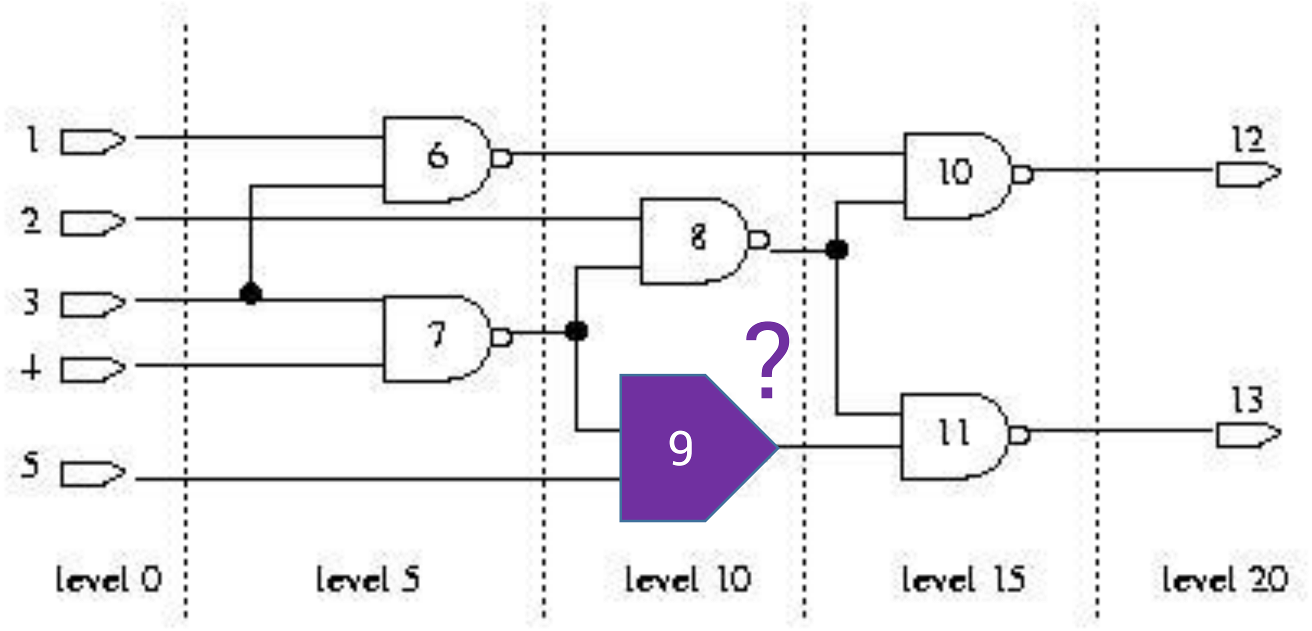
Reverse Engineering of Gate Level Netlists

For, CS5234: Advanced Parallel Computation

By, Sonal Pinto

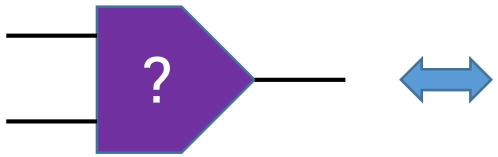
Problem Statement

Given a gate-level netlist, sequential or combinational, with unknown gates, and an incomplete simulation trace, identify the mystery gates that satisfy the simulation.



