


4-Socket (4TB) 8-Socket (8TB) 16-Socket (16TB) 20-Socket (20TB) 32-Socket (32TB)

Scaling beyond four sockets - the SGI way

An eight blade system has

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



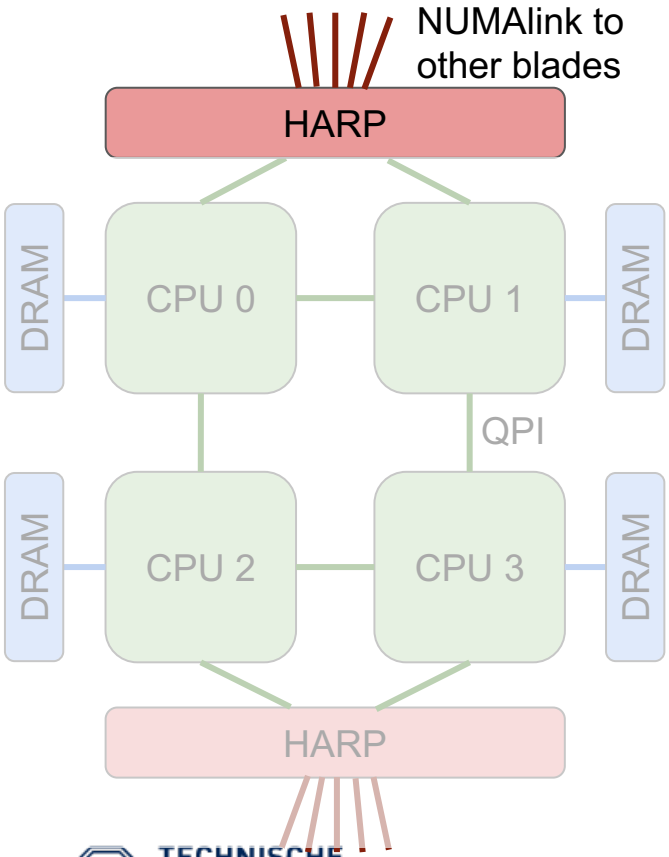
The NL7 link has a transfer rate of 7.47 GB/s (QPI here 16 GB/s)

Latency

Local	1 hop (same blade)	2 hops (same blade)	Different blade
100 ns	200 ns	250 ns	< 500 ns

87.5% of memory is on a different blade

HARP and GRU



HARP

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:

```
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

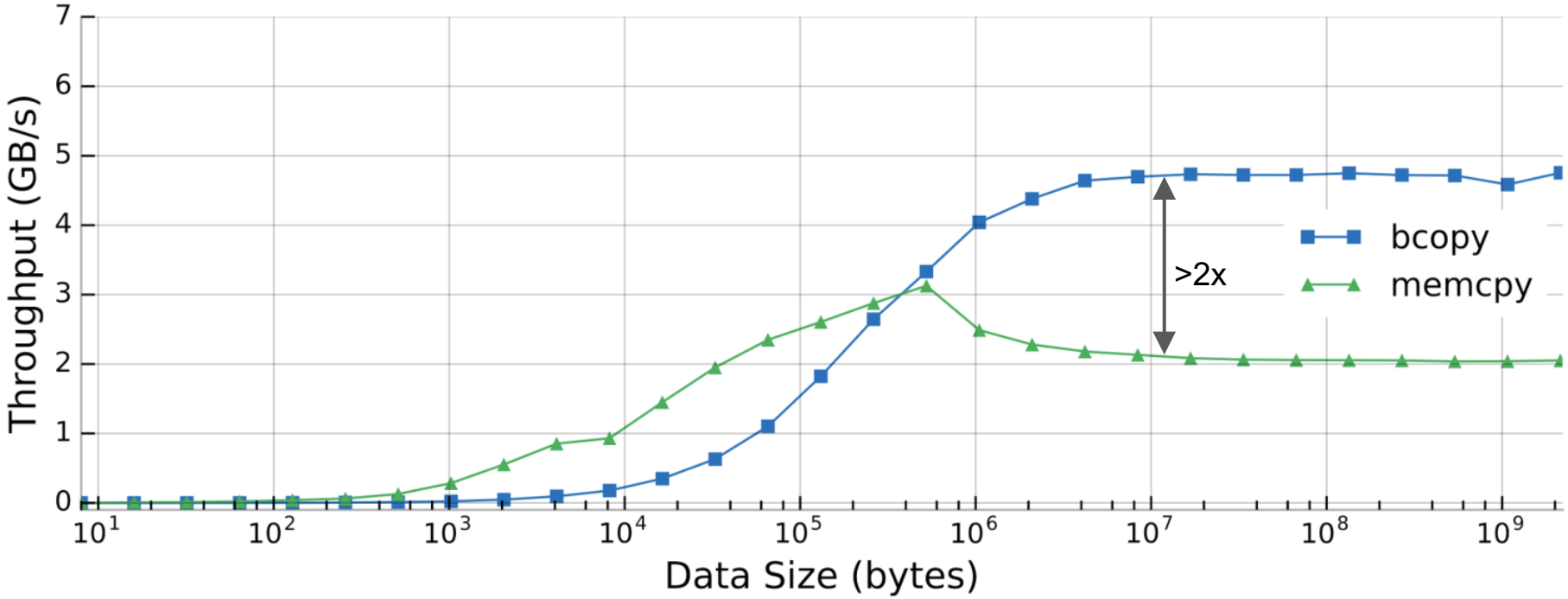
```
void* memcpy(dest, src, n);



