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

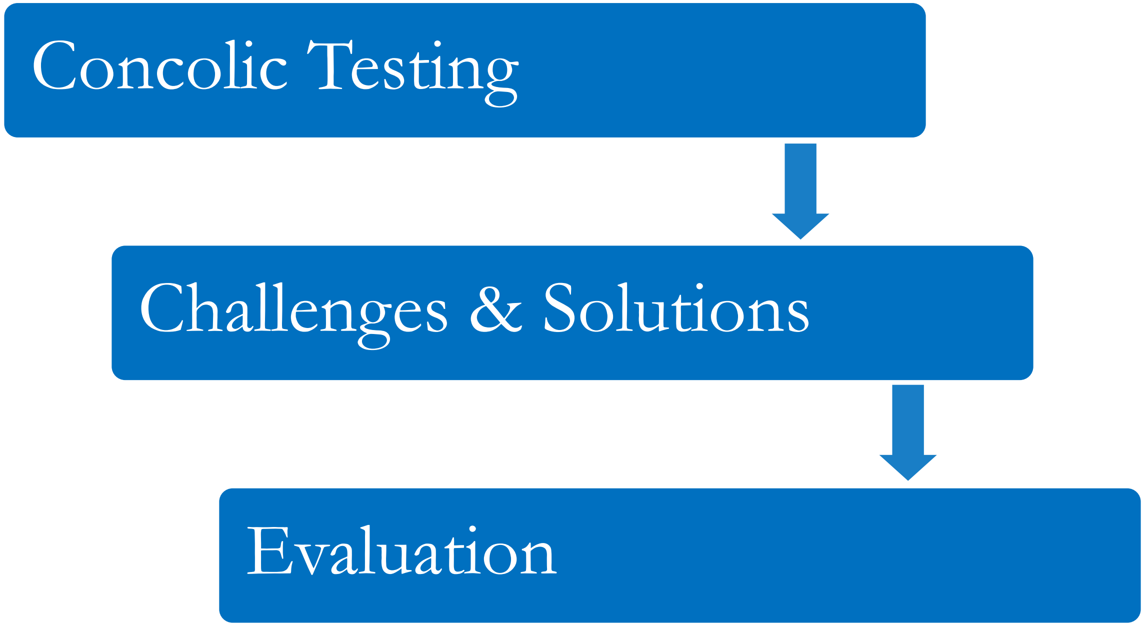
HPC Also Needs Testing

- 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

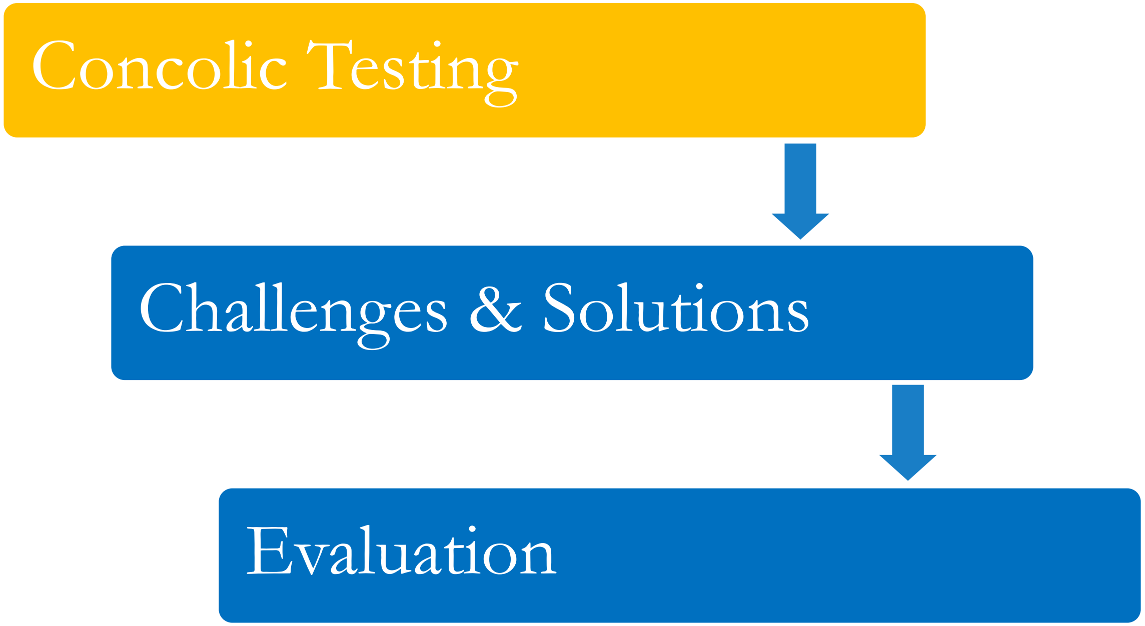


Concolic Testing

Challenges & Solutions

Evaluation

Outline

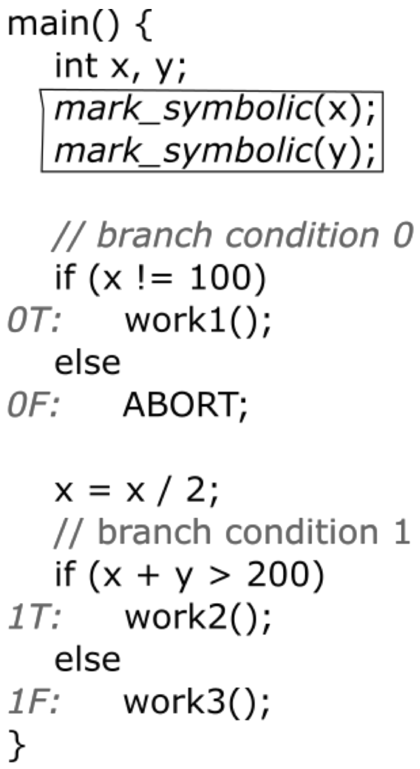


Concolic Testing

Challenges & Solutions

Evaluation

Concolic Testing



```
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();
}
```

Concolic Testing

```
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();
}
```

Inputs	Branches	Constraints	Coverage
[10, 50]			