Motivation

2-input Mystery Gate

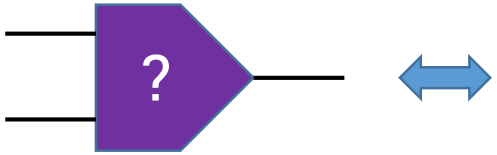


and	nand
or	nor
xor	xnor

From Library

Motivation

2-input Mystery Gate



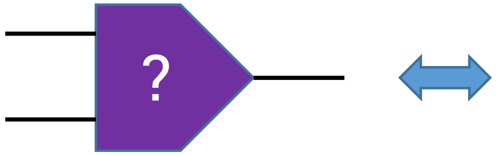
and	nand
or	nor
xor	xnor

For, N mystery Gates, the search space is: 6^N

8 mystery gates → 1.68 million possible solutions that need to be verified

Motivation

2-input Mystery Gate



and nand
or nor
xor xnor

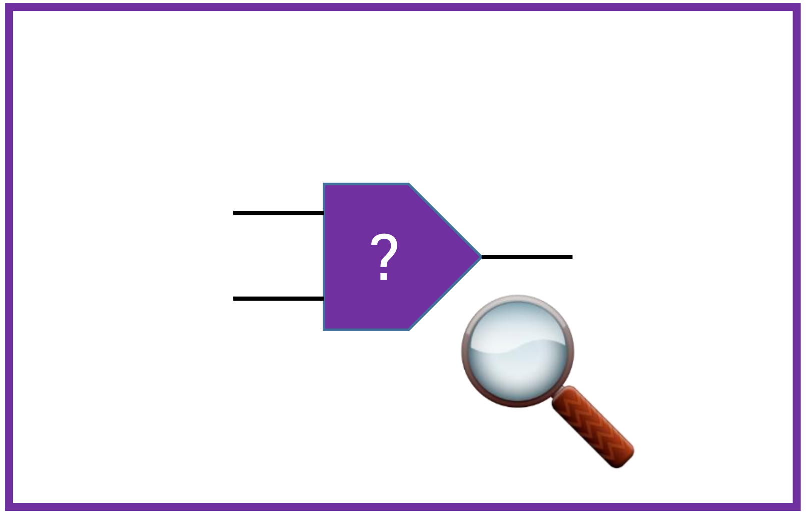
Brute Force – not scalable!

Goal: Develop and implement a practical parallelizable algorithm

Basic Principles

10001
00100
00000
00100
10110
11000
11001
01111
00000
01110
01000

Input



suspect_list,

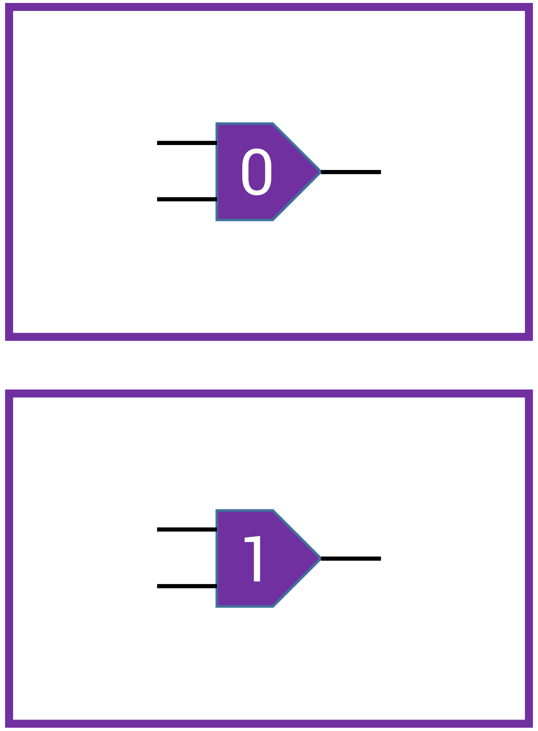
and nand
or nor
xor xnor

00
00
00
00
11
11
11
00
00
01
11

Output

Basic Principles

10001
00100
00000
00100
10110
11000
11001
01111
00000
01110
01000



00
00
00
00
11
11
11
00
00
01
11

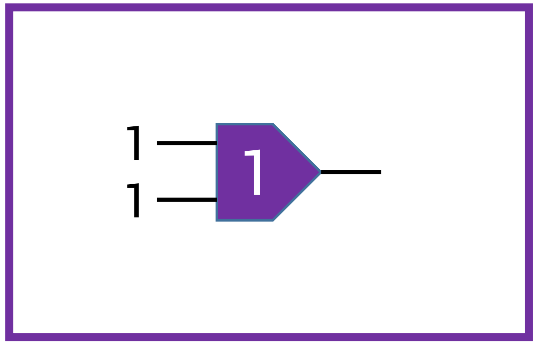
Introduce Stuck-at Faults at the mystery gate, and propagate them, by stepping into the simulation.

Reduce suspect_list for mystery gate based on valid circuit operation

Similar to filling up the Truth Table for that gate.