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

void gru_wait(control_block);
```

gru_bcopy vs memcpy



Using gru_bcopy

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

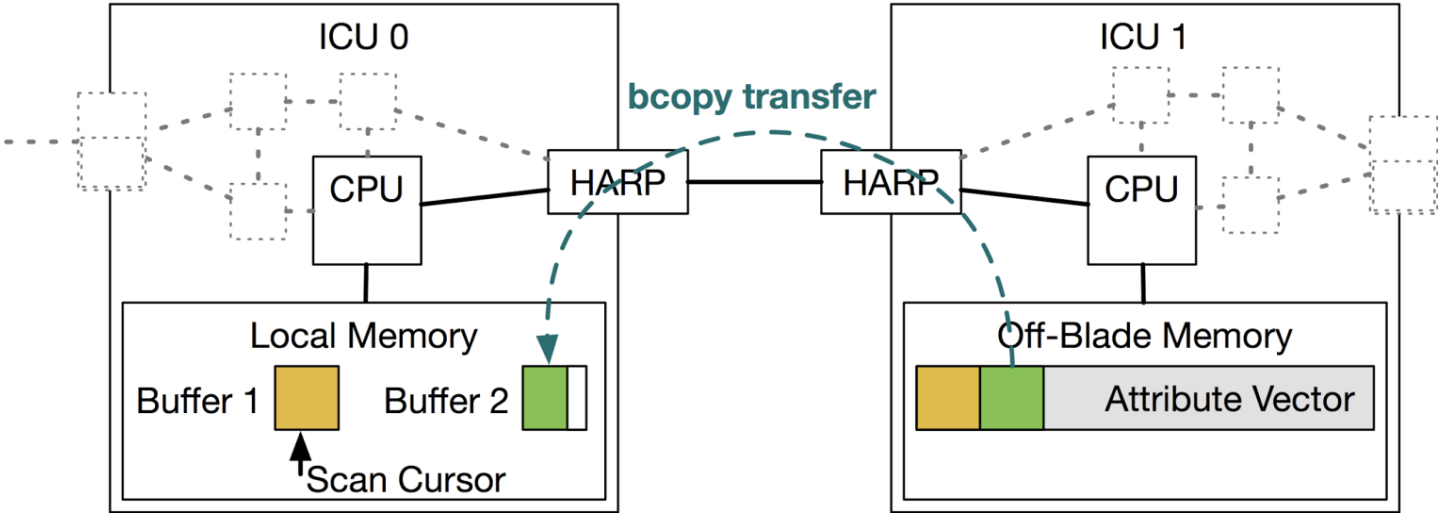
- executes asynchronously
- has a higher throughput than regular CPU reads

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

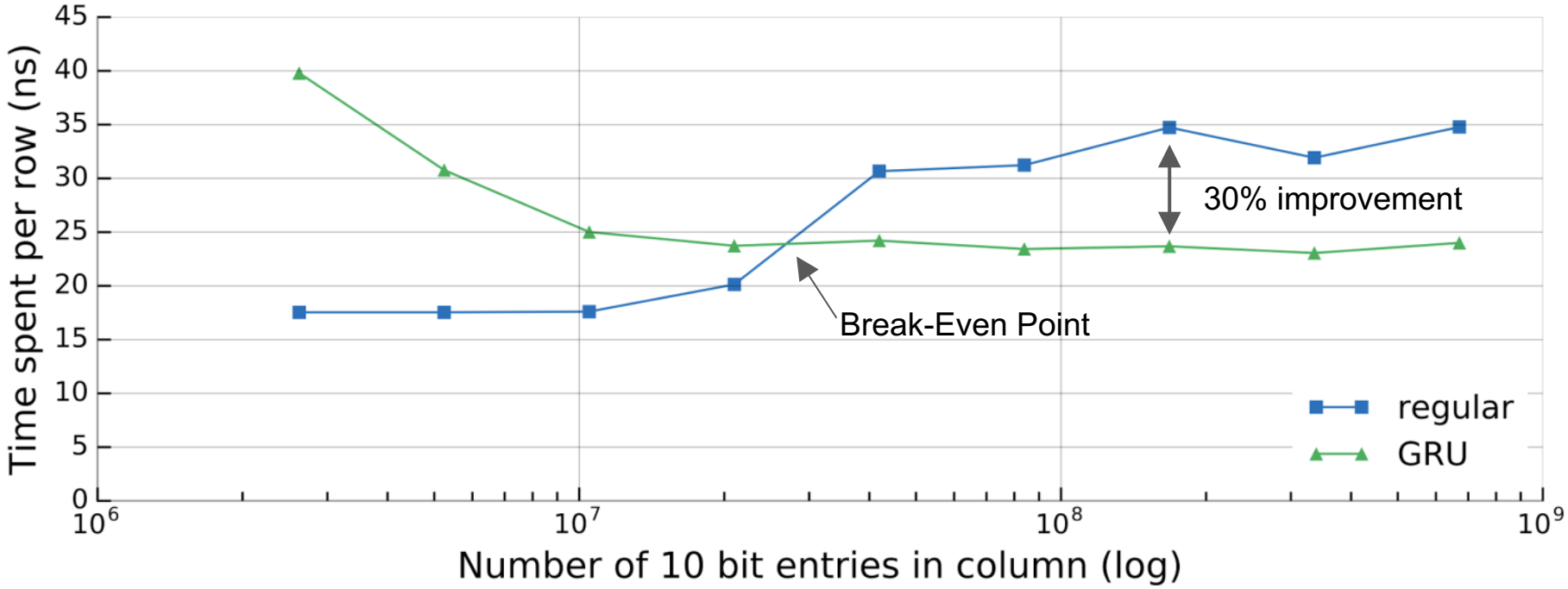
➔Where do we read and process large amounts of contiguous memory?

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


Time spent per row (ns)
45
40
35
30
25
20
15
10
5
0
10^6
10^7
10^8
10^9
Number of 10 bit entries in column (log)
30% improvement
