MC-Checker: Detecting Memory Consistency Errors in MPI One-Sided Applications

Zhezhe Chen[1], James Dinan[2], Zhen Tang[3], Pavan Balaji[4], Hua Zhong[3], Jun Wei[3], Tao Huang[3], and Feng Qin[5]

1. Twitter Inc.
2. Intel Corporation
3. Chinese Academic of Sciences
4. Argonne National Laboratory
5. The Ohio State University

MPI One-Sided Communication

- Remote Memory Access (RMA) extends MPI with one-sided communication
  - Allows one process to specify both sender and receiver communication parameters
  - Facilitates the coding of partitioned global address space (PGAS) data models
- Dinan et al. [1] ported the Global Arrays runtime system, ARMCI to MPI RMA
  - NWChem is a user of MPI RMA, which we use to evaluate our tool
- We focus on MPI-2 RMA, which is compatible with MPI-3 (future work)

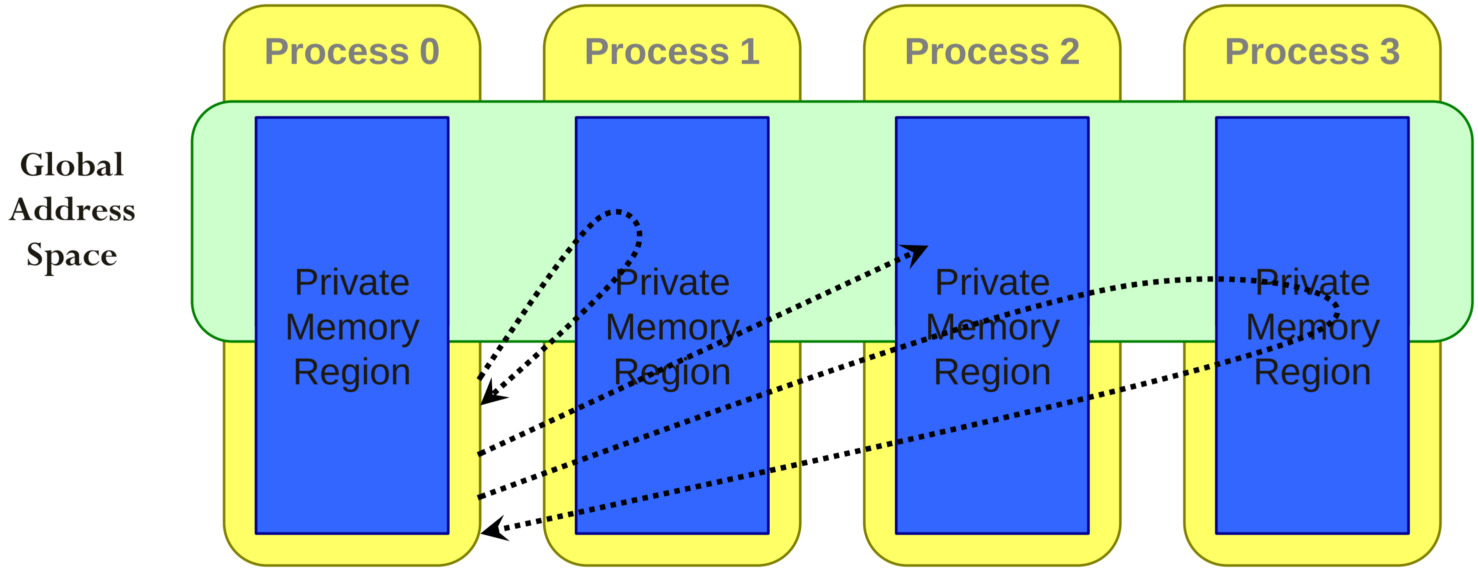
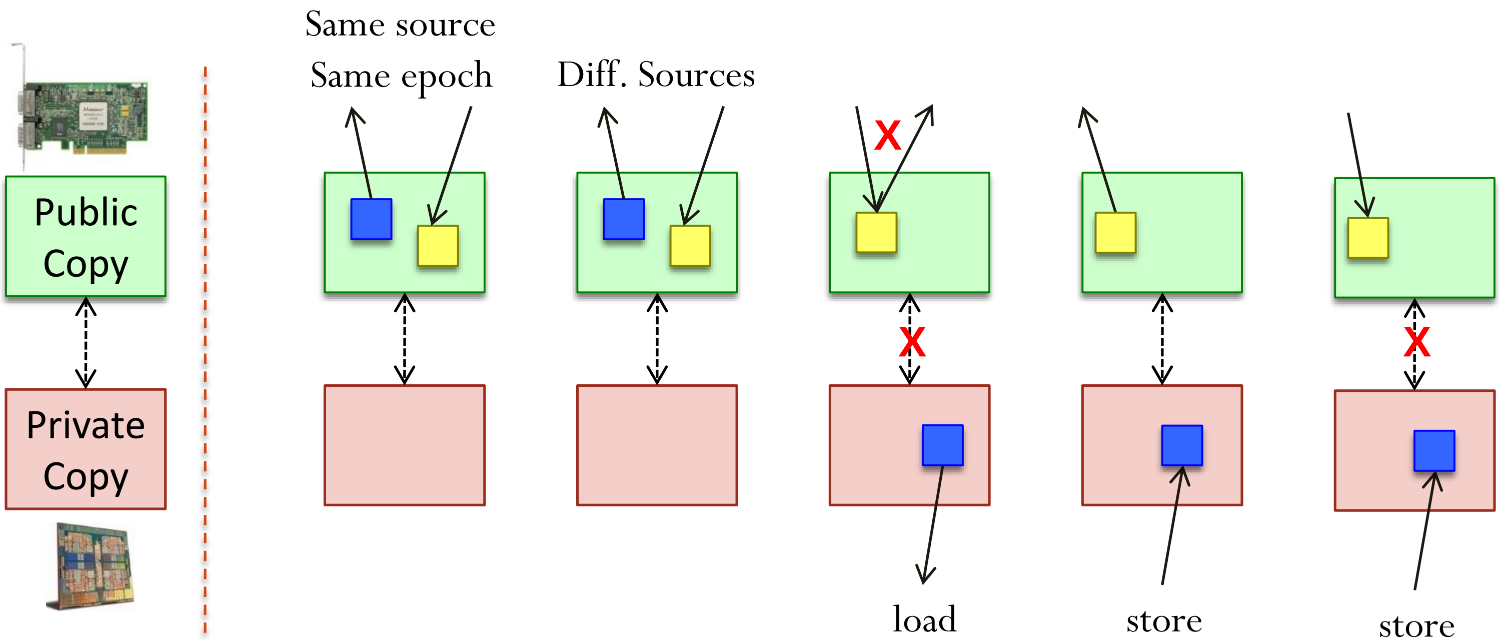


