Nonblocking Epochs in MPI One-Sided Communication

Judicael A. Zounmevo, Xin Zhao, Pavan Balaji, William Gropp, Ahmad Afsahi

Queen’s University, Kingston, ON, Canada
University of Illinois at Urbana-Champaign
Argonne National Laboratory
Outline

- MPI One-Sided: Epochs and Synchronizations
- Synchronization Burden
- Nonblocking Synchronizations
- Semantics
- Visualizing Improvements Scenarios
- Design Notes
- Evaluation
- Conclusion
MPI One-Sided: Epoch and Synchronizations

One-sided communications

Origin
Process0

Target
Process1

Origin
Process2

Target exposes memory for remote access

There are many kinds of epochs: Fence, GATS, lock, lock_all

Different synchronization routines lead to different epoch types

Figure adapted from Tipparaju, V.; Gropp, W.; Ritzdorf, H.; Thakur, R.; Traff, J.L., "Investigating High Performance RMA Interfaces for the MPI-3 Standard," ICPP, pp.293,300, 22-25 Sept. 2009
Synchronization Burden: Inefficiency Patterns

- Late Post \[1\]
- Late Complete \[1\]
- Early Fence \[1\]

Synchronization Burden: Inefficiency Patterns

- Wait at Fence [1]

- Late Unlock

Late Unlock is introduced in this work
Synchronization Burden: Analysis

Critical section thinking ↔ Tradeoff → Communication latency mitigation thinking

Ideal scenario … which is an utopia

Legend:
- Overlapped work
- Work
- RMA transfer
- MPI_WIN_COMPLETE entrance
- MPI_WIN_COMPLETE exit
- MPI_WIN_WAIT earliest exit
- Blocking time
- RMA call
- Initiation of nonblocking version of MPI_WIN_COMPLETE
- Completion of nonblocking version of MPI_WIN_COMPLETE
- End of RMA data transfer

Scenario 1:
- Origin
- Target

Scenario 2:
- Origin
- Target

Scenario 3:
- Origin
- Target

Critical section thinking

Tradeoff

Communication latency mitigation thinking

Ideal scenario … which is an utopia

Legend:
- Overlapped work
- Work
- RMA transfer
- MPI_WIN_COMPLETE entrance
- MPI_WIN_COMPLETE exit
- MPI_WIN_WAIT earliest exit
- Blocking time
- RMA call
- Initiation of nonblocking version of MPI_WIN_COMPLETE
- Completion of nonblocking version of MPI_WIN_COMPLETE
- End of RMA data transfer

Scenario 1:
- Origin
- Target

Scenario 2:
- Origin
- Target

Scenario 3:
- Origin
- Target
Nonblocking Synchronizations

We proposed nonblocking synchronizations

… to completely remove the aforementioned tradeoff

All epoch-opening, epoch-closing and flush routines are covered
“Open” and “close” define the boundaries of the application-level lifetime of the epoch. The epoch is said to be open inside these boundaries.

Open epoch $e_0$

Close epoch $e_0$

“Activation” and “completion” define the boundaries of the middleware-level lifetime of the epoch. The epoch is said to be active inside these boundaries.

Activate epoch $e_0$
Progression of $e_0$ starts

Complete epoch $e_0$.
e_0$ is done being progressed

Between opening and activation, the epoch is said to be deferred.
Default Semantics of Nonblocking Synchronizations

- Epoch closing is **not** completion

- In active target, access epochs and exposure epochs are matched in a FIFO fashion.

- Epochs for a given RMA window are always activated serially; in the order where they become pending inside the progress engine.

- Epoch $e_{k+1}$ is activated only after epoch $e_k$ has completed.
Semantics of Nonblocking Synchronizations with Progress Engine Optimizations

- One can do better than “activating $e_{k+1}$ only after $e_k$ completion”
- We provide info object key-value to enable out-of-order message progression:
  - MPI_WIN_ACCESS_AFTER_ACCESS_REORDER (A_A_A_R)
  - MPI_WIN_ACCESS_AFTER EXPOSURE_REORDER (A_A_E_R)
  - MPI_WIN_EXPOSURE_AFTER_EXPOSURE_REORDER (E_A_E_R)
  - MPI_WIN_EXPOSURE_AFTER_ACCESS_REORDER (E_A_A_R)

Fence and win_lock_all are currently not covered by the optimization flags
Visualizing Improvement Scenarios

MPI-3.0 RMA
Blocking epoch-closing synchronization

Nonblocking synchronizations. Default progress engine behavior.

Nonblocking synchronizations. Progress engine optimizations enabled.
We had to redesign from scratch MPI-3.0 RMA as a whole along with the nonblocking synchronizations and the new concepts.
Design Notes: Deferring, Recording and Replaying Epochs

Epoch $e_1$ cannot be activated yet

Epoch $e_1$ is deferred

Deferred epoch being recorded

Replayed when activated
Design Notes: Deferring, Recording and Replaying Epochs
Design Notes: Epoch Matching

- Need to keep history of granted accesses from targets to origins
  - Message queues again? …. Scalability?
  - We Achieved all epoch matching in $O(1)$ for both time and space!
Evaluation: Setup

- 320 dual-socket quad-core 2.67GHz Intel Xeon x5550 processors (total of 2560 CPU cores)
- Cache:
  - 4 x 32KB L1 instruction
  - 4 x 32KB L1 data
  - 4 x 256KB L2
  - 8MB L3
- 36GB RAM / node
- InfiniBand: Mellanox ConnectX QDR HCAs
- Linux kernel 2.6.18
- MVAPICH 2-1.9 (Vanilla)
- MVAPICH 2-1.9 (with new RMA)
Evaluation: Fixing the Inefficiency Patterns

Late Post

- Target is 1000µs late in exposing its epoch
- Origin does 2 activities: Access epoch, then two-sided
- All RMA are MPI_PUT (1MB)
- The graph shows the origin latencies for the various activities

The nonblocking series overlaps the subsequent activity with the delay
Evaluation: Fixing the Inefficiency Patterns

Late Complete
- Origin overlaps 1000µs of work inside its epoch.
- The graph shows the target epoch length.