Break-Even Point
regular
GRU

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
yxy@csail.mit.edu

George Bezerra
MIT CSAIL
gbezerra@csail.mit.edu

Andrew Pavlo
Carnegie Mellon University
pavlo@cs.cmu.edu

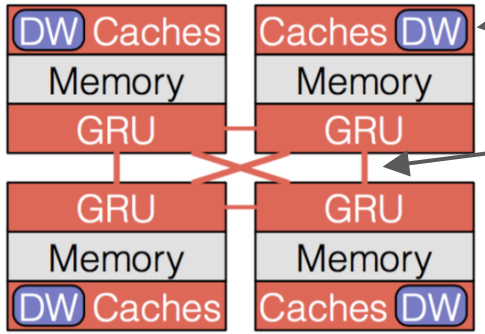
Srinivas Devadas
MIT CSAIL
devadas@csail.mit.edu

Michae
M
stonebrake

2PL	**DL_DETECT**	Scales under low-contention. Suffers from lock thrashing.
	NO_WAIT	Has no centralized point of contention. Highly scalable. Very high abort rate.
	WAIT_DIE	Suffers from lock thrashing and timestamp bottleneck.
T/O	**TIMESTAMP**	High overhead from copying data locally. Non-blocking writes. Suffers from timestamp bottleneck.