Figure credit: *Advanced MPI Tutorial*, P. Balaji, J. Dinan, T. Hoefler, R. Thakur, SC '13

[1] *Supporting the Global Arrays PGAS Model Using MPI One-Sided Communication*, J. Dinan, P. Balaji, S. Krishnamoorthy, V. Tipparaju. IPDPS 2012

MPI RMA Challenges



- To ensure portable, well-defined behavior, programs must follow the rules:
    1. Operations must be synchronized using, e.g., lock/unlock or fence
    2. Communication operations are nonblocking
        - Local buffers cannot be accessed until put/get/accumulate are completed
    3. Concurrent, conflicting operations are erroneous
    4. Local load/store updates conflict with remote accesses
- The MPI-2 model is referred to as the "separate" memory model in MPI-3
    - The MPI-3 "unified" model relaxes some rules, so we are solving the harder problem

A Bug Example Within an Epoch

```
MPI_Win_lock(MPI_LOCK_EXCLUSIVE, 0, 0, win);
MPI_Get(&out, 1, MPI_INT, 0, 0, 1, MPI_INT, win);
if(out % 2 == 0) /* bug: load/store access of out */
    out++;
…
MPI_Win_unlock(0, win);
```

A Bug Example Across Processes

P0 (Origin Process)	P1 (Target Process) window location X	P2 (Origin Process)
MPI_Barrier	MPI_Barrier	MPI_Barrier
MPI_Win_lock (SHARED, P1)	…	MPI_Win_lock (SHARED, P1)
…	…	…
MPI_Put(X)	…	MPI_Put(X)
…	…	…
MPI_Win_unlock(P1)	…	MPI_Win_unlock(P1)
MPI_Barrier	MPI_Barrier	MPI_Barrier

Previous Works

- Bug detection for MPI one-sided programs
  - e.g., Marmot, [Pervez-EuroPVM/MPI'06], and Scalasca
  - Detect parameter errors, deadlocks, and performance bottlenecks
- Shared-memory data race detection
  - e.g., Locksmith, Pacer, Eraser, and Racetrack
  - Detect data races for shared-memory programs
  - Fine-grain analysis is not feasible for analysis of MPI programs
- Need new techniques for one-sided communication bug detection in one-sided communication models

MC-Checker Highlights

- MC-Checker is a new tool to detect memory consistency errors in MPI one-sided applications
  - First comprehensive approach to address memory consistency errors in MPI one-sided communication
  - Incur relatively low overhead (45.2% on average)
  - Require no modification of source code