Concolic Testing

```
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();
}
```

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

Concolic Testing

```
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();
}
```

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

Select and negate x != 100

Concolic Testing

```
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();
}
```

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

Select and negate x != 100

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

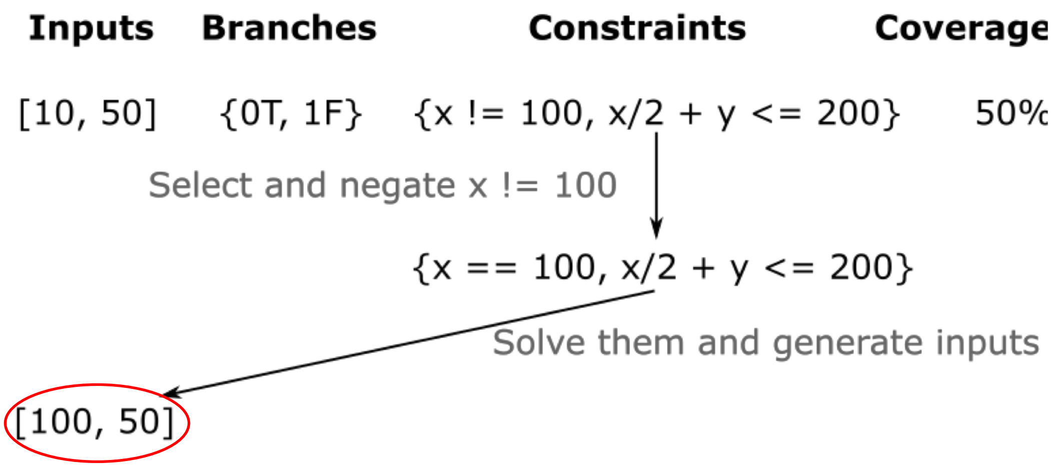
Solve them and generate inputs

Concolic Testing

```
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();
}
```



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

Select and negate x != 100

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

Solve them and generate inputs

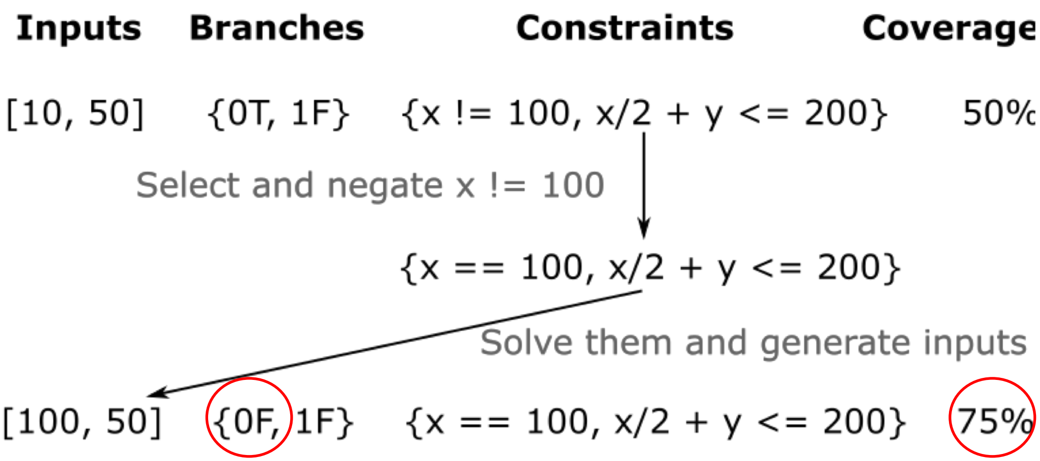
[100, 50]

Concolic Testing

```
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();
}
```



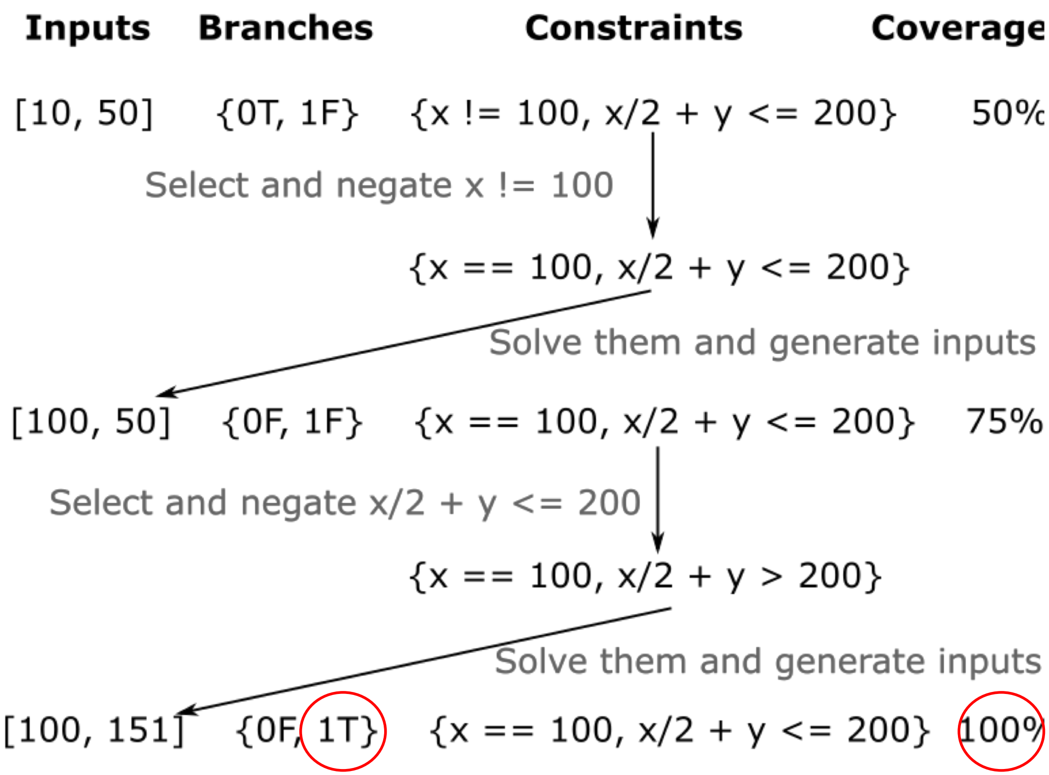
Inputs	Branches	Constraints	Coverage
[10, 50]	{0T, 1F}	{x != 100, x/2 + y <= 200}	50%
		Select and negate x != 100	
		{x == 100, x/2 + y <= 200}	
		Solve them and generate inputs	
[100, 50]	{0F, 1F}	{x == 100, x/2 + y <= 200}	75%

