COMPI: Concolic Testing for MPI Applications

Hongbo Li, Sihuan Li, Zachary Benavides, Zizhong Chen, and Rajiv Gupta

May 24th, 2018
Testing in Industry

- Software bugs can be VERY costly
  - In 1998, the crash of NASA’s Mars Climate Orbiter costs $125 millions
  - In 2004, a software bug in the child support agency IT system in UK costs over $1 billion

- Testing is widely used in industry to ensure code quality
HPC Also Needs Testing

- The study of practical systematic testing techniques is scarce in the field of HPC

- HPC applications drives scientific discovery and technological innovation

- The study of testing is a must in our field
Outline

Concolic Testing

Challenges & Solutions

Evaluation
Outline

Concolic Testing

Challenges & Solutions

Evaluation
/**
 * Concolic Testing
 */

int x, y;
mark_symbolic(x);
mark_symbolic(y);

// branch condition 0
if (x != 100)
    OT: work1();
else
    OF: ABORT;

x = x / 2;
// branch condition 1
if (x + y > 200)
    1T: work2();
else
    1F: work3();
}
Concolic Testing

```c
main() {
    int x, y;
    mark_symbolic(x);
    mark_symbolic(y);

    // branch condition 0
    if (x != 100)
        OT: work1();
    else
        OF: ABORT;

    x = x / 2;
    // branch condition 1
    if (x + y > 200)
        1T: work2();
    else
        1F: work3();
}
```

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Branches</th>
<th>Constraints</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10, 50]</td>
<td></td>
<td>[10, 50]</td>
<td></td>
</tr>
</tbody>
</table>
Concolic Testing

```c
main() {
    int x, y;

    mark_symbolic(x);
    mark_symbolic(y);

    // branch condition 0
    if (x != 100)
        0T: work1();
        else
            0F: ABORT;

    x = x / 2;
    // branch condition 1
    if (x + y > 200)
        1T: work2();
        else
            1F: work3();
}
```

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Branches</th>
<th>Constraints</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10, 50]</td>
<td>{0T, 1F}</td>
<td>{x \neq 100, x/2 + y \leq 200}</td>
<td>50%</td>
</tr>
</tbody>
</table>
main() {
    int x, y;
    mark_symbolic(x);
    mark_symbolic(y);

    // branch condition 0
    if (x != 100)
        OT: work1();
    else
        OF: ABORT;

    x = x / 2;
    // branch condition 1
    if (x + y > 200)
        1T: work2();
    else
        1F: work3();
}

Inputs  Branches  Constraints  Coverage
[10, 50]  {0T, 1F}  {x != 100, x/2 + y <= 200}  50%

Select and negate x != 100
Concolic Testing

```c
main() {
    int x, y;
    mark_symbolic(x);
    mark_symbolic(y);

    // branch condition 0
    if (x != 100)
        OT:    work1();
    else
        OF:    ABORT;

    x = x / 2;
    // branch condition 1
    if (x + y > 200)
        1T:    work2();
    else
        1F:    work3();
}
```

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Branches</th>
<th>Constraints</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10, 50]</td>
<td>{OT, 1F}</td>
<td>{x \neq 100, x/2 + y \leq 200}</td>
<td>50%</td>
</tr>
</tbody>
</table>

Select and negate \( x \neq 100 \)

Solve them and generate inputs
Concolic Testing

```c
main() {
    int x, y;
    mark_symbolic(x);
    mark_symbolic(y);

    // branch condition 0
    if (x != 100)
    OT:  work1();
    else
    OF:  ABORT;

    x = x / 2;
    // branch condition 1
    if (x + y > 200)
    1T:  work2();
    else
    1F:  work3();
}
```

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Branches</th>
<th>Constraints</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10, 50]</td>
<td>{OT, 1F}</td>
<td>{x != 100, x/2 + y &lt;= 200}</td>
<td>50%</td>
</tr>
</tbody>
</table>

Select and negate \(x \neq 100\)

\{x = 100, x/2 + y <= 200\}

Solve them and generate inputs

\([100, 50]\)
Concolic Testing

```c
main() {
    int x, y;
    mark_symbolic(x);
    mark_symbolic(y);

    // branch condition 0
    if (x != 100)
        OT: work1();
    else
        OF: ABORT;

    x = x / 2;
    // branch condition 1
    if (x + y > 200)
        1T: work2();
    else
        1F: work3();
}
```

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Branches</th>
<th>Constraints</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10, 50]</td>
<td>{0T, 1F}</td>
<td>{x != 100, x/2 + y &lt;= 200}</td>
<td>50%</td>
</tr>
</tbody>
</table>

Select and negate `x != 100`

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Branches</th>
<th>Constraints</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>[100, 50]</td>
<td>{0F, 1F}</td>
<td>{x == 100, x/2 + y &lt;= 200}</td>
<td>75%</td>
</tr>
</tbody>
</table>

Solve them and generate inputs.
Concolic Testing

```c
main() {
    int x, y;
    mark_symbolic(x);
    mark_symbolic(y);
    
    // branch condition 0
    if (x != 100)
        OT: work1();
            else
            OF: ABORT;
        
    x = x / 2;
    // branch condition 1
    if (x + y > 200)
        1T: work2();
            else
            1F: work3();
}
```

