Scalable Tools for Debugging Non-Deterministic MPI Applications

ReMPI: MPI Record-and-Replay tool

Kento Sato, Dong H. Ahn, Ignacio Laguna, Gregory L. Lee, Martin Schulz and Chris Chambreau
Debugging large-scale applications is already challenging

“On average, software developers spend 50% of their programming time finding and fixing bugs.”[1]


With trends towards asynchronous communication patterns in MPI applications, **MPI non-determinism** will significantly increase debugging cost
What is MPI non-determinism?

- Message receive orders can be different across executions
  - Unpredictable system noise (e.g. network, system daemon & OS jitter)
- Floating point arithmetic orders can also change across executions

Execution A: \((a+b)+c\)
Execution B: \(a+(b+c)\)
Non-determinism also increases debugging cost

- Control flows of an application can change across different runs

- Non-deterministic control flow
  - Successful run, seg-fault or hang

- Non-deterministic numerical results
  - Floating-point arithmetic is non-associative

\[(a+b)+c \neq a+(b+c)\]

➡ Developers need to do debug runs until the target bug manifests

In non-deterministic applications, it’s hard to reproduce bugs and incorrect results. It costs excessive amounts of time for “reproducing” target bugs.
Non-deterministic bugs
--- Case study: Pf3d and Diablo/Hypre 2.10.1

- Debugging non-deterministic hangs often cost computational scientists substantial time and efforts

- **Diablo** - hung only once every 30 runs after a few hours

- **Pf3d** – hung only when scaling to half a million MPI processes

- The scientists spent 2 months in the period of 18 months and gave up debugging it

  Hypre is an MPI-based library for solving large, sparse linear systems of equations on massively parallel computers

- The scientists refused to debug for 6 months ...
Non-deterministic numerical result

--- Case study: “Monte Carlo Simulation” (MCB)

- CORAL proxy application
- MPI non-determinism

**Table 1: Catalyst Specification**

<table>
<thead>
<tr>
<th>Nodes</th>
<th>304 batch nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2.4 GHz Intel Xeon E5-2695 v2 (24 cores in total)</td>
</tr>
<tr>
<td>Memory</td>
<td>128 GB</td>
</tr>
<tr>
<td>Interconnect</td>
<td>InfiniBand QDR (QLogic)</td>
</tr>
<tr>
<td>Local Storage</td>
<td>Intel SSD 910 Series (PCIe 2.0, MLC)</td>
</tr>
</tbody>
</table>

Final numerical results are different between 1st and 2nd run

```
$ diff result_run1.out result_run2.out
result_run1.out:< IMC E_RR_total =-3.3140234409e-05 -8.302693774e-08 2.915332360e-08 -4.8198506756e-06 2.3113821822e-06
result_run2.out:> IMC E_RR_total =-3.3140234410e-05 -8.302693766e-08 2.915332360e-08 -4.819850657e-06 2.311382121e-06
```

* The source was modified by the scientist to demonstrate the issue in the field
Why MPI non-determinism occurs?

- It’s typically due to communication with MPI_ANY_SOURCE
- In non-deterministic applications, each process doesn’t know which rank will send message
- Messages can arrive in any order from neighbors ➔ inconsistent message arrivals

MPI_ANY_SOURCE communication

```c
MPI_Irecv(..., MPI_ANY_SOURCE, ...);
while(1) {
    MPI_Test(flag);
    if (flag) {
        <computation>
        MPI_Irecv(..., MPI_ANY_SOURCE, ...);
    }
}
```

Communications with neighbors

MCB: Monte Carlo Benchmark
ReMPI can reproduce message matching

- ReMPI can reproduce message matching by using record-and-replay technique

- Traces, records message receive orders in a run, and replays the orders in successive runs for debugging
  - Record-and-replay can reproduce a target control flow
  - Developers can focus on debugging a particular control flow

Diagram:
- Rank 0, 1, 2, 3
- Seg-fault
- Developer can focus on debugging particular control flow
- Input
- Output
- Output A, B
- Hanging
Record overhead to performance

- Performance metric: how many particles are tracked per second

![Performance chart showing the number of particles tracked per second for different numbers of processes.]

- ReMPI becomes scalable by recording to local memory/storage
  - Each rank independently writes record → No communication across MPI ranks

![Diagram showing node communication and recording. Each node](https://example.com/diagram.png)
Record-and-replay won't work at scale

- Record-and-replay produces large amount of recording data
  - Over "10 GB/node" per day in MCB
  - Over "24 GB/node" per day in Diablo

- For scalable record-replay with low overhead, the record data must fit into local memory, but capacity is limited
  - Storing in shared/parallel file system is not scalable approach
  - Some systems may not have fast local storage

Challenges

Record size reduction for scalable record-replay
Clock Delta Compression (CDC)

sender 1

sender 2

sender 3

Logical clock

1

3

2

6

4

5

Receiver

Received order
(Order by wall-clock)

Logical order
(Order by logical-clock)

1

2

4

3

5

6

≈
Logical clock vs. wall clock

“The global order of messages exchanged among MPI processes are very similar to a logical-clock order (e.g., Lamport clock)“

![Diagram showing message exchange and Lamport clock values](image)

Each process frequently exchanges messages with neighbors.

![Graph showing Lamport clock values](image)

- Received messages in received order (MPI rank = 0)
- Lamport clock of received message
Clock Delta Compression (CDC)

- Our approach, clock delta compression, only records the difference between received order and logical order instead of recording entire received order.
Logical clock order is reproducible [1]

- Logical-clock order is always reproducible, so CDC only records the permutation difference.

Clock Delta Compression (CDC)

- Our approach, clock delta compression, only records the difference between received order and logical order instead of recording entire received order.

This logical order is reproducible.
Implementation

- We use PMPI wrapper
  - Tracing message receive order
  - Clock piggybacking
- Clock piggybacking [1]
  - When sending an MPI message, the PMPI wrapper defines a new MPI_Datatype that combines message payload & clock

Compression improvement in MCB

Total compressed record sizes on MCB at 3,072 procs (12.3 sec)

- w/o Compression: 196 MB
- gzip: 25 MB
- CDC: 5 MB

- gzip itself can reduce the original format by 8x
- 5x more reduction

- For example, if 1GB of memory per node for record-and-replay ...
  - w/o compression: 2 hours
  - gzip: 19 hours
  - CDC: 4 days

High compression

Compressed size becomes 40x smaller than original size
Summary

- Non-determinism is a common issue in debugging MPI applications
- ReMPI can help to reproduce buggy MPI behaviors with minimum record size