Concolic Testing

```
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();
}
```



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

Select and negate x != 100

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

Solve them and generate inputs

Inputs	Branches	Constraints	Coverage
[100, 50]	{0F, 1F}	{x == 100, x/2 + y <= 200}	75%

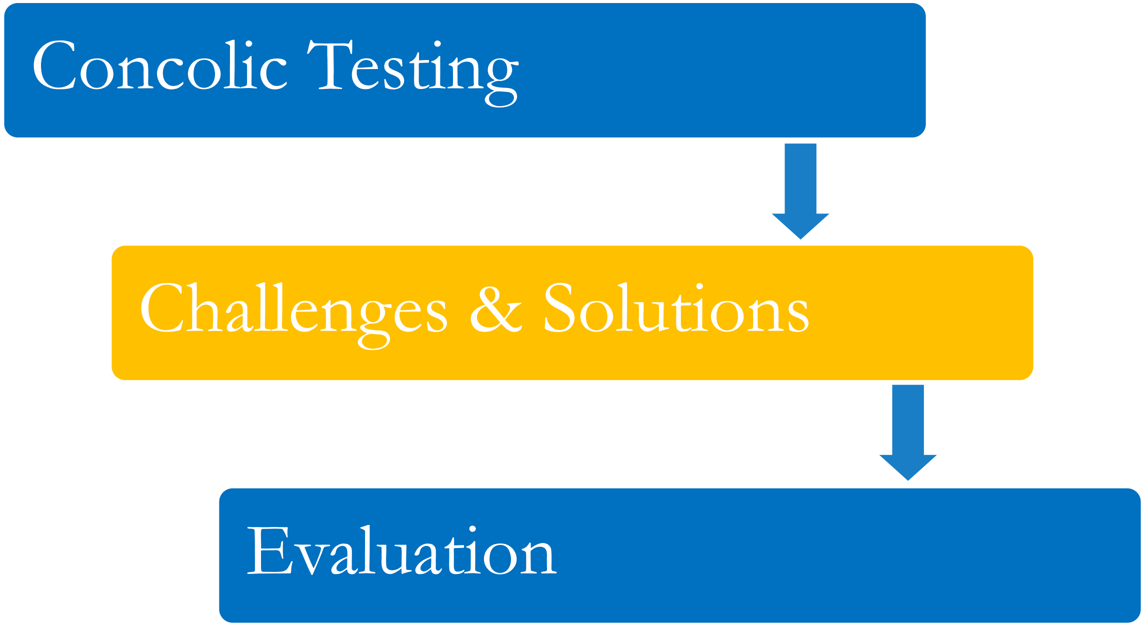
Select and negate x/2 + y <= 200

{x == 100, x/2 + y > 200}

Solve them and generate inputs

Inputs	Branches	Constraints	Coverage
[100, 151]	{0F, 1T}	{x == 100, x/2 + y <= 200}	100%

Outline



Concolic Testing

Challenges & Solutions

Evaluation

Challenge (1)

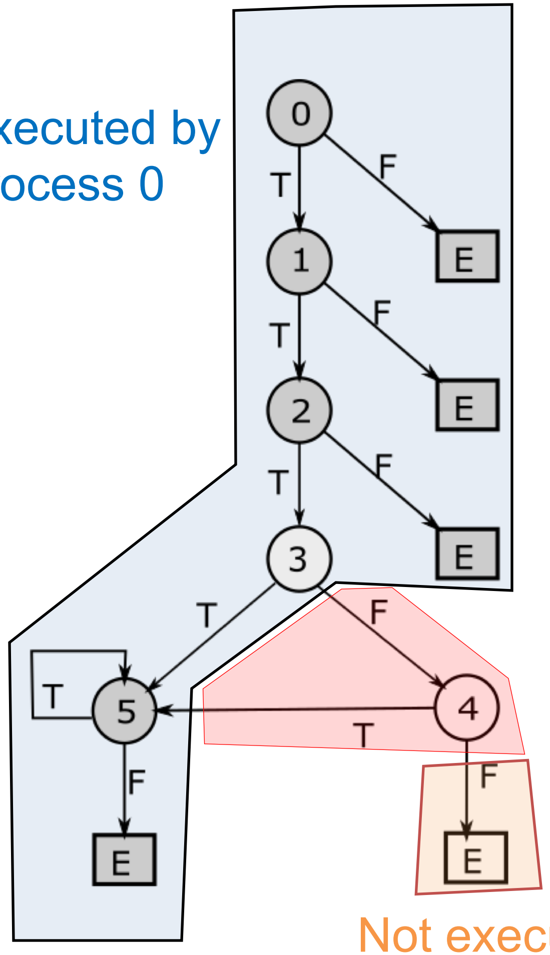
- Fail to tackle important MPI semantics
  - Multi-process execution
    - Branch coverage using ONLY one process is not enough!
    - How many processes should be used?
  - MPI rank
    - Which process should be the FOCUS process that is used for input generation (concolic testing)?