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Branches</th>
<th>Constraints</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10, 50]</td>
<td>{0T, 1F}</td>
<td>{x != 100, x/2 + y &lt;= 200}</td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Select and negate x != 100</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>{x == 100, x/2 + y &lt;= 200}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Solve them and generate inputs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{0F, 1F}</td>
<td>{x == 100, x/2 + y &lt;= 200}</td>
<td>75%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Select and negate x/2 + y &lt;= 200</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>{x == 100, x/2 + y &gt; 200}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Solve them and generate inputs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{0F, 1T}</td>
<td>{x == 100, x/2 + y &lt;= 200}</td>
<td>100%</td>
</tr>
</tbody>
</table>
```
Outline

Concolic Testing

Challenges & Solutions

Evaluation
Challenge (1)

- Fail to tackle important MPI semantics
  - Multi-process execution
    - Branch coverage using ONLY one process is not enough!
    - How many processes should be used?
  - MPI rank
    - Which process should be the FOCUS process that is used for input generation (concolic testing)?
Challenge (1)

- Fail to tackle important MPI semantics
  - Multiple processes
    - Branch coverage using ONLY one process is not enough!
    - How many processes should be used?
  - MPI rank
    - Which process should be the FOCUS process that is used for input generation (concolic testing)?
Concolic testing with ONLY process 0

- Fail to record branches 3F & 4T
- Fail to uncover branch 4F
Solution (1)

› COMPI’s Framework
  › Record branch coverage based on ALL processes
  › Dynamically vary the number of processes
  › Dynamically vary the focus
main() {
    int x, y, rank;
    MPI_Init();
    mark_symbolic(x), mark_symbolic(y);
    mark_symbolic_mpi(rank);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    // sanity check: branch cond. 0, 1 & 2
    if (x < ... || y < ... || x*y > ...)
        ABORT;
    3T: if (rank == 0) split_work_and_send_work();
    3F: else {
        receive_work();
        4T: if (y < 100) ...
        4F: else ABORT;
    }
    while(...) {
    5T: compute_and_exchange_data();
    }
    5F: MPI_Finalize();
}
Challenge (2)

- Too high testing cost hinders COMPI’s practicality
  - Too large input value
    - Require long execution time
    - Break testing platform’s memory limit
    - Crash a computer when too many processes are started
Execution time and coverage for HPL using different matrix sizes.

- **Solution:** input capping --- set an upper bound for input variables that dominate a program’s execution time
Challenge (2)

- Too high testing cost hinders COMPI’s practicality
  - Too large input value
  - Heavy instrumentation
Two-way instrumentation incurs less I/O.

- One-way instrumentation
  - launch all processes including non-focus processes with the same heavily instrumented program

- Solution: two-way instrumentation
  - launch only the focus process with the heavily instrumented program and launch non-focus processes with lightly instrumented program
Challenge (2)

- Too high testing cost hinders COMPI’s practicality
  - Too large inputs
  - Heavy instrumentation
  - Redundant constraints in loops
• Solution: constraints reduction --- only record a constraint (a) at the first time a branch is encountered or (b) the branch’s evaluated Boolean value changes
• **Solution: constraints reduction** --- Only record a constraint (a) at the first time a branch is encountered or (b) the branch’s evaluated Boolean value changes.
Solution Summary

- Concolic testing framework targeting MPI programs
- Controlling testing cost
  - Input capping
  - Two-way instrumentation
  - Constraints reduction
Outline

Concolic Testing

Challenges & Solutions

Evaluation
Evaluation Setting

- Hardware platform
  - One single computer with two intel E5607 CPUs totaling 8 cores and 32 GB DRAM

- Programs

<table>
<thead>
<tr>
<th>Programs</th>
<th>Lines of code</th>
<th># Reachable branches</th>
<th>Selected variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUSY-HMC</td>
<td>19,201</td>
<td>2,030</td>
<td>Lattice size</td>
</tr>
<tr>
<td>HPL</td>
<td>15,699</td>
<td>3,754</td>
<td>Matrix width</td>
</tr>
<tr>
<td>IMB-MPI1</td>
<td>7,092</td>
<td>1,290</td>
<td># iterations</td>
</tr>
</tbody>
</table>

Denoted as $N$
Evaluation – Bugs

2 or 4 Processes fail the test!
1 or 3 processes succeed!
Evaluation – Controlling Testing Cost

Input capping forms the basis of practical testing
Evaluation – Controlling Testing Cost

- Two-way instrumentation saves up to 66% testing time cost

## One-way v.s. Two-way

<table>
<thead>
<tr>
<th>Program</th>
<th>N</th>
<th>Time cost (seconds)</th>
<th>Avg. log size (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1-way</td>
<td>2-way</td>
</tr>
<tr>
<td>SUSY-HMC</td>
<td>2</td>
<td>163</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>479</td>
<td>226</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>92</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>382</td>
<td>127</td>
</tr>
<tr>
<td>HPL</td>
<td>100</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>1600</td>
<td>43</td>
<td>38</td>
</tr>
<tr>
<td>IMB-MPI1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation – Controlling Testing Cost

- With constraints reduction COMPI achieves 4.7-10.6% more branch coverage than without using it.

High Reduction Efficiency: A few thousands or even millions to a few hundreds.
Evaluation – COMPI Framework

- COMPI (Fwk)
- No_Fwk: concolic testing without COMPI’s framework
- Random: random input values generated for each test

Effectiveness of COMPI’s framework.

<table>
<thead>
<tr>
<th>Program</th>
<th>COMPI (Fwk)</th>
<th>No_Fwk</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUSY-HMC</td>
<td>84.7%</td>
<td>86.1%</td>
<td>3.4%</td>
</tr>
<tr>
<td>HPL</td>
<td>69.4%</td>
<td>71.6%</td>
<td>58.9%</td>
</tr>
<tr>
<td>IMB-MPII</td>
<td>69.0%</td>
<td>69.1%</td>
<td>64.2%</td>
</tr>
</tbody>
</table>
Thank you!