Symbolic Verification of Message Passing Interface Programs

Zhenbang Chen

( zbchen@nudt.edu.cn )

Joint work with Hengbiao Yu, Xianjin Fu, Ji Wang, Zhendong Su, Jun Sun, Chun Huang, and Wei Dong
High Performance Computing
MPI Paradigm

• Key features of MPI Applications
  • Non-deterministic, e.g. wildcard receive
  • **Blocking** and **non-blocking** communications
  • Data and control intensive
• MPI programs are hard to develop and maintain

*Verifying MPI programs is challenging*
Challenges

• Non-determinism (Soundness)

• Input & Schedule coverage
Challenges

- Non-determinism (Soundness)
- Input & Schedule coverage

```
Send(P1, buff0)

P0

if (x != 'a'){
    Recv(P0, buff1)
} else {
    Recv(ANY, buff1)
}

P1

Recv(P2, buff1)

P2

Send(P1, buff2)

Deadlock
```
## Existing Work of MPI Verification

<table>
<thead>
<tr>
<th></th>
<th>Automation</th>
<th>Input Coverage</th>
<th>Non-Blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CIVL(TASS)</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>MOPPER</strong></td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>MPI-SPIN</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Either do not support **input-related verification** or fail to support the verification of **non-blocking** MPI programs.
## Existing Work of MPI Verification

<table>
<thead>
<tr>
<th></th>
<th>Automation</th>
<th>Input Coverage</th>
<th>Non-Blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIVL(TASS)</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MOPPER</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>MPI-SPIN</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MPI-SV</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Challenges

• The explosion of state space

• Parallel execution, wildcard receive, branches

• Exponential increasing
Key Ideas (1/2)

- Challenge 1: non-determinism
- Symbolic execution for input coverage
- Blocking-driven matching of wildcard receives
Key Ideas (1/2)

- Challenge 1: non-determinism
- Symbolic execution for input coverage
- Blocking-driven matching of wildcard receives

The happens-before requirements in MPI standard are preserved
Key Ideas (2/2)

- Challenge 2: the explosion of state space
- Partial order reduction
- Model checking-based boosting
Basic Framework

MPI Programs

Property

Symbolic Executor

State Pruner

CSP Model Checker

CSP Model

Violation Path

Yes

Test Case

No

Yes

No

MPI-SV

Consider the MPI program in Figure 1 and the property. Since MPI processes are memory independent, the possible matched messages, especially for non-blocking operations, significantly increase the complexity of MPI programs. A non-blocking operation does not block but returns immediately under the assumption of infinite local buffers. Hence, the intuition behind is to collect all the possible matchings for the wildcard receive operation for each candidate. Suppose MPI-SV forks a state for each candidate. Without loss of generality, we collect the possible matchings of a wildcard receive operation for the first candidate. If the matched operation is Send(1), we continue exploring the two branches as follows.

When exploring an MPI program, the MPI-SV parallel prover generates a test case.

In principle, MPI-SV could continue exploring the path space, and checks the property along the interleavings and matchings of the communication.

Figure 2 shows the basic framework of MPI-SV.

Symbolic Verification of MPI Programs Conference '17, July 2017, Washington, DC, USA
Basic Framework

MPI Programs

Property

LTL properties

Symbolic Executor

State Pruner

No Violation

CSP Model Checker

CSP Model

Violation Path

Yes

No

Test Case

MPI-SV
Basic Framework

Symbolic Verification of MPI Programs Conference'17, July 2017, Washington, DC, USA

2.2 Our Approach

Consider the MPI program in Figure 1 and the property. Since MPI processes are memory independent, MPI-SV utilizes a CSP model to systematically enumerate all possible interleavings of the processes and wildcard operations. When the path transformation is matched with all the possible matched messages, especially for non-blocking operations, MPI-SV checks the violation-free consistency of the property. If the model checker reports a counterexample, a violation is found. Otherwise, MPI-SV utilizes a CSP model to perform the message exchanges as thoroughly as possible, which significantly increases the complexity of MPI-SV.

The basic framework of MPI-SV is shown in Figure 2. We adopt symbolic executor to automatically explore all possible paths.

To ensure soundness, all possible inputs of the MPI program, the interleaving sequence of MPI operations, and the wildcard receive operation matching needs to be done. In this way, MPI-SV helps to get all paths need to be explored. However, non-blocking and wildcard operations tend to result in an explosion problem. However, it becomes even more significant when there is a non-determinism caused by the input wildcard operations. The intuition behind is to collect the complete interleavings of the matched processes. Obviously, the deadlock will be found in the finite local buffer and all the other processes terminate. When the path P terminates, the deadlock is recorded.
Basic Framework

Consider the MPI program in Figure 1 and the property. If the model checkers violate the property, called a test case. Figure 2 shows the basic framework of MPI-SV.

2.2 Our Approach

The second challenge is how to significantly increase the complexity of MPI program verification. For example, consider the MPI program in Figure 1 and the property. If the model checkers violate the property, called a test case. Figure 2 shows the basic framework of MPI-SV.