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)?

```
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  {
          recieve_work();
4T:        if (y < 100)   …
4F:       else ABORT;
      }
      while(…) {
5T:compute_and_exchange_data();
      }
5F: MPI_Finalize();
}
```

Executed by process 0



Executed process *i* ($(i \neq 0)$)

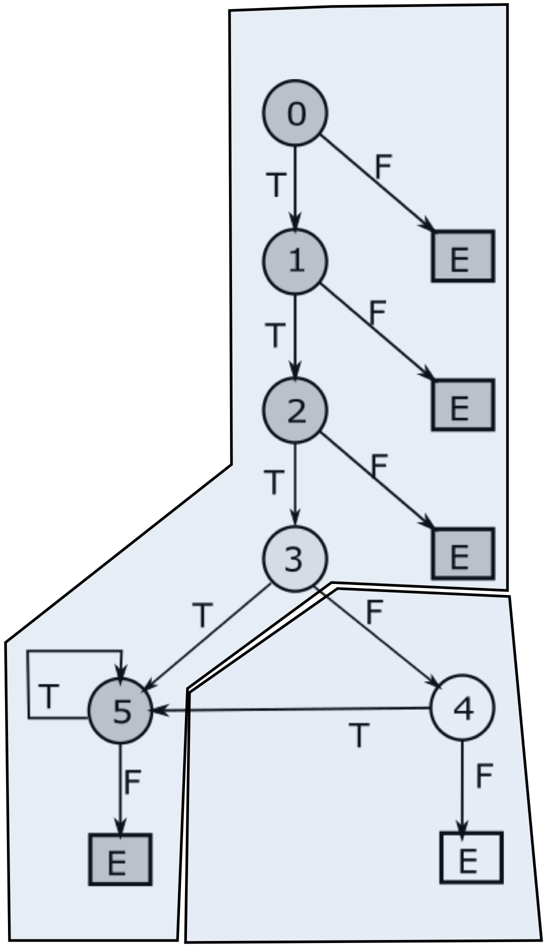
Not executed

- 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

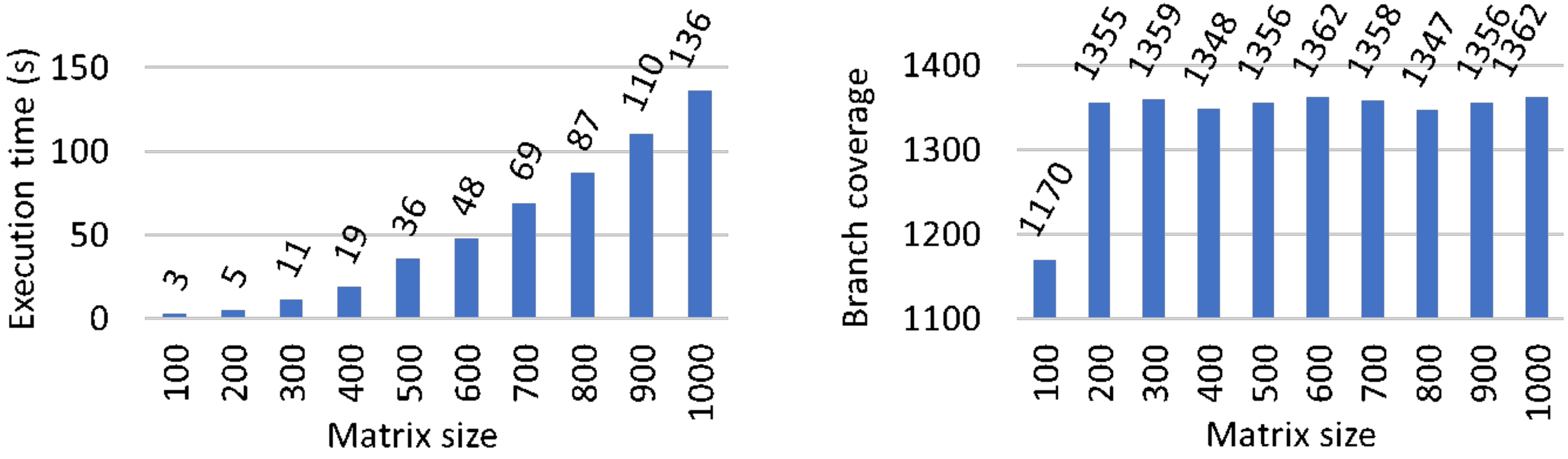
```
main() {
     int x, y, rank;
     MPI_Init();
     mark_symbolic(x), mark_symbolic(y);
     mark_symbolic_mpi(rank);
     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
     // sanity check: branch cond. 0, 1 & 2
     if (x < ... || y < ... || x*y > ...)
          ABORT;
3T: if (rank == 0)    split_work_and_send_work();
3F: else  {
          recieve_work();
4T:        if (y < 100)  ...
4F:       else ABORT;
      }
      while(...) {
5T:compute_and_exchange_data();
      }
5F: MPI_Finalize();
}
```



- Concolic testing USING our Framework
  - Help uncover: 3F & 4T
  - Help uncover branch *4F*

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

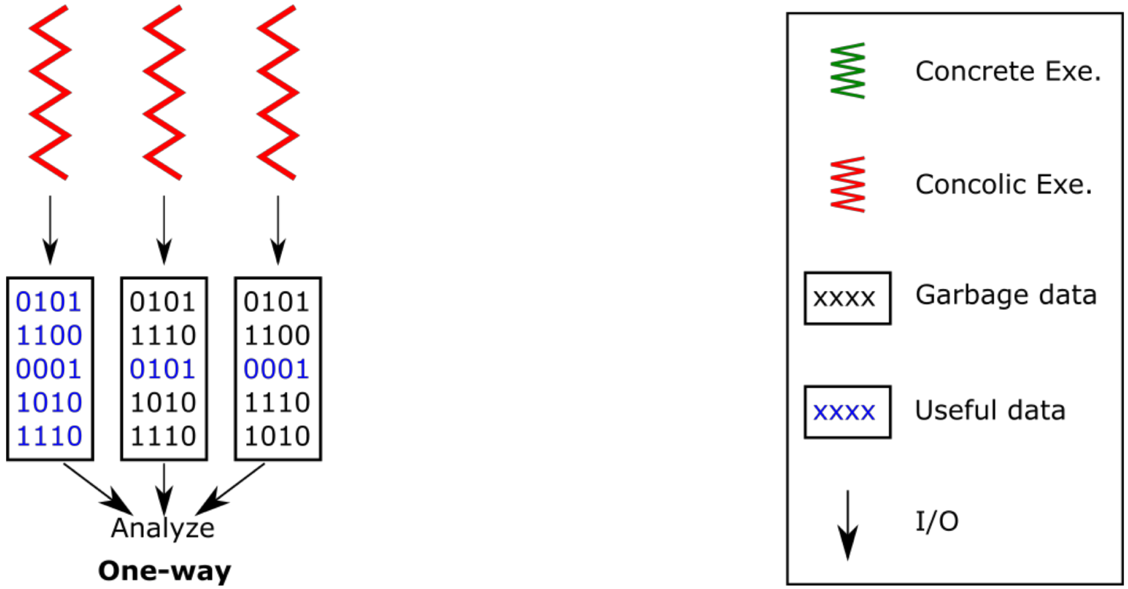


Execution time and coverage for HPL using different matrix sizes.