- Data access DAG analysis technique
  - Applicable to variety of one-sided communication models
  - Identifies bugs based on concurrency of accesses
    - Finds errors that did happen and *could have* happened

Outline

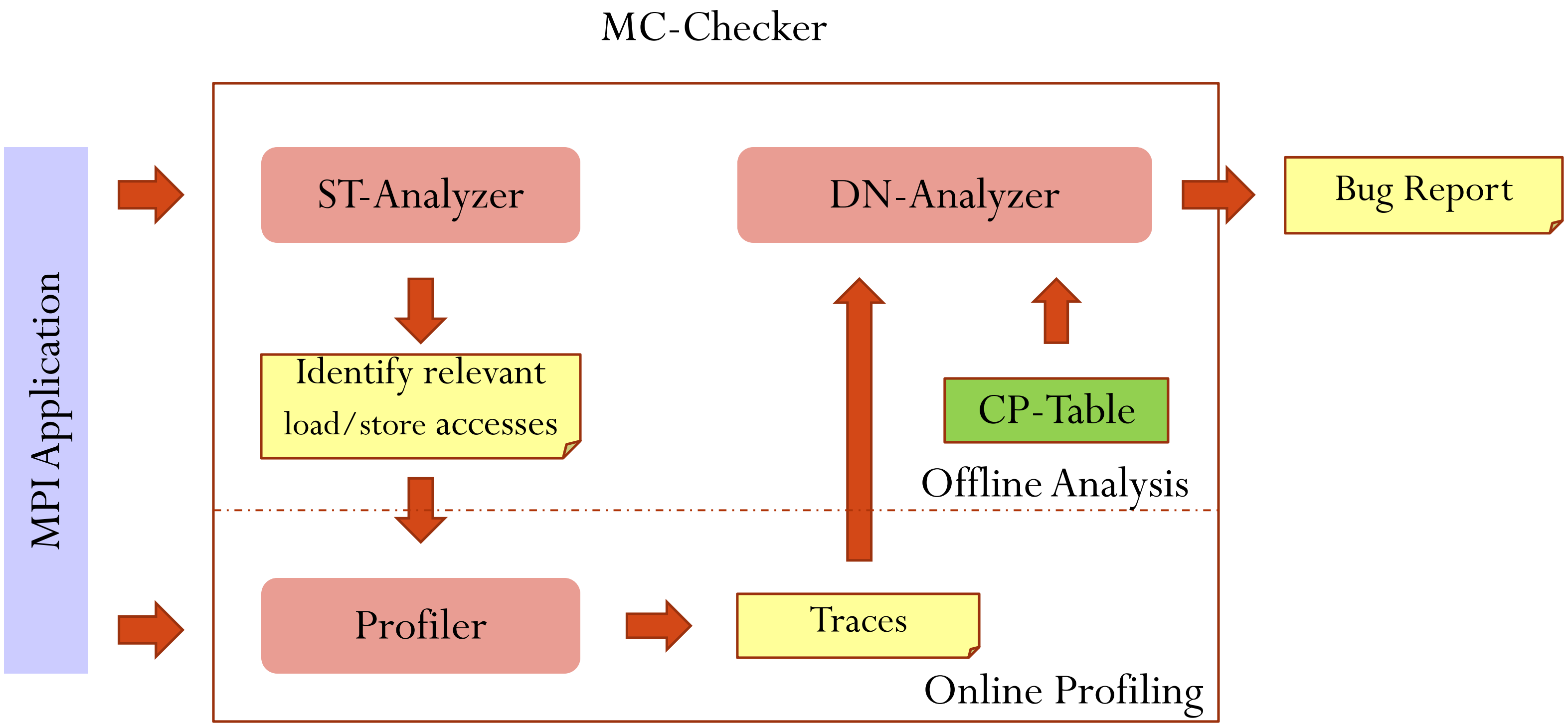
MC-Checker Main Idea

- Check the one-sided operations and local memory accesses and then check against compatibility tables to see whether there are memory consistency errors.

- Check bugs within an epoch:
  - Identify epoch region
  - Check operations within an epoch against compatibility table

- Check bugs across processes:
  - Identify concurrent regions by matching synchronization calls
  - Check operations in the concurrent regions against compatibility table