	MVCC	Performs well w/ read-intensive workload. Non-blocking reads and writes. Suffers from timestamp bottleneck.
	OCC	High overhead for copying data locally. High abort cost. Suffers from timestamp bottleneck.
	H-STORE	The best algorithm for partitioned workloads. Suffers from multi-partition transactions and timestamp bottleneck.

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

GRU-supported Counters



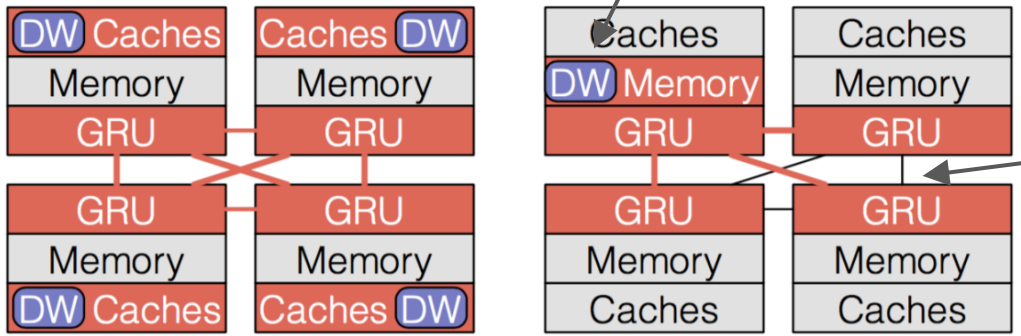
(a) CPU

Problem 1: DoubleWord is cached by all processors

Problem 2: All blades have to communicate to keep counter in sync

GRU-supported Counters

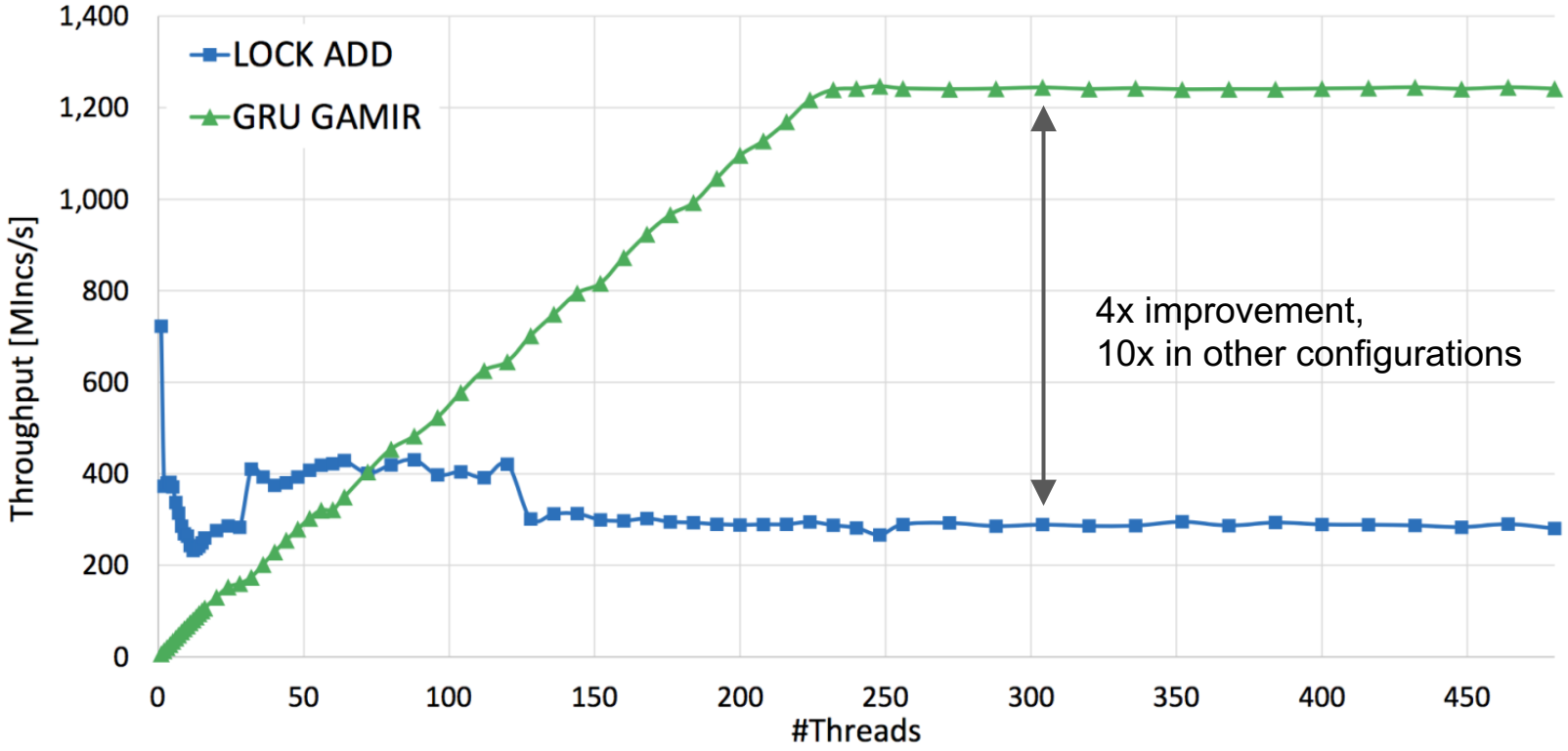