- **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

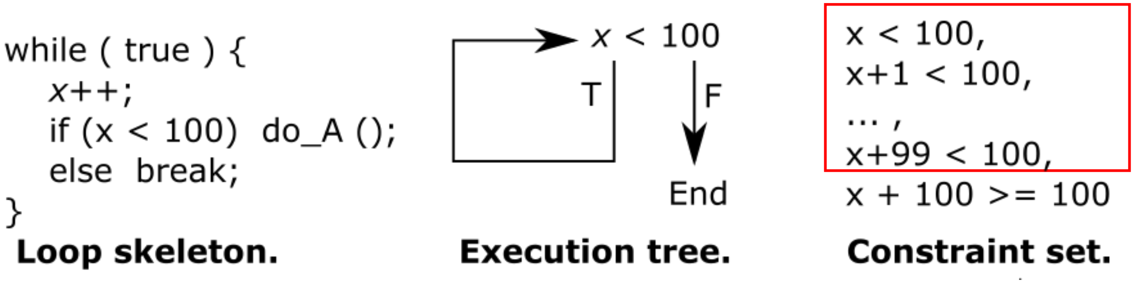


Two-way instrumentation incurs less I/O.

- 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



```
while ( true ) {
    x++;
    if (x < 100)  do_A ();
    else  break;
}
```

Loop skeleton.

Execution tree.

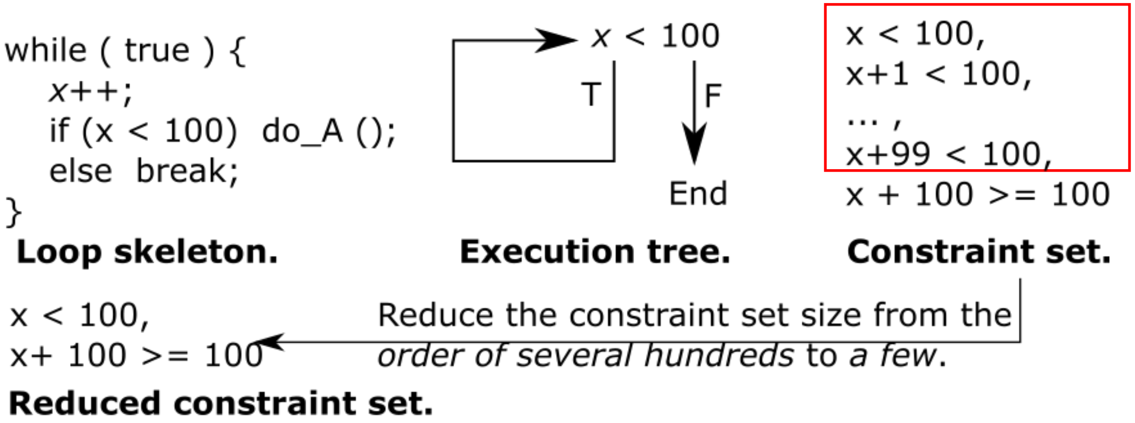
x < 100,
x+1 < 100,
... ,
x+99 < 100,
x + 100 >= 100

Constraint set.

$$\{x \mid x + i < 100 \text{ and } 0 < i < 100\} \subset \{x \mid x < 100\}$$

Constraints reduction.

- **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



```
while ( true ) {
   x++;
   if (x < 100)  do_A ();
   else  break;
}
```

Loop skeleton.

Execution tree.

x < 100,
x+1 < 100,
… ,
x+99 < 100,
x + 100 >= 100

Constraint set.

Reduce the constraint set size from the *order of several hundreds* to a few.

x < 100,
x+ 100 >= 100

Reduced constraint set.

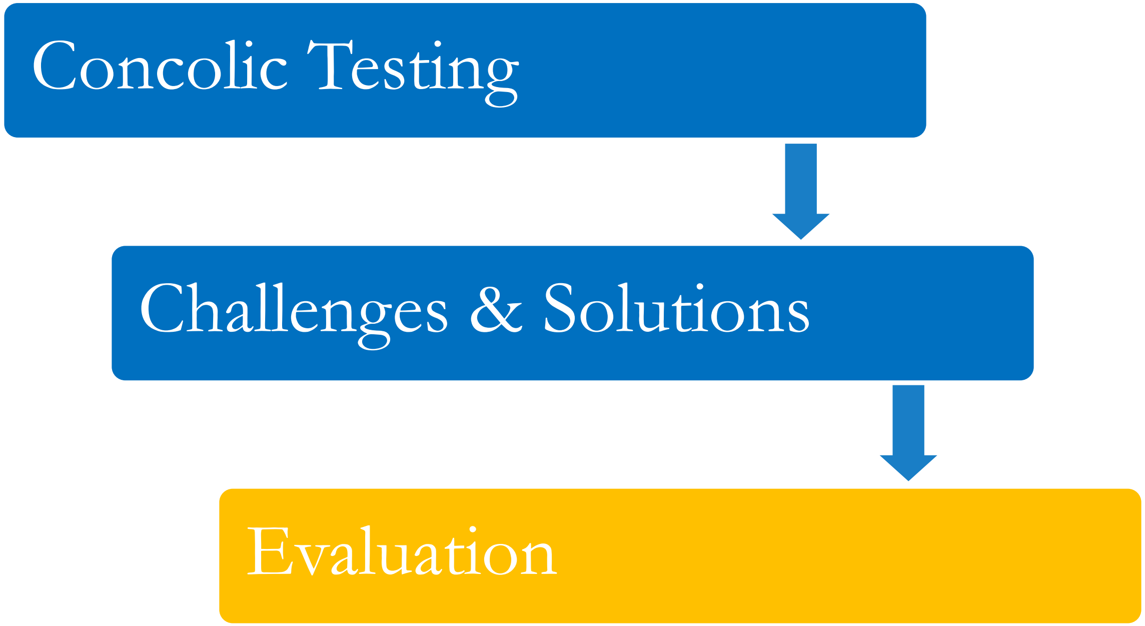
Constraints reduction.