Design of MC-Checker

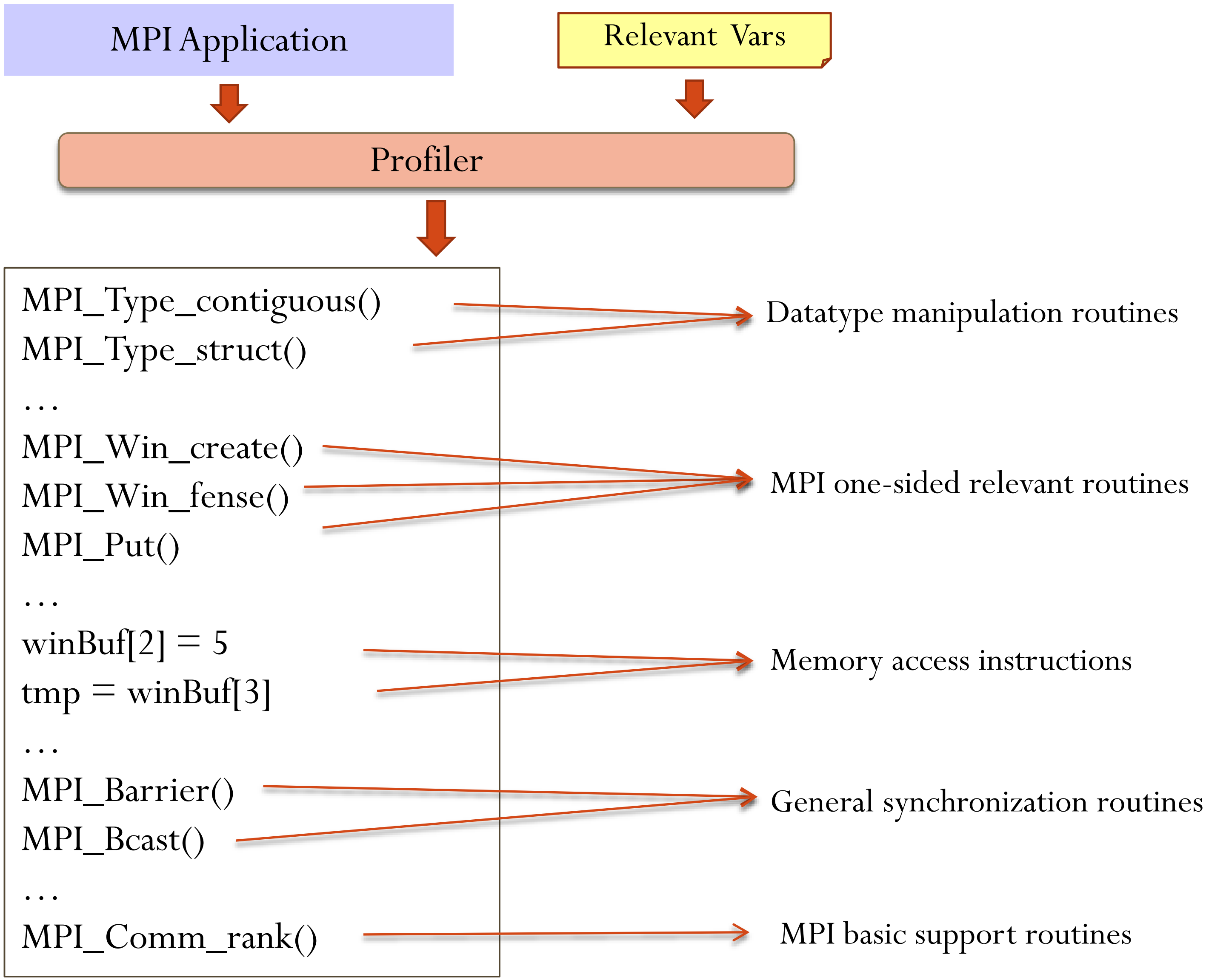


ST-Analyzer: Identify Relevant Memory Accesses

- Profiling each memory load/store is very heavy-weight

- Perform static analysis to identify relevant memory accesses
  - Mark the variables and pointers belong to the window buffers and the buffers accessed by one-sided operations
  - Propagate the markers by using pointer alias analysis
  - Propagate the markers by following function calls involving pointers and references