The nonblocking synchronization prevents the target from suffering the origin-side work latency.
Evaluation: Fixing the Inefficiency Patterns

Early fence

- Target is 1000µs late in exposing its epoch
- Acting target does 2 activities: Access epoch, then two-sided
- The graph shows the cumulative latency of the acting origin for both activities

The nonblocking series overlaps the subsequent activity with the epoch latency.
Evaluation: Fixing the Inefficiency Patterns

Wait a Fence

- Acting origin overlaps 1000µs of work inside its epoch.
- The graph shows the acting target epoch length.

Longer than data transfer time

The nonblocking synchronization prevents the acting target from suffering the origin-side work latency.
Evaluation: Fixing the Inefficiency Patterns

Late Unlock

- All RMA are MPI_PUT (1MB)
- Origin O₀ asks for the lock first (exclusively).
- Origin O₁ asks for the same lock ... after O₀ did
- Origin O₀ works for 1000μs before releasing the lock
- The graph shows the epoch latency for each origin

The nonblocking series prevents the first lock holder from propagating its work latency to the subsequent requester
Evaluation: Out-of-Order Epoch Completion

A_A_A_R, GATS

- All test series are nonblocking
- A single origin O opens an access towards targets T_0 and T_1 in that order
- T_0 is 1000µs late in exposing its epoch (Late Post)
- The graph shows the epoch length of T_1 and the cumulative length of both origin epochs.

A_A_A_R prevents T_1 from suffering the delay of T_0. The origin completes faster overall as well.
Evaluation: Out-of-Order Epoch Completion

A_A_A_R, lock/unlock

- All test series are nonblocking
- Origin O₁ requests a lock from target T₀ right after O₀ gets it exclusively.
- Origin O₁ then requests a subsequent lock from T₁.
- Origin O₀ works for 1000µs before releasing the lock of T₀
- The graph shows the cumulative epoch latency of O₁

The overall O₁ latency for both locks is lower when A_A_A_R is enabled.
Evaluation: Out-of-Order Epoch Completion

A_A_E_R, GATS

- All test series are nonblocking
- \( P_0 \) is origin and \( P_1 \) is target
- \( P_2 \) is target for \( P_0 \) and then origin for \( P_1 \)
- \( P_0 \) works 1000\( \mu \)s and is late for closing its access epoch.
- The graph shows the latency of \( P_1 \) and the overall latency of \( P_2 \)

\[ \text{A_A_E_R prevents } P_1 \text{ from suffering the delay of } P_0. \text{ } P_2 \text{ completes faster overall as well.} \]
Evaluation: Out-of-Order Epoch Completion

E_A_E_R, GATS

- All test series are nonblocking
- A single target opens an exposure epoch towards an origin $O_0$ and then towards another origin $O_1$
- $O_0$ is 1000µs late in closing its access epoch.
- The graph shows the latency of $P_1$ and the overall latency of $P_2$

E_A_E_R prevents $O_1$ from suffering the delay of $O_0$. The target completes faster overall as well.
A_A_E_R, GATS

- All test series are nonblocking
- $P_0$ is target and $P_1$ is origin
- $P_2$ is origin for $P_0$ and then target for $P_1$
- $P_0$ exposes its epoch 1000$\mu$s late.
- The graph shows the latency of $P_1$ and the overall latency of $P_2$

A_A_E_R prevents $P_1$ from suffering the delay of $P_0$. $P_2$ completes faster overall as well.
Evaluation: Application Pattern

Massive unstructured dynamic transactions.

- Multiple rounds of a set \{P_i\} of peers **atomically updating** another (not necessarily disjoint) set \{P_j\} of peers.
- … successions of lock\_exclusive, accumulate, unlock

A\_A\_A\_R removes contention between adjacent locks meant for disjoint targets!

A\_A\_A\_R allows 184,422 (39%), 205,377(20%) and 339,359(16%) more trs/s than the “New” series for 64, 128 and 256 CPU cores respectively.
Evaluation: Application (One-Sided|GATS LU)

Matrix of 8,192 x 8,192

Performance improvement of up to 50% with nonblocking synchronizations
Evaluation: Application (One-Sided|GATS LU)

Matrix of 16,384 x 16,384

Performance improvement of up to 50% again with nonblocking synchronizations
Conclusion

- We introduced entirely nonblocking epoch synchronizations with the following benefits:
  - All inefficiency patterns can now be avoided or mitigated
  - Flexible communication/computation overlapping
  - Communication/communication overlapping
  - Communication/delay overlapping
  - Contention avoidance
  - New use cases are made possible or performant

- Future work:
  - Enable progress engine optimization for fence and lock_all
  - Investigate the use of nonblocking one-sided epochs in rule-based distributed engines.
Thank you

Questions?