Symbolic Executor

State Pruner

CSP Model Checker

CSP Model

MPI-SV

Test Case

Violation Path

Yes

No

No Violation

Yes

Symbolic Verification of MPI Programs Conference'17, July 2017, Washington, DC, USA
Basic Framework

Figure 2 shows the basic framework of MPI-SV.

Symbolic Verification of MPI Programs Conference'17, July 2017, Washington, DC, USA
Basic Framework

MPI Programs

Property

Symbolic Executor

State Pruner

Violation

No

Yes

CSP Model Checker

CSP Model

Violation Path

No

Yes

Test Case

MPI-SV
MPI-SV uses the built-in symbolic executor to automatically explore the path space, and checks the property along the path. Figure 2 shows the basic framework of MPI-SV.

```
Consider the MPI program in Figure 1 and the
Symbolic Executor
CSP Model Checker
CSP Model

Symbolic Verification of MPI Programs Conference'17, July 2017, Washington, DC, USA
```
Basic Framework

CSP modeling is sound and complete
Basic Framework

CSP modeling is sound and complete
High-Level View

Automatic Model Extraction

Symbolic Execution

Model Checking

Boosting

Enlarge the scope of verifiable properties
Experiments

- **Property**
- **Deadlock freedom**
- **Non-reachability properties**
- Analyze each task (program/property) in one hour

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTG</td>
<td>90</td>
<td>Dependence transition group</td>
</tr>
<tr>
<td>Matmat</td>
<td>105</td>
<td>Matrix multiplication</td>
</tr>
<tr>
<td>Integrate</td>
<td>181</td>
<td>Integral computing</td>
</tr>
<tr>
<td>Diffusion2d</td>
<td>197</td>
<td>Simulation of diffusion equation</td>
</tr>
<tr>
<td>Gauss_elim</td>
<td>341</td>
<td>Gaussian elimination</td>
</tr>
<tr>
<td>Heat</td>
<td>613</td>
<td>Heat equation solver</td>
</tr>
<tr>
<td>Mandelbrot</td>
<td>268</td>
<td>Mandelbrot set drawing</td>
</tr>
<tr>
<td>Sorting</td>
<td>218</td>
<td>Array sorting</td>
</tr>
<tr>
<td>Image_manip</td>
<td>360</td>
<td>Image manipulation</td>
</tr>
<tr>
<td>DepSolver</td>
<td>8988</td>
<td>Multimaterial electrostatic solver</td>
</tr>
<tr>
<td>K fray</td>
<td>12728</td>
<td>KF-Ray parallel raytracer</td>
</tr>
<tr>
<td>ClusterW</td>
<td>23265</td>
<td>Multiple sequence alignment</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>47354</strong></td>
<td><strong>12 open source MPI programs</strong></td>
</tr>
</tbody>
</table>

Main Results

III verification tasks

<table>
<thead>
<tr>
<th>Program (Benchmark)</th>
<th>F</th>
<th>Deadlock</th>
<th>No Modeling</th>
<th>CSP Modeling</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlobFlow</td>
<td>y</td>
<td>17</td>
<td>61 (55%)</td>
<td>100 (90%)</td>
<td>&gt;5X</td>
</tr>
<tr>
<td>Gauss smoother</td>
<td>y</td>
<td>44</td>
<td></td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>Heat</td>
<td>y</td>
<td>17</td>
<td></td>
<td>52</td>
<td>&gt;19X</td>
</tr>
<tr>
<td>Mandelbrot</td>
<td>y</td>
<td>17</td>
<td></td>
<td>52</td>
<td>&gt;19X</td>
</tr>
</tbody>
</table>

MPI-SV is more effective and efficient than pure symbolic execution.
The Progress Distribution

<table>
<thead>
<tr>
<th>#Completed Tasks</th>
<th>Verification Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Modeling</td>
<td>CSP Modeling</td>
</tr>
</tbody>
</table>

MPI-SV can complete more tasks under the same time threshold.
Conclusion

High Performance Computing

Existing Work of MPI Verification

<table>
<thead>
<tr>
<th>Tool</th>
<th>Automation</th>
<th>Input Coverage</th>
<th>Non-Blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIVL(TASS)</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MOPPER</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>MPI-SPIN</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MPI-SV</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

High-Level View

Automatic Model Extraction

Symbolic Execution

Model Checking

Boosting

Enlarge the scope of verifiable properties

Main Results

<table>
<thead>
<tr>
<th>111 verification tasks</th>
<th>No Modeling</th>
<th>CSP Modeling</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finished</td>
<td>61</td>
<td>100</td>
<td>&gt;5X</td>
</tr>
<tr>
<td>Deadlock</td>
<td>44</td>
<td>48</td>
<td>&gt;5X</td>
</tr>
<tr>
<td>No Deadlock</td>
<td>17</td>
<td>52</td>
<td>&gt;19X</td>
</tr>
</tbody>
</table>

MPI-SV is more effective and efficient than pure symbolic execution
Thank you!

Q&A

https://mpi-sv.github.io