

# Nasty-MPI: Debugging Synchronization Errors in MPI-3 One-Sided Applications

Authors: Roger Kowalewski and Karl Furlinger

LMU Munich, MNM-Team

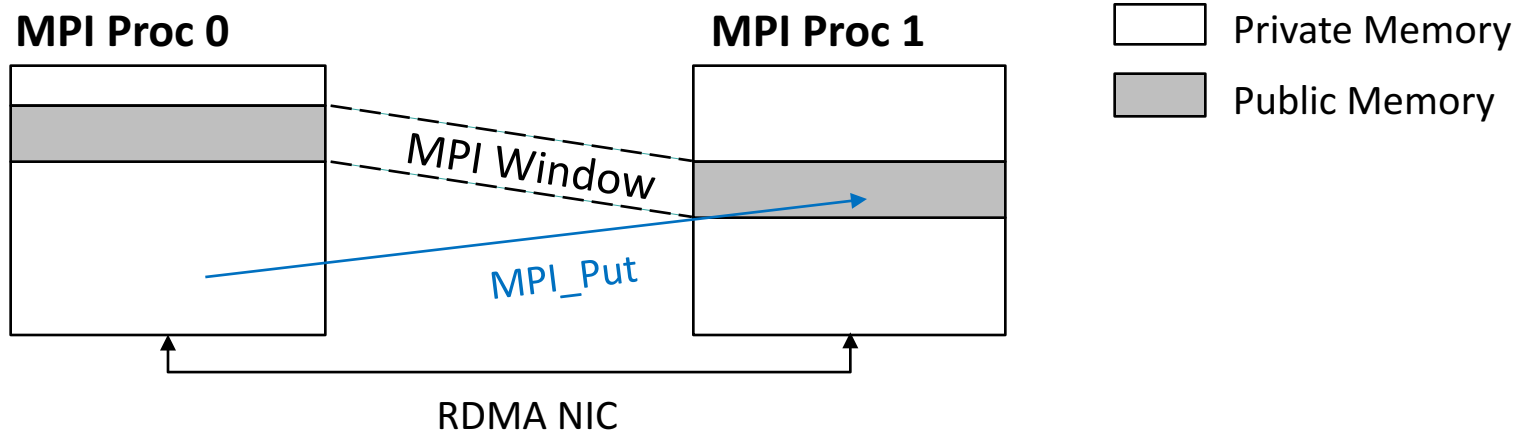
E-Mail: [kowalewski@mnm-team.org](mailto:kowalewski@mnm-team.org)

<http://mnm-team.org/~kowalewski/>



- Brief overview about MPI-3 one-sided communication
- Understanding the major challenges (Synchronization)
  - Semantic Perspective (MPI standard)
  - Technical Perspective (Behavior in the real world)
- A novel concept to debug synchronization bugs in MPI-3 one-sided communication

# MPI RMA: Conceptual Overview



- Remote Memory Access Model

- Origin specifies all required communication parameters
- Target (receiver) does not need to actively participate

- Decoupling of communication and synchronization

- Fundamental contrast to traditional Message Passing

- Very good match for PGAS (Partitioned Global Address Space)

- Examples: DASH, UPC, GA, CAF

- Communication primitives
  - Efficient Data movement: MPI\_Put (Write), MPI\_Get (Read)
  - Atomic operations (e.g. accumulate, compare and swap) → slow
- All communication primitives are in fact **non-blocking**
  - No implicit **atomicity** or **ordering** guarantees
  - Exception: RMA atomics
    - Explicit synchronization required
- Synchronization model is further split into
  - Process synchronization
  - Memory consistency → focus of this talk

## Example: Read-Modify-Write

```

1 //Window creation
2 MPI_Win win;
3 //1. start access epoch (non-blocking)
4 Win_lock(remote_proc, win);
5 int out;
6 //2. RMA Read (non-blocking)
7 Get(&out, remote_proc, remote_disp);
8 //load / store access
9 if (out % 2 == 0)
10     out++;
11 //3. end access epoch (blocking)
12 Win_unlock(remote_proc, win);

```

Synchronization Bug!