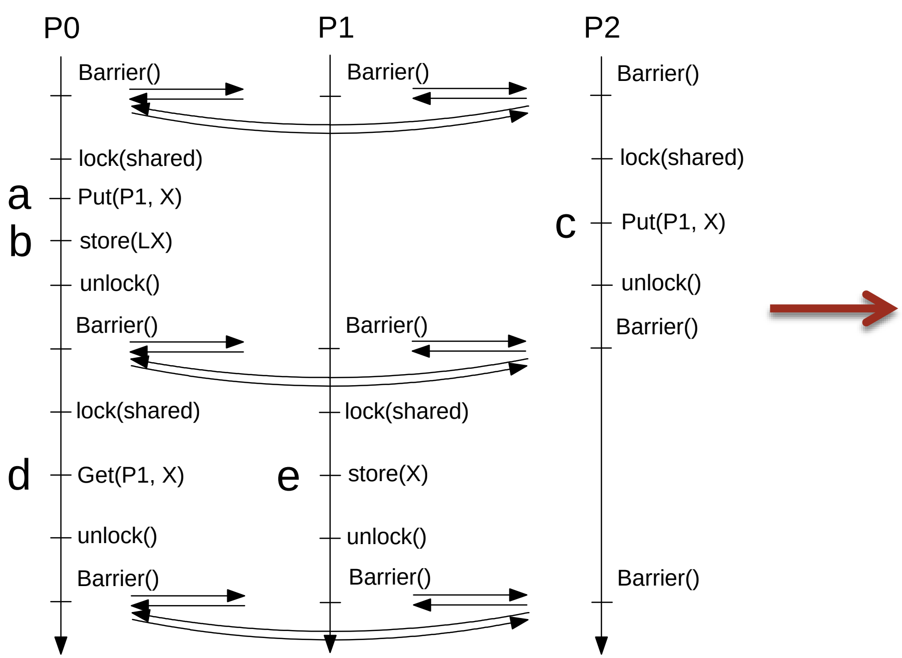
Profiler: Profiling Runtime Events

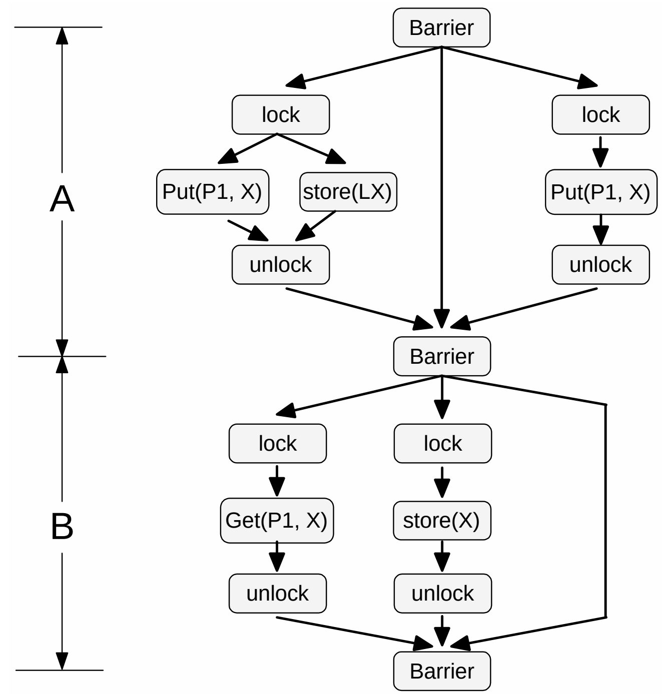


DN-Analyzer: Memory Consistency

- Memory consistency errors occur when conflicting operations are potentially concurrent during program execution
  - Conflicting operations: e.g. overlapping MPI_Put and MPI_Put
  - Happen concurrently: operations are not ordered
    - a $\xrightarrow{hb}$ b means a happens before b
      - Ordered by barrier, send/recv, etc.
    - a $\xrightarrow{co}$ b means the memory effects of a are visible before b
      - Memory updates are synchronized by unlock, fence, etc.

DN-Analyzer: DAG Analysis Technique





- Capture dynamic execution and convert to data access DAG
  - Edges capture ordering and concurrency of access
- Identifies logical concurrency – bugs that happened and *could have* happened
- General analysis technique for one-sided and PGAS models

DN-Analyzer: Within an Epoch

2nd \ 1st	Load	Store	Get	Put/Acc
Load	BOTH	BOTH	NOVL	BOTH
Store	BOTH	BOTH	NOVL	NOVL
Get	BOTH	BOTH	NOVL	NOVL
Put/Acc	BOTH	BOTH	NOVL	BOTH

