High-Performance Distributed RMA Locks

Patrick Schmid, Maciej Besta, Torsten Hoefler
NEED FOR EFFICIENT LARGE-SCALE SYNCHRONIZATION
**Locks**

- Inuitive semantics
- Various performance penalties

An example structure

Diagram:
- Proc p
- Proc q
- Lock
- Accesses
- Lock
- Unlock
- Accesses

**ETH Zürich**
LOCKS: CHALLENGES
LOCKS: CHALLENGES

- We need intra- and inter-node topology-awareness
- We need to cover arbitrary topologies
We need to distinguish between readers and writers

We need flexible performance for both types of processes

What will we use in the design?
WHAT WE WILL USE
MCS Locks

Proc
Cannot enter
Next proc

Proc
Cannot enter
Next proc

Proc
Can enter
Next proc

Proc
Can enter
Next proc

...
WHAT WE WILL USE
Reader-Writer Locks

![Reader-Writer Lock Diagram]
How to manage the design complexity?

How to ensure tunable performance?

What mechanism to use for efficient implementation?
REMOTE MEMORY ACCESS (RMA) PROGRAMMING

Process p

Memory

A

B

Process q

Memory

A

B

A put

get B

flush

Cray
BlueWaters
REMOTE MEMORY ACCESS PROGRAMMING

- Implemented in hardware in NICs in the majority of HPC networks support RDMA
How to manage the design complexity?

How to ensure tunable performance?

What mechanism to use for efficient implementation?
How to manage the design complexity?

Each element has its own distributed MCS queue (DQ) of writers.

Readers and writers synchronize with a distributed counter (DC).

MCS queues form a distributed tree (DT).

Modular design.
How to ensure tunable performance?

Each DQ: fairness vs throughput of writers

DC: a parameter for the latency of readers vs writers

DT: a parameter for the throughput of readers vs writers

A tradeoff parameter for every structure
**DISTRIBUTED MCS QUEUES (DQs)**

**Throughput vs Fairness**

- Larger $T_{L,i}$: more throughput at level $i$.
- Smaller $T_{L,i}$: more fairness at level $i$.

Each DQ: The maximum number of lock passings within a DQ at level $i$, before it is passed to another DQ at $i$. $T_{L,i}$
DISTRIBUTED TREE OF QUEUES (DT)
Throughput of readers vs writers

DT: The maximum number of consecutive lock passings within readers ($T_R$).
DISTRIBUTED COUNTER (DC)
Latency of readers vs writers

DC: every \( k \)th compute node hosts a partial counter, all of which constitute the DC.

\[ k = T_{DC} \]

A writer holds the lock

Readers that arrived at the CS

Readers that left the CS

\[ T_{DC} = 1 \]
\[ T_{DC} = 2 \]

0|9|7
R1
R2
0|3|1
R3
R4
0|8|5
R5
R6
0|5|3
R8
R9
0|12|8
0|13|8
THE SPACE OF DESIGNS

Higher throughput of writers vs readers

Design A
Design B

Locality vs fairness (for writers)

Lower latency of writers vs readers

$T_{DC}$

$T_{L,i}$

$T_R$
**LOCK ACQUIRE BY READERS**

A lightweight acquire protocol for readers: only one atomic fetch-and-add (FAA) operation.

- A writer holds the lock
- Readers that arrived at the CS
- Readers that left the CS

Diagram:
- FAA
- R1
- R2
- R3
- R4
- 0|9|7
- 0|3|1
- 0|8|7
- 0|9|7
- b|x|y
**Lock Acquire by Writers**

Acquire the main lock

Acquire the main MCS lock

Acquire MCS

1|9|9

1|3|3

1|8|8

1|5|5

Acquire MCS
EVALUATION

- CSCS Piz Daint (Cray XC30)
- 5272 compute nodes
- 8 cores per node
- 169TB memory
**EVALUATION**
**CONSIDERED BENCHMARKS**

- **The latency benchmark**
- **DHT**
  - Distributed hashtable evaluation
- **Throughput benchmarks:**
  - Empty-critical-section
  - Single-operation
  - Wait-after-release
  - Workload-critical-section
EVALUATION
DISTRIBUTED COUNTER ANALYSIS

Throughput, 2% writers
Single-operation benchmark

Throughput [mln locks/s]

\[ T_{DC} \]

- 64
- 32
- 16
- 8
- 4
- 2

MPI processes (P)

16 64 256 1024
EVALUATION
READER THRESHOLD ANALYSIS

Throughput, 0.2% writers,
Empty-critical-section benchmark

Throughput [mln locks/s] vs MPI processes (P)

- $T_R$: 6000
- $T_R$: 5000
- $T_R$: 4000
- $T_R$: 3000
- $T_R$: 2000
- $T_R$: 1000
EVALUATION
COMPARISON TO THE STATE-OF-THE-ART

EVALUATION
COMPARISON TO THE STATE-OF-THE-ART

Throughput, single-operation benchmark

Percentages are values of $F_W$

EVALUATION DISTRIBUTED HASHTABLE

20% writers

10% writers

EVALUATION DISTRIBUTED HASHTABLE

2% of writers

0% of writers

OTHER ANALYSES
CONCLUSIONS

Modular distributed RMA lock, correctness with SPIN. Parameter-based design, feasible with various RMA libraries/languages. Improves latency and throughput over state-of-the-art. Enables high-performance distributed hashtabled.