Blocks until the  
MPI\_Get is completed

- Application may or may not execute with the expected outcome
  - MPI implementation
  - Hardware Platform, Network Interconnect
  - Scheduling algorithms, etc.
- MPI-3 Standard: Undefined behavior

Definition: A program forms a **well-defined execution** if all memory accesses are **data-race free**. For an execution to be free of data-races all memory accesses must be **synchronized** by: <sup>1</sup>

- a) Happens-before order, i.e.  $a \xrightarrow{hb} b$ 
    - Program order (single MPI process)
    - Synchronization order among a group of MPI processes
  - b) Consistency order, i.e.  $a \xrightarrow{co} b$ 
    - Remote completion
      - Win\_unlock (end of access epoch)
      - Win\_flush (during access epoch)
    - Local completion
      - Win\_flush\_local (during access epoch)
- Abbreviation in further slides:  $a \xrightarrow{cohb} b$

<sup>1</sup> Hoefler et al. Remote Memory Access Programming in MPI-3. ACM Trans. (Jun 2015)

# Example: Ordering guarantees

## Origin

```

1 Win_lock(B, win);
2 //1. write data
3 int data = 42;
4 Put(&data, B, disp_data);
5
6 //2. set guard
7 int guard = 1;
8 Put(&guard, B, disp_guard);
9
10 Win_unlock(B, win);

```

## Target

```

1 Win_lock(B, win);
2
3 while (win[disp_guard] == 0)
4 {
5     //wait for guard
6 }
7 //read data
8 assert(win[disp_data] == 42);
9
10 Win_unlock(B, win);

```

cohb

- *Origin* modifies a remote memory location at *target* to a particular value (data = 42)
- *Target* verifies this value, i.e., `assert(data == 42)`
- *Guard* establishes a synchronization order between origin and target

# Example: Ordering guarantees

## Origin

```

1 Win_lock(B, win);
2 //1. write data
3 int data = 42;
4 Put(&data, B, disp_data);
5 Win_flush(B, win); //Sync
6 //2. set guard
7 int guard = 1;
8 Put(&guard, B, disp_guard);
9
10 Win_unlock(B, win);

```

cohb

## Target

```

1 Win_lock(B, win);
2
3 while (win[disp_guard] == 0)
4 {
5     //wait for guard
6 }
7 //read data
8 assert(win[disp_data] == 42);
9
10 Win_unlock(B, win);

```

cohb

- *Origin* modifies a remote memory location at *target* to a particular value (data = 42)
- *Target* verifies this value, i.e., `assert(data == 42)`
- *Guard* establishes a synchronization order between origin and target



## Origin

```

1 Win_lock(B, win);
2 //1. write data
3 int data = 42;
4 Put(&data, B, disp_data);
5
6 //2. set guard
7 int guard = 1;
8 Put(&guard, B, disp_guard);
9
10 Win_unlock(B, win);

```

## Target

```

1 Win_lock(B, win);
2
3 while (win[disp_guard] == 0)
4 {
5     //wait for guard
6 }
7 //read data
8 assert(win[disp_data] == 42);
9
10 Win_unlock(B, win);

```

- Small Experiment on 2 HPC systems

- 100x repeatedly executed
- origin and target randomly chosen (out of 48 MPI processes, 2 nodes)
- NERSC Edison: Cray MPT, Aries Network Interconnect
- SuperMUC: IBM Platform, non-blocking IB

## Origin

```

1 Win_lock(B, win);
2 //1. write data
3 int data = 42;
4 Put(&data, B, disp_data);
5
6 //2. set guard
7 int guard = 1;
8 Put(&guard, B, disp_guard);
9
10 Win_unlock(B, win);

```

## Target

```

1 Win_lock(B, win);
2
3 while (win[disp_guard] == 0)
4 {
5     //wait for guard
6 }
7 //read data
8 assert(win[disp_data] == 42);
9
10 Win_unlock(B, win);