Epoch Region

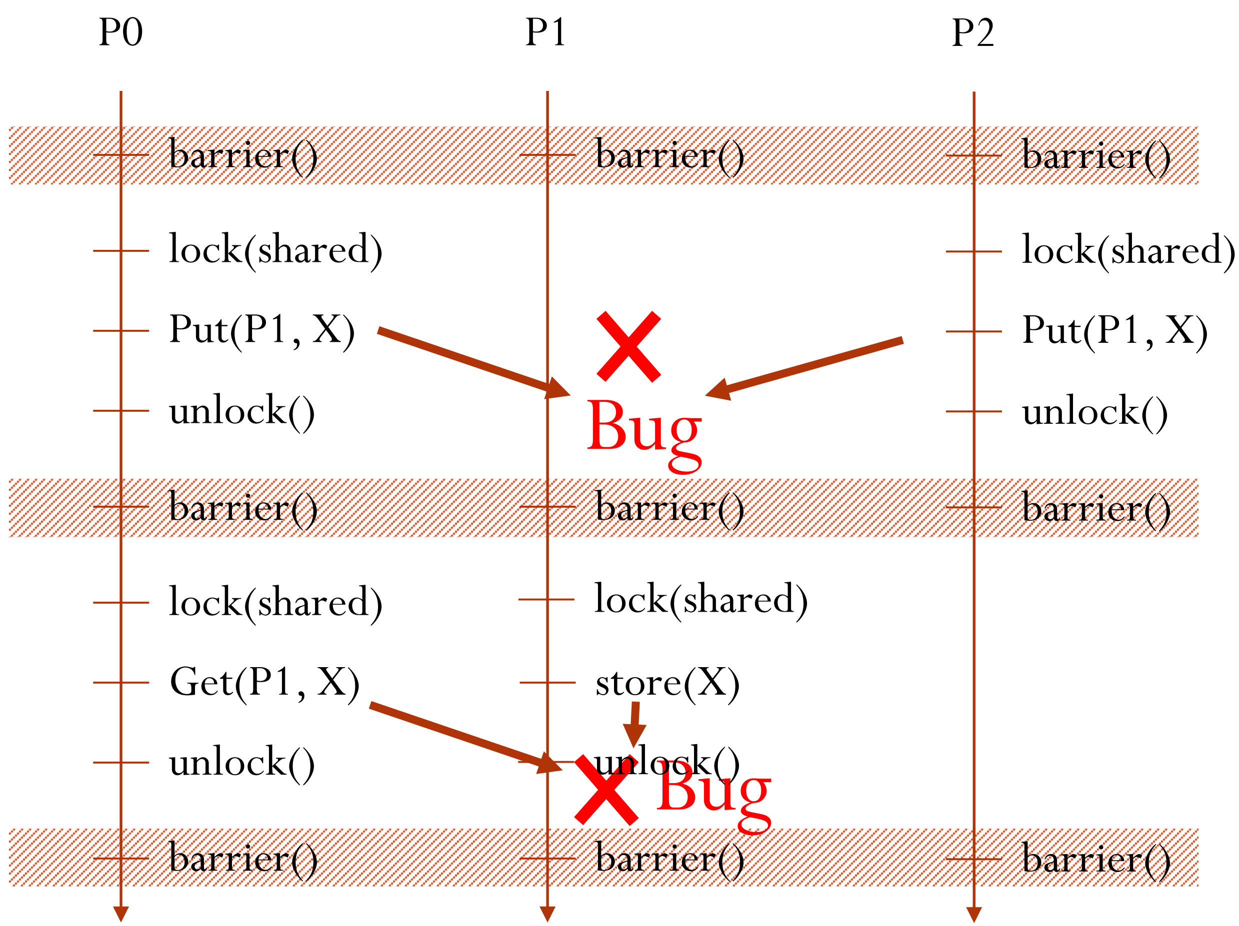
1. MPI_Win_lock(MPI_LOCK_EXCLUSIVE, 0, 0, win);
2. MPI_Get(&out, 1, MPI_INT, 0, 0, 1, MPI_INT, win);
3. if(out % 2 == 0) ✗ Bug (overlapping)
4. out++; ✗ Bug (overlapping)
5. …
6. MPI_Win_unlock(0, win);

DN-Analyzer: Across Processes

	Load	Store	Get	Put	Acc
Load	BOTH	BOTH	BOTH	NOVL	NOVL
Store	BOTH	BOTH	NOVL	X	X
Get	BOTH	NOVL	BOTH	NOVL	NOVL
Put	NOVL	X	NOVL	NOVL	NOVL
Acc	NOVL	X	NOVL	NOVL	BOTH

- Compatibility matrix of RMA operations
  - BOTH: overlapping and nonoverlapping combinations of the given operations are permitted
  - NOVL: only non-overlapping combinations are permitted
  - X: combination is erroneous.

DN-Analyzer: Across Processes



Match synchronization calls

Outline

- 1. Motivation
- 2. Bug Examples
- 3. Main Idea
- 4. Design and Implementation
- 5. Evaluation
- 6. Conclusion

Evaluation Methodology

- Hardware
  - Glenn cluster at Ohio Supercomputer Center
  - 658 computer nodes
  - 2.5 GHz Opterons quad-core CPU each node
  - 24 GB RAM, 393 GB local disk each node
- Software
  - Compiler: Modified LLVM to annotate load/store ops of interest
  - OS: Linux 2.6.18
  - MPI Library: MPICH2
- Evaluation
  - Effectiveness: 3 real-world and 2 injected bug cases
  - Overhead: 5 benchmarks