- **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

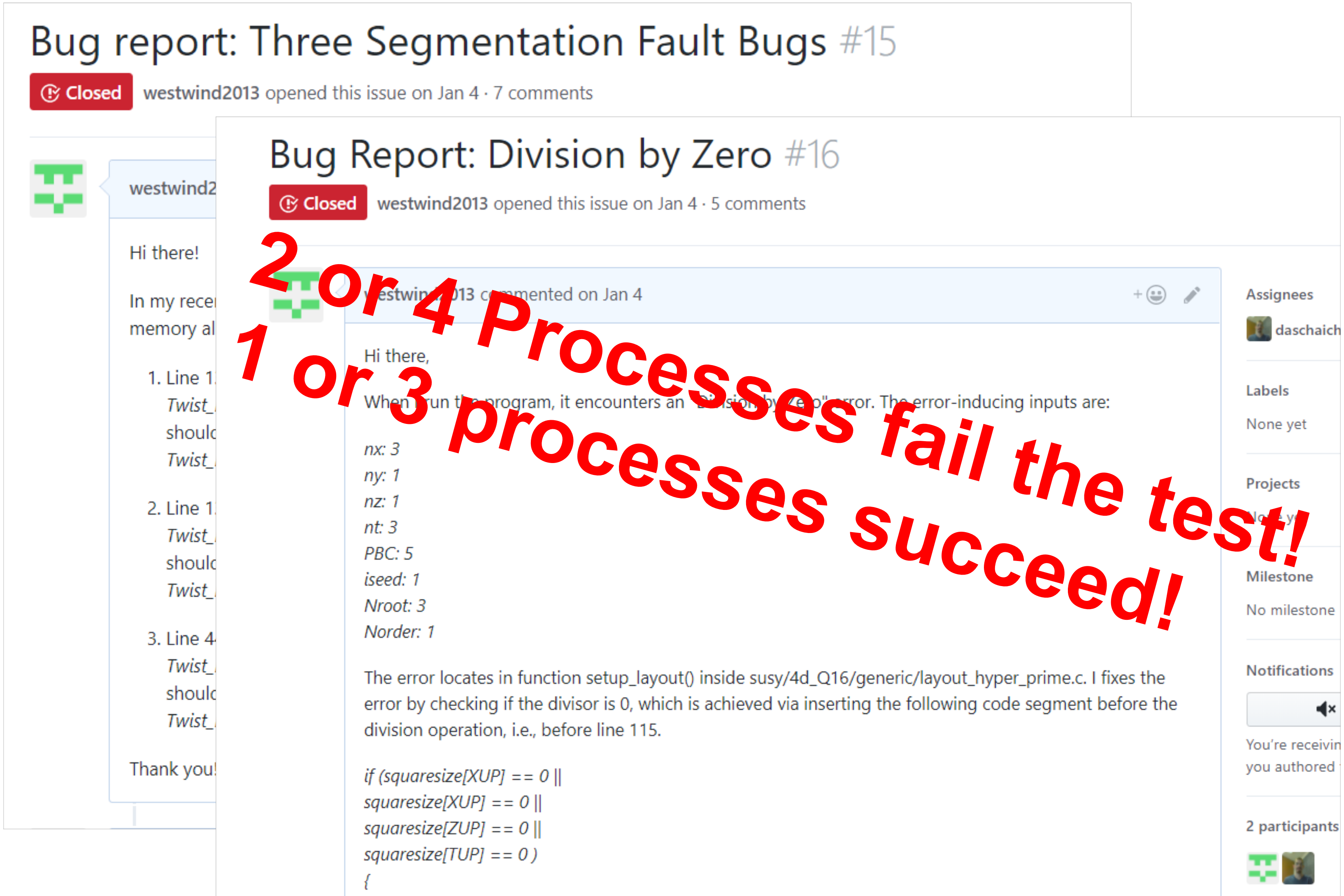
- Hardware platform
  - One single computer with two intel E5607 CPUs totaling 8 cores and 32 GB DRAM

- Programs

Programs	Lines of code	# Reachable branches	Selected variable
SUSY-HMC	19,201	2,030	Lattice size
HPL	15,699	3,754	Matrix width
IMB-MPI1	7,092	1,290	# iterations

Denoted as N

Evaluation – Bugs



Bug report: Three Segmentation Fault Bugs #15

Closed westwind2013 opened this issue on Jan 4 · 7 comments

westwind2

Hi there!

In my rece
memory al

1. Line 1
   Twist_
   should
   Twist_
2. Line 1
   Twist_
   should
   Twist_
3. Line 4
   Twist_
   should
   Twist_

Thank you!

Bug Report: Division by Zero #16

Closed westwind2013 opened this issue on Jan 4 · 5 comments

westwind2013 commented on Jan 4

Hi there,

When I run the program, it encounters an "Division by Zero" error. The error-inducing inputs are:

nx: 3
ny: 1
nz: 1
nt: 3
PBC: 5
iseed: 1
Nroot: 3
Norder: 1

The error locates in function setup_layout() inside susy/4d_Q16/generic/layout_hyper_prime.c. I fixes the error by checking if the divisor is 0, which is achieved via inserting the following code segment before the division operation, i.e., before line 115.