```

## ■ Test results

- NERSC Edison (Cray): Passes 100%
- SuperMUC (IBM): Succeeds only if both processes run on the same node
  - Utilizes shared memory semantics

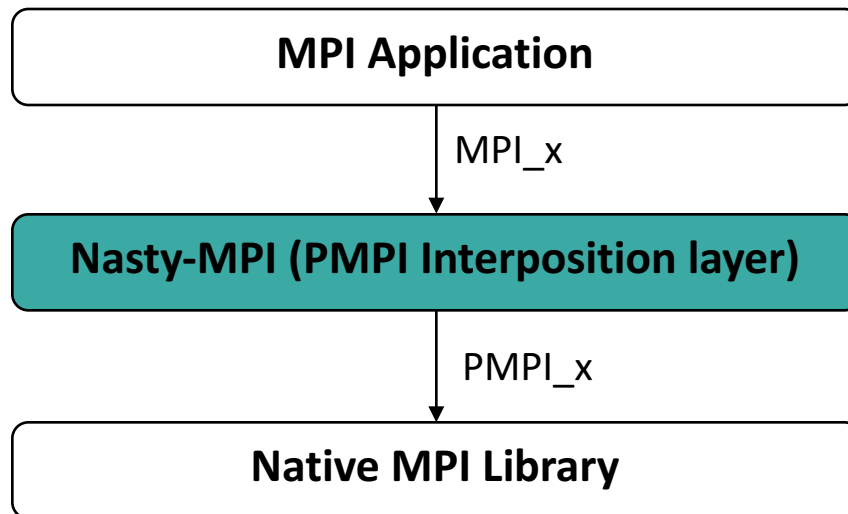
- Cray MPT uses DMAPP for communication which offers parametric in-order guarantees (IB Verbs do not)

- MPI RMA has a complex synchronization model
  - Couple of semantic pitfalls
- Manifestation of synchronization bugs depends on various factors
  - May often only happen in large-scale scenarios
  - Environment (MPI library, Network conditions)
- Programmers must understand the synchronization model to guarantee **well-defined** and **portable** programs

- Strategies to prevent synchronization bugs
  - Unit Testing
  - Consistency checks in the source code (e.g., assertions)
  - Verbose Mode
- Limited functionality and not appropriate for large code bases
  - Example: DASH library
- Additional Tools are imperative

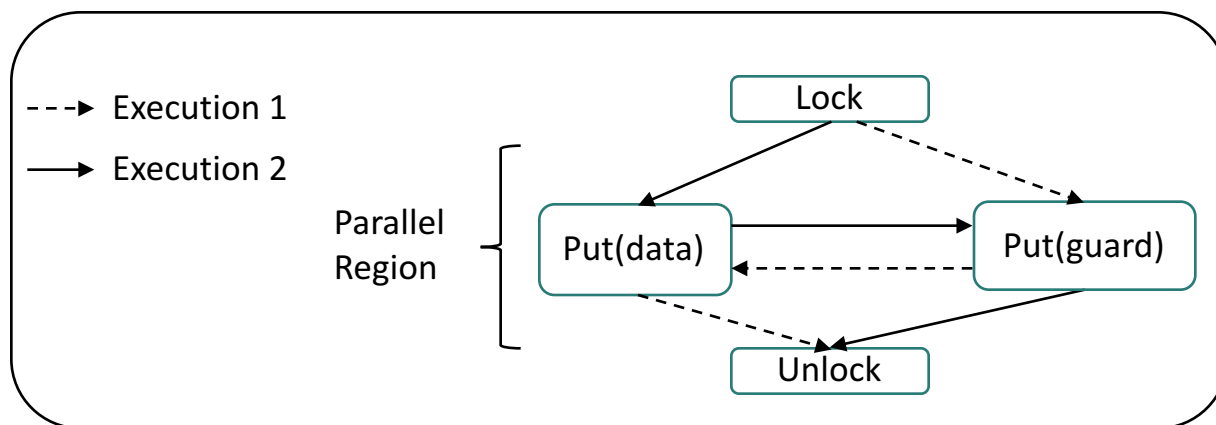
- MPI Spin: Model Checking (Pervez et al., 2006)
  - Formal model of application required
  - Model State Explosion Possible
  - Covers only MPI-2 Standard
- Wait States (Herrmanns et al., 2013)
  - Does not focus on memory consistency
- Marmot / MUST (Krammer et al., 2006)
  - Does only focus on static parameter checking
- MC Checker (Chen et al., 2014)
  - Closely related to this paper
  - Static and dynamic analysis techniques
  - Covers only MPI-2 specification