Bug Cases

MPI Applications	Bug IDs	Bug Locations	Mode
emulate	04/2011	within an epoch	passive
BT-broadcast	06/2004	within an epoch	active
lockopts	r10308	across processes	passive
pingpong-inj	3.0.3	across processes	passive
jacobi-inj	09/2008	across processes	active

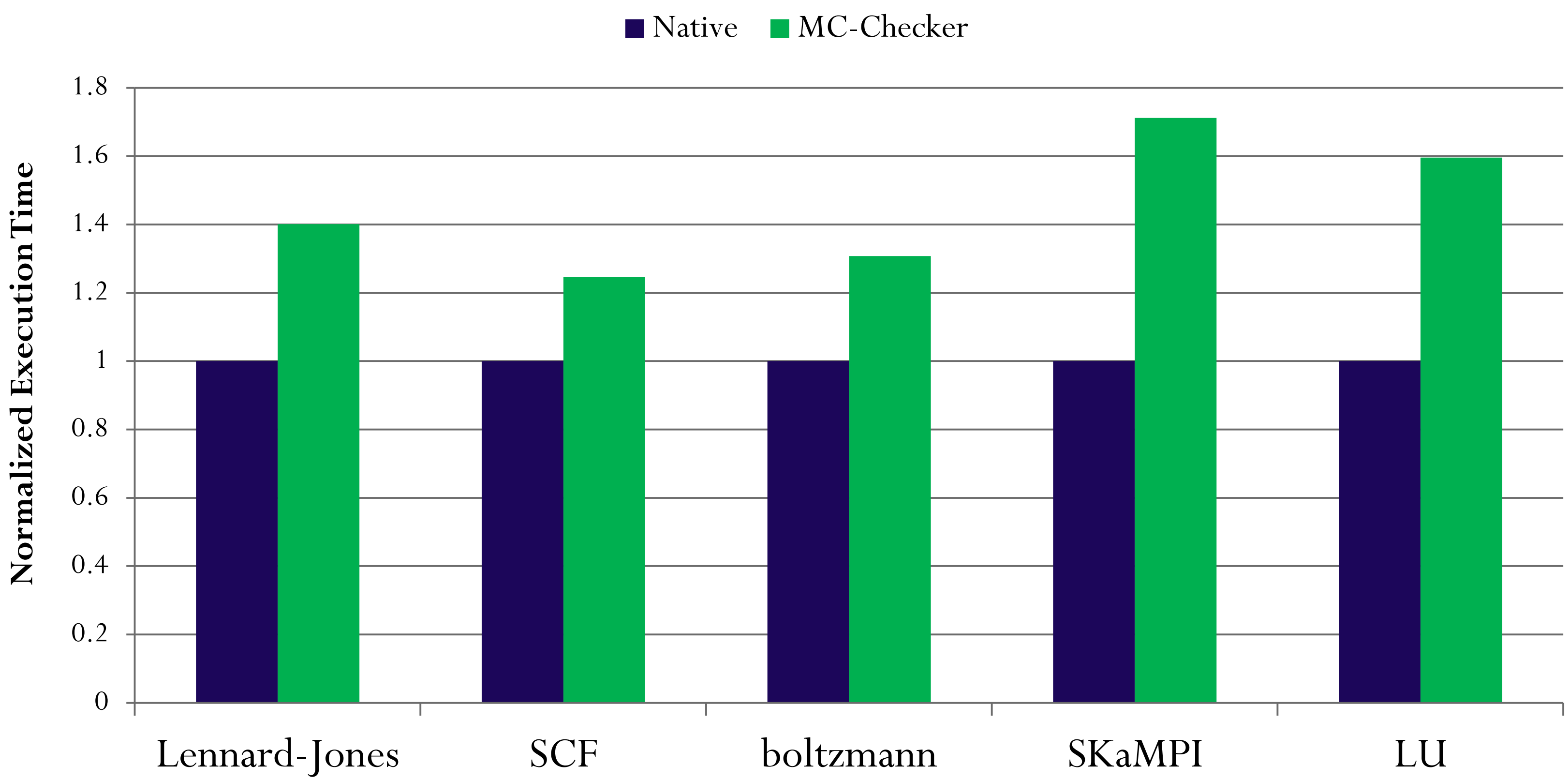
- 3 real-world and 2 injected bug cases from 5 MPI applications

Effectiveness

MPI Apps	Bug IDs	Detected?	Pinpoint Root Cause?	Error Locations	Conflicting Operations	Failure Symptoms	# of Processes
emulate	04/2011	Yes	Yes	within an epoch	get and load/store	incorrect result	2
BT-broadcast	06/2004	Yes	Yes	within an epoch	get and load	program hang	2
lockopts	r10308	Yes	Yes	across processes	put/get and load/store	incorrect result	64
pingpong-inj	3.0.3	Yes	Yes	across processes	put and put	incorrect result	64
jacobi-inj	09/2008	Yes	Yes	across processes	put and get	incorrect result	64