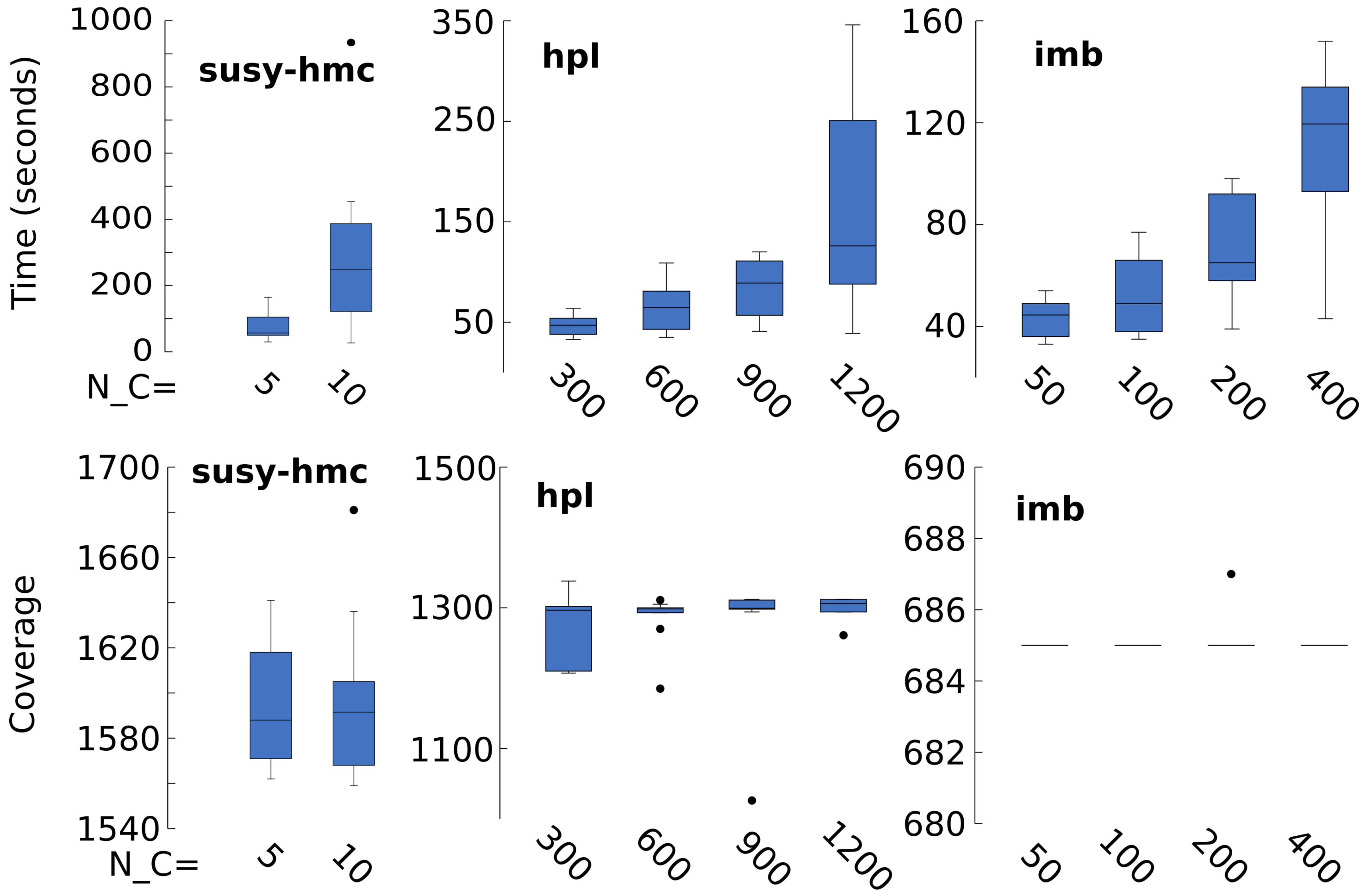
```
if (squaresize[XUP] == 0 ||
squaresize[XUP] == 0 ||
squaresize[ZUP] == 0 ||
squaresize[TUP] == 0 )
{
```

2 or 4 Processes fail the test!

1 or 3 processes succeed!

Evaluation – Controlling Testing Cost

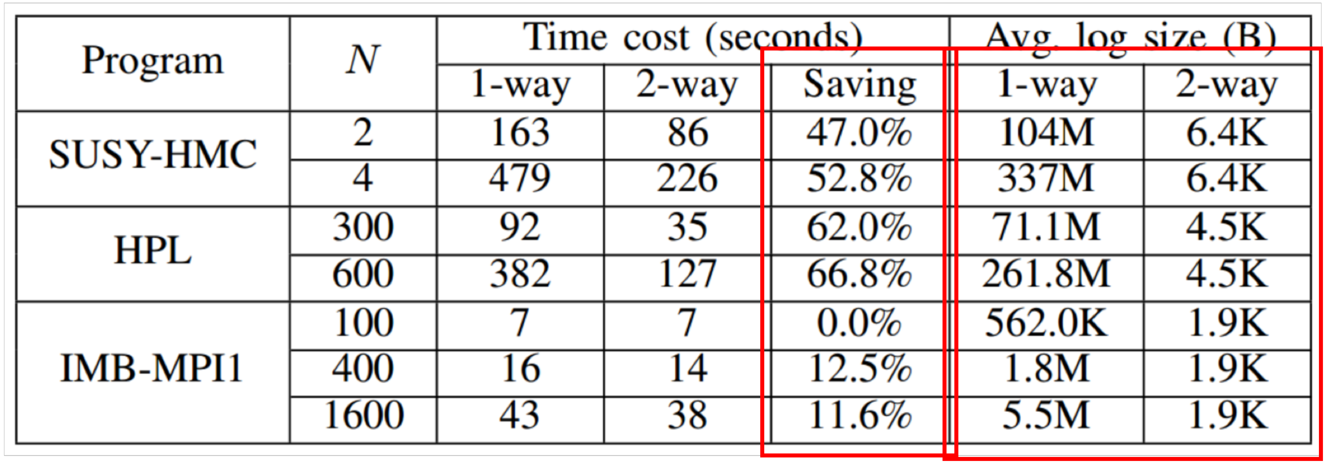
- **Input capping** forms the basis of practical testing