- Approach: Emulate a nasty MPI-3 implementation
  - Exploit full flexibility of RMA semantics to force **pessimistic executions**
  - Requires deterministic (pre-defined) outcome of target application
- Can be easily linked into any MPI application
  - Requires **no** code modification (based on PMPI interface)
  - May complement with other tools



Given: Target application which issues MPI RMA calls

1. Intercept and Buffer all MPI RMA communication calls
  - No call reaches the MPI library
2. Dynamically construct a DAG among set of buffered RMA calls
  - Based on transitive closure of  $\xrightarrow{hb}$  and  $\xrightarrow{co}$
3. Identify parallel regions to obtain the set of possible executions



Triggered by a **blocking** RMA synchronization action

## 1. Completion

- Distinguish local and remote completion
- Only calls which have to be **remotely** completed will reach the library

## 2. Atomicity

- Break non-atomic operations into a set of smaller (single-byte) messages
- Atomicity guarantees of *RMA Accumulates* are considered

## 3. Reorder all RMA operations

- Ordering guarantees of *RMA Accumulates* are considered

## 4. Issue MPI RMA calls to native MPI library

➤ Interventions always result in identical memory semantics



## Refinement of Nasty-MPI scheduling process

Category	Parameter	Option	Description
Ordering	NASTY_SUBMIT_ORDER	Random *	Random order
		reverse_po	Reverse program order
		put_before_get	Issue Puts before Gets
		get_before_put	Issue Gets before Puts
Completion	NASTY_COMPLETION	0,1*	Enable / disable flag
	NASTY_LOCAL_COMPLETION	0,1*	Nasty local compl.
	NASTY_FLUSH	0,1*	Intervening flushes
Atomicity	NASTY_ATOMICITY	0,1*	Enable / disable flag

\* Default value

- Approach: Try different parameter settings on the same target application

## ■ SuperMUC

- Interconnect: Non-blocking Infiniband
- MPI Libraries
  - IBM MPI v9.1.4
  - Intel MPI v5.0
  - Open MPI v1.8

## ■ NERSC Edison

- Interconnect: Cray Aries
- MPI Library: Cray MPT

## ■ Compiler: icc 15.04

- Small test applications
  - Based on algorithms and papers for one-sided communication
  - Pre-defined deterministic (expected) outcome
  - **Injected latent synchronization bugs**
- Assumption
  - Test applications terminate with the expected outcome
- Applying Nasty-MPI manifests the latent synchronization bugs

# Test Case 1: Revisiting the Example

## MPI Proc 0 (Origin)

```

1 Win_lock(B, win);
2 //1. write data
3 int data = 42;
4 Put(&data, B, disp_data);
5
6 //2. set guard
7 int guard = 1;
8 Put(&guard, B, disp_guard);
9
10 Win_unlock(B, win);

```

## MPI Proc 1

```

1 Win_lock(B, win);
2
3 while (win[disp_guard] == 0)
4 {
5     MPI_Iprobe(0, MPI_ANY_TAG);
6 }
7 //read data
8 assert(win[disp_data] == 42);
9
10 Win_unlock(B, win);