Thank you for your attention.
DISTRIBUTED TREE OF QUEUES (DT)
Throughput of readers vs writers

DT: The maximum number of consecutive lock passings within writers ($T_W$) and readers ($T_R$).

$$T_W = \prod_{i=1}^{N} T_{L,i}$$
THE SPACE OF DESIGNS

Higher throughput of writers vs readers

Locality vs fairness (for writers)

Design A

Design B

Lower latency of writers vs readers

$T_{DC}$

$T_{L,i}$

$T_R$
EVALUATION
D-MCS vs OTHERS

Latency (LB)

Throughput (ECSB)

- Performance initially increases due to high intra-node bandwidth

- Scheme
  - foMPI-Spin
  - D-MCS
  - RMA-MCS

<table>
<thead>
<tr>
<th>Scheme</th>
<th>foMPI-Spin</th>
<th>D-MCS</th>
<th>RMA-MCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>intra-node</td>
<td>□</td>
<td>△</td>
<td>□</td>
</tr>
<tr>
<td>inter-node</td>
<td>□</td>
<td>△</td>
<td>□</td>
</tr>
</tbody>
</table>

Latency [μs]

Throughput [mln locks/s]
**Evaluation**

**Writer Threshold Analysis**

Throughput, 25% of writers

Single-operation benchmark

Throughput [mln locks/s] vs. MPI processes (P)

- $T_{Li}$ product
  - 500
  - 1000
  - 2500
  - 5000
  - 7500

- Throughput peaks at different points for each $T_{Li}$ product.
EVALUATION
FAIRNESS VS THROUGHPUT ANALYSIS

Throughput, 25% of writers, Single-operation benchmark

Throughput [mln locks/s]

MPI processes (P)

\[ T_{Li} \]
- 50–20
- 25–40
- 10–100
EVALUATION
READER THRESHOLD ANALYSIS

Throughput, 2% and 5% writers,
Empty-critical-section benchmark

Throughput [mln locks/s]

$T_R$
- 3000-2
- 4000-2
- 5000-2
- 3000-5
- 4000-5
- 5000-5

MPI processes (P)
## Feasibility Analysis

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Put</td>
<td>UPC_SET</td>
<td>bupc_atomicX_set_RS</td>
<td>shmem_swap</td>
</tr>
<tr>
<td>Get</td>
<td>UPC_GET</td>
<td>bupc_atomicX_read_RS</td>
<td>shmem_mswap</td>
</tr>
<tr>
<td>Accumulate</td>
<td>UPC_INC</td>
<td>bupc_atomicX_fetchadd_RS</td>
<td>shmem_fadd</td>
</tr>
<tr>
<td>FAO (SUM)</td>
<td>UPC_INC, UPC_DEC</td>
<td>bupc_atomicX_fetchadd_RS</td>
<td>shmem_fadd</td>
</tr>
<tr>
<td>FAO (REPLACE)</td>
<td>UPC_SET</td>
<td>bupc_atomicX_swap_RS</td>
<td>shmem_swap</td>
</tr>
<tr>
<td>CAS</td>
<td>UPC_CSWAP</td>
<td>bupc_atomicX_cswap_RS</td>
<td>shmem_cswap</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Fortran 2008 [27]</th>
<th>Linux RDMA/IB [33, 43]</th>
<th>iWARP [18, 41]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put</td>
<td>atomic_define</td>
<td>MskCmpSwap</td>
<td>masked CmpSwap</td>
</tr>
<tr>
<td>Get</td>
<td>atomic_ref</td>
<td>MskCmpSwap</td>
<td>masked CmpSwap</td>
</tr>
<tr>
<td>Accumulate</td>
<td>atomic_add</td>
<td>FetchAdd</td>
<td>FetchAdd</td>
</tr>
<tr>
<td>FAO (SUM)</td>
<td>atomic_add</td>
<td>FetchAdd</td>
<td>FetchAdd</td>
</tr>
<tr>
<td>FAO (REPLACE)</td>
<td>atomic_define*</td>
<td>MskCmpSwap</td>
<td>masked CmpSwap</td>
</tr>
<tr>
<td>CAS</td>
<td>atomic_cas</td>
<td>CmpSwap</td>
<td>CmpSwap</td>
</tr>
</tbody>
</table>