Evaluation – Controlling Testing Cost

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

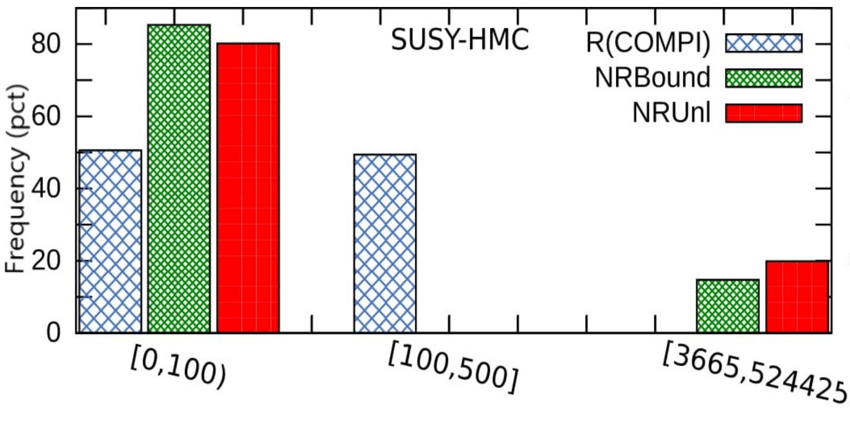
One-way v.s. Two-way

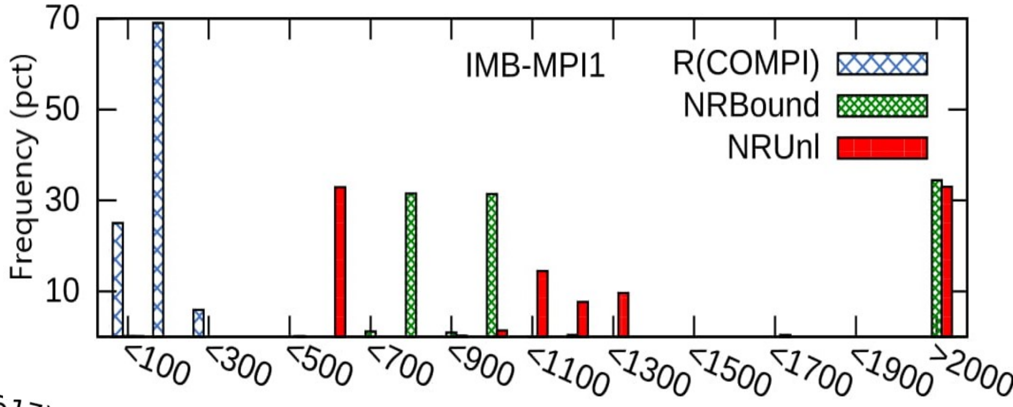


Program	*N*	Time cost (seconds)			Avg. log size (B)	
		1-way	2-way	Saving	1-way	2-way
SUSY-HMC	2	163	86	47.0%	104M	6.4K
	4	479	226	52.8%	337M	6.4K
HPL	300	92	35	62.0%	71.1M	4.5K
	600	382	127	66.8%	261.8M	4.5K
IMB-MPI1	100	7	7	0.0%	562.0K	1.9K
	400	16	14	12.5%	1.8M	1.9K
	1600	43	38	11.6%	5.5M	1.9K

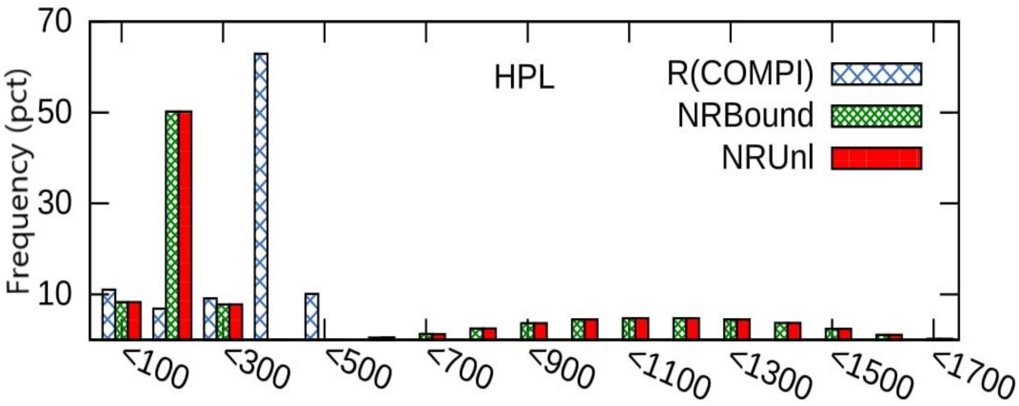
Evaluation – Controlling Testing Cost

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





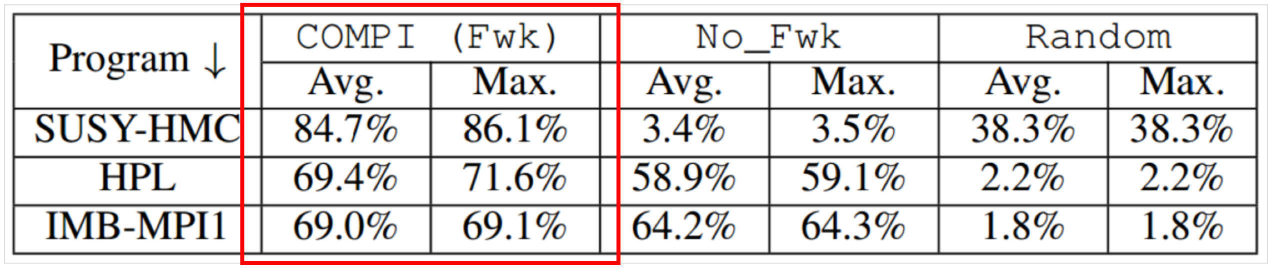
High Reduction Efficiency: A few thousands or even millions to a few hundreds



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.



Program ↓	COMPI (Fwk)		No_Fwk		Random	
	Avg.	Max.	Avg.	Max.	Avg.	Max.
SUSY-HMC	84.7%	86.1%	3.4%	3.5%	38.3%	38.3%
HPL	69.4%	71.6%	58.9%	59.1%	2.2%	2.2%
IMB-MPI1	69.0%	69.1%	64.2%	64.3%	1.8%	1.8%

Thank you!