```

Table: Test results without linking Nasty-MPI

MPI library	Expected Outcome?	Comment
Cray MPI	✓	
IBM MPI	(✓)	Origin and Target reside on same node
Open MPI	(✓)	Origin and Target reside on same node
Intel MPI	✗	Proc1 sticks in busy wait


# Test Case 1: Applying Nasty-MPI

## Origin (Nasty-MPI)

```

1 Win_lock(B, win);
2 int data = 42, guard = 1;
3 //1. set guard
4 Put(&guard, B, disp_guard);
5
6 Win_flush(B, win); //Sync
7
8 //2. write data
9 Put(&data, B, disp_data);
10 Win_unlock(B, win);

```



## Target

```

1 Win_lock(B, win);
2
3 while (win[disp_guard] == 0)
4 {
5     //wait for guard
6 }
7 //read data
8 assert(win[disp_data] == 42);
9
10 Win_unlock(B, win);

```

## ■ Environment Settings

- export NASTY\_COMPLETION = 1 (default)
- export NASTY\_FLUSH = 1; (default)
- export NASTY\_SUBMIT\_ORDER = reverse\_po

MPI library	Expected Outcome?	Comment
Cray MPI	✗	
IBM MPI	✗	Origin and Target reside on same node
Open MPI	✗	Origin and Target reside on same node
Intel MPI	✗	Proc1 stuck in busy wait

## Put Buffer Out of scope

```

1 int main(...)
2 {
3   Win_lock(target, win);
4   //call kernel with computation
5   kernel();
6   Win_unlock(target, win);
7 }
8
9 void kernel(...)
10 {
11   //Complex Computation
12   int val = 1;
13   MPI_Put(&val, target, ...);
14 }

```

## Evaluation Results

MPI library	Expected Outcome?	Nasty-MPI
Cray MPI	✓	✗
IBM MPI	✗	✗
Open MPI	✓	✗
Intel MPI	✓	✗

- This kind of synchronization bug was originally found in an algorithm in the DASH library (unit tests could not flag this error!)
- Linking Nasty-MPI forced a manifestation

## ■ Additional selected Use Cases

Table: Test Results without Nasty-MPI

#	Application	Bug	Cray MPI	IBM MPI	Intel MPI	Open MPI	Nasty MPI
3	Binary Broadcast <sup>1</sup>	Missing flush	(✓)	✗	(✓)	✗	✗
4	MCS lock <sup>2</sup>	Replaced Win_flush by Win_flush_local	✓	✓	✓	✓	✗

- ✓ PASSED with expected outcome (✓) Passed only, if origin and target resided on same node
- ✗ FAILED

<sup>1</sup> Luecke, G.R., Spanoyannis, S., Kraeva, M.: The Performance and Scalability of SHMEM and MPI-2 One-sided Routines on a SGI Origin 2000 and a Cray T3E-600: Performances. *Concurr. Comput.* (Aug 2004)

<sup>2</sup> Mellor-Crummey, J.M., Scott, M.L.: Algorithms for Scalable Synchronization on Shared-memory Multiprocessors. *ACM Trans. Comput. Syst.* 9(1), 21–65 (Feb 1991)

- MPI-3 RMA synchronization model is complex
  - Atomicity, ordering, completion
- Execution of RMA calls differs
  - Among MPI libraries
  - Underlying hardware platform
- Introduced a novel debugging approach
  - Forces manifestation of latent synchronization bugs
  - Support to write well-defined programs
  - Complements with existing debugging tools (e.g., Model Checking)
- Future work
  - Evaluate Nasty-MPI on real-world scientific applications which MPI RMA
  - Improve Nasty-MPI heuristics (Uncover synchronization bugs is difficult)



- Nasty-MPI Source: <https://github.com/rkowalewski/nasty-MPI>
- DASH Project: <http://www.dash-project.org/>
- Contact
  - Roger Kowalewski
  - LMU Munich, Germany
  - E-Mail: [kowalewski@mnm-team.org](mailto:kowalewski@mnm-team.org)