Solution 1: DoubleWord is held in one place



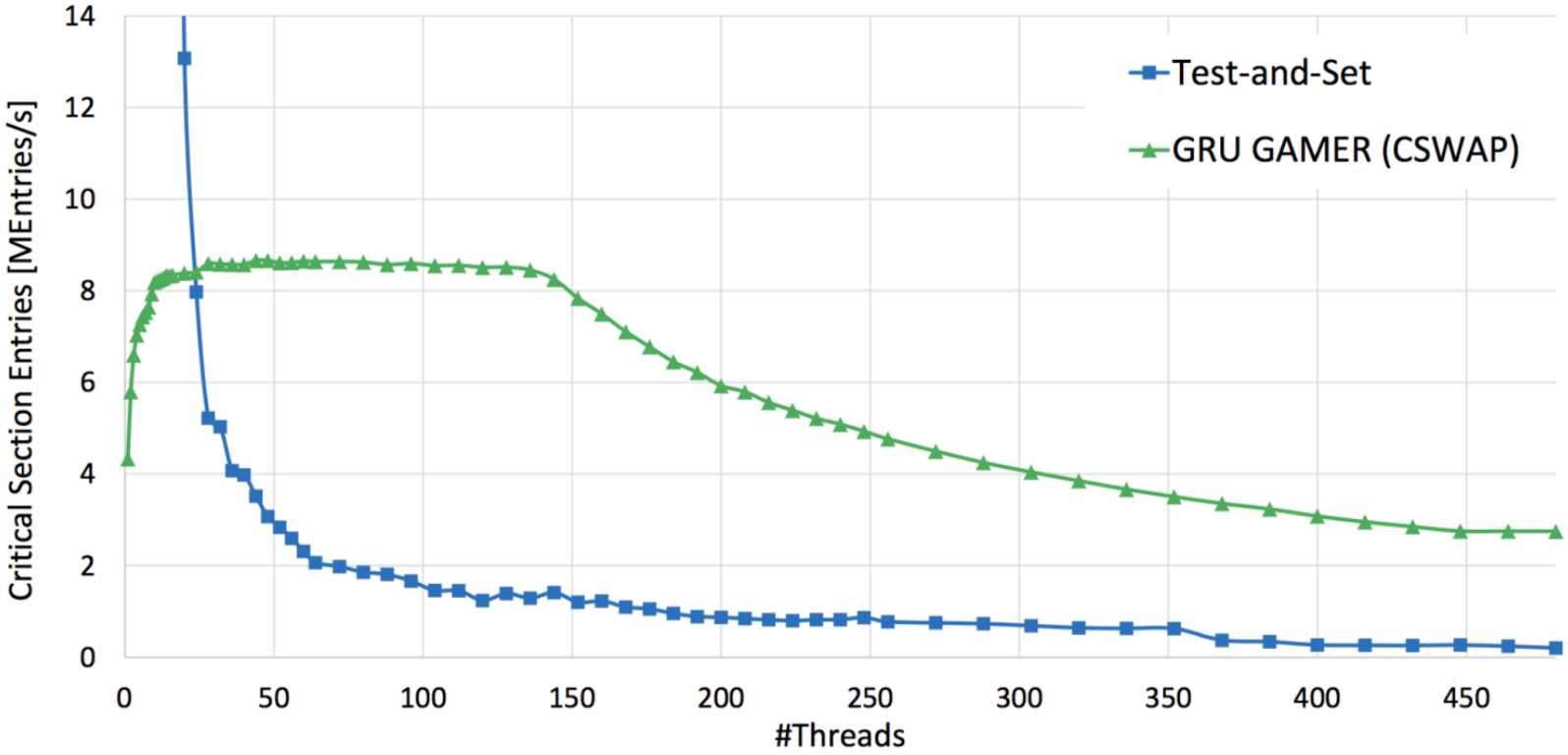
(a) CPU (b) GRU

Solution 2: Blades only talk to a single GRU

GRU-supported Counters



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].

[1] Wolf Rödiger et al. "High-Speed Query Processing over High-Speed Networks". In: PVLDB 2015.
[2] Karthikeyan Vaidyanathan et al. "Efficient Asynchronous Memory Copy Operations on Multi-Core Systems and I/OAT". In: IEEE Cloud 2007.
[3] Yu et al., "Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores", In: PVLDB 2014

Scale-Up vs Scale-Out

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.