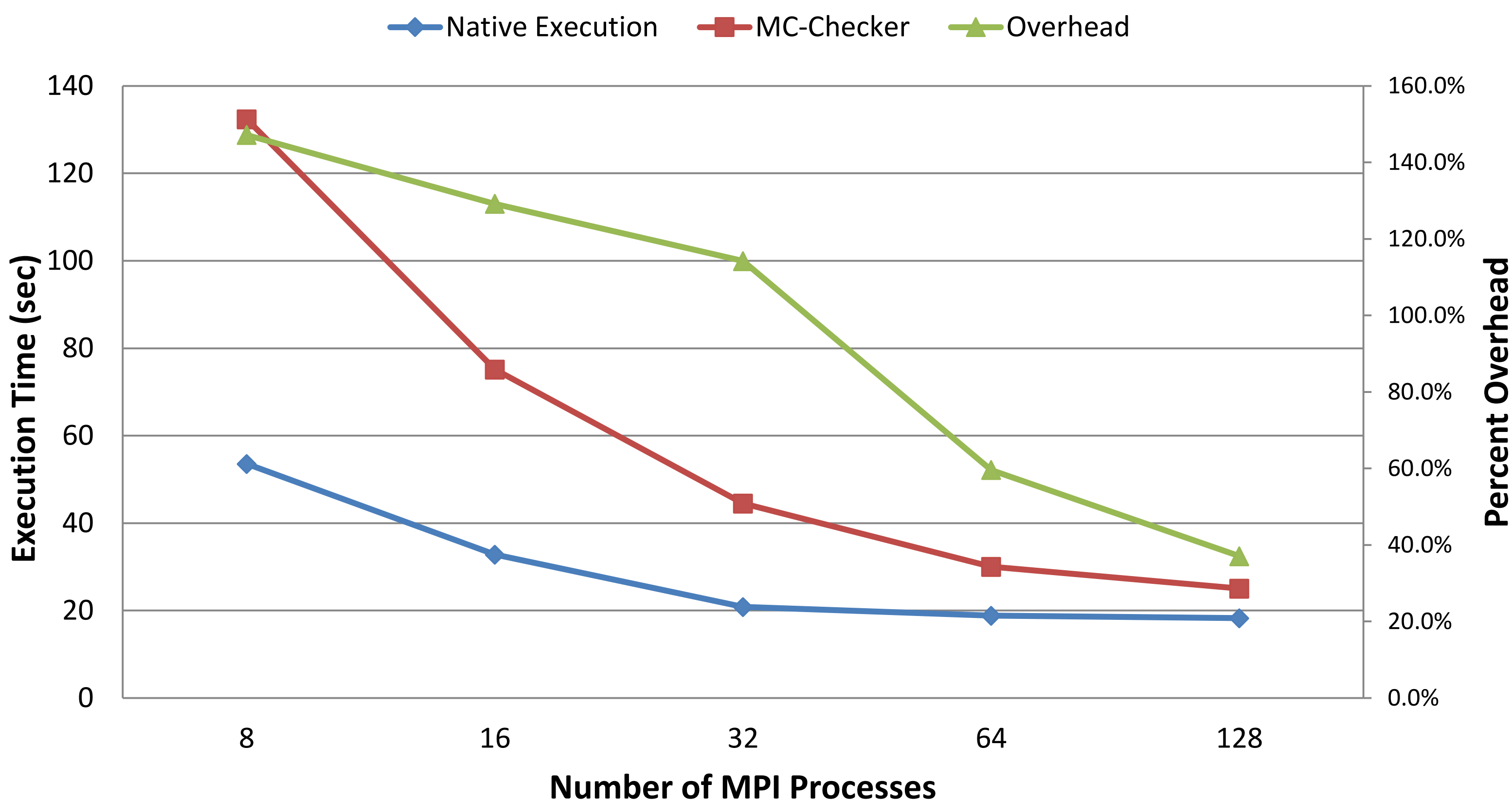
- Detect and locate root cause for all of the 5 bug cases

Runtime Overhead



- Runtime overhead is low, ranging from 24.6% to 71.1%, with an average of 45.2%

Scalability of Overheads



- The runtime overhead decreases from 147.2% to 37.1% when the number of processes increase from 8 to 128

Conclusion

- MC-Checker
  - Detects memory consistency errors in MPI one-sided apps
  - Detect and locate the root causes of the bugs
  - Incur low runtime overhead
- Happens-before analysis identifies concurrency bugs
- Tools to enable debugging of one-sided applications are important in enabling users to overcome complexity

Thanks!