Basic Principles

10001
00100
00000
00100
10110
11000
11001
01111
00000
01110
01000



00
00
00
00
11
11
11
00
00
01
11

Input at the mGate:
11

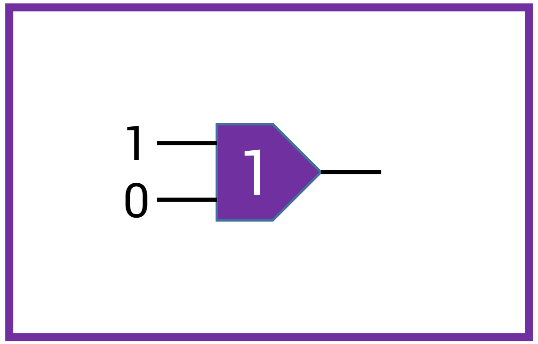
Valid mGate ouput:
1

Reduced suspect_list,

and nand
or nor
xor xnor

Basic Principles

10001
00100
00000
00100
10110
11000
11001
01111
00000
01110
01000



00
00
00
00
11
11
11
00
00
01
11

Input at the mGate:
10

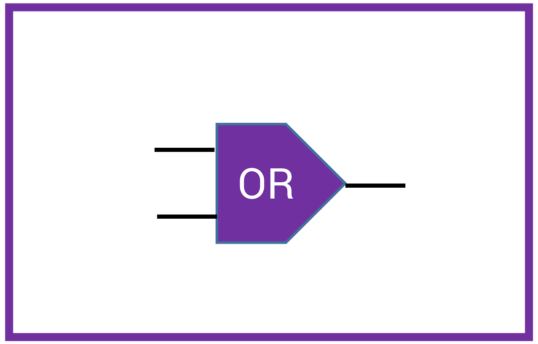
Valid mGate ouput:
1

Reduced suspect_list,

and	nand
or	nor
xor	xnor

Basic Principles

10001
00100
00000
00100
10110
11000
11001
01111
00000
01110
01000



00
00
00
00
11
11
11
00
00
01
11

Final suspect_list,

{ and nand
or nor
xor xnor }

Solution: mGate = OR

Step through the full simulation trace

Algorithm

For, N mystery gates, the local search space is, 2^N

8 mGates → local search size: 256

Consider a sim trace of length, K

Then, the Global search space is, 2^{NK} = *state explosion!*

Algorithm

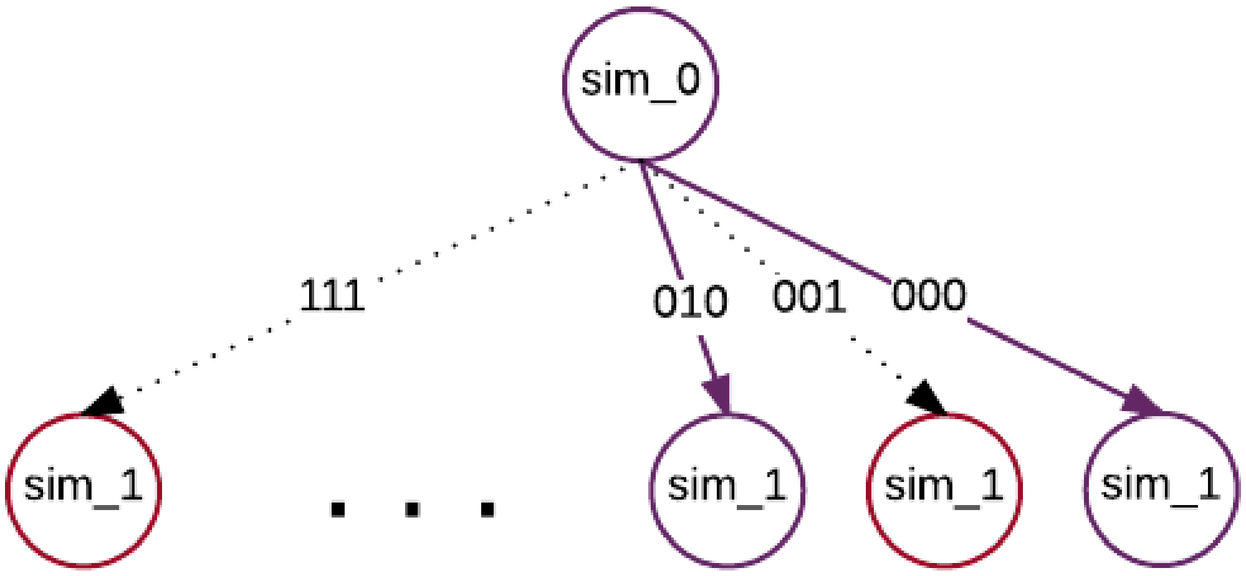
For, N mystery gates, the local search space is, 2^N

8 mGates → local search size: 256

Consider a sim trace of length, K

Then, the Global search space is, 2^{NK} = *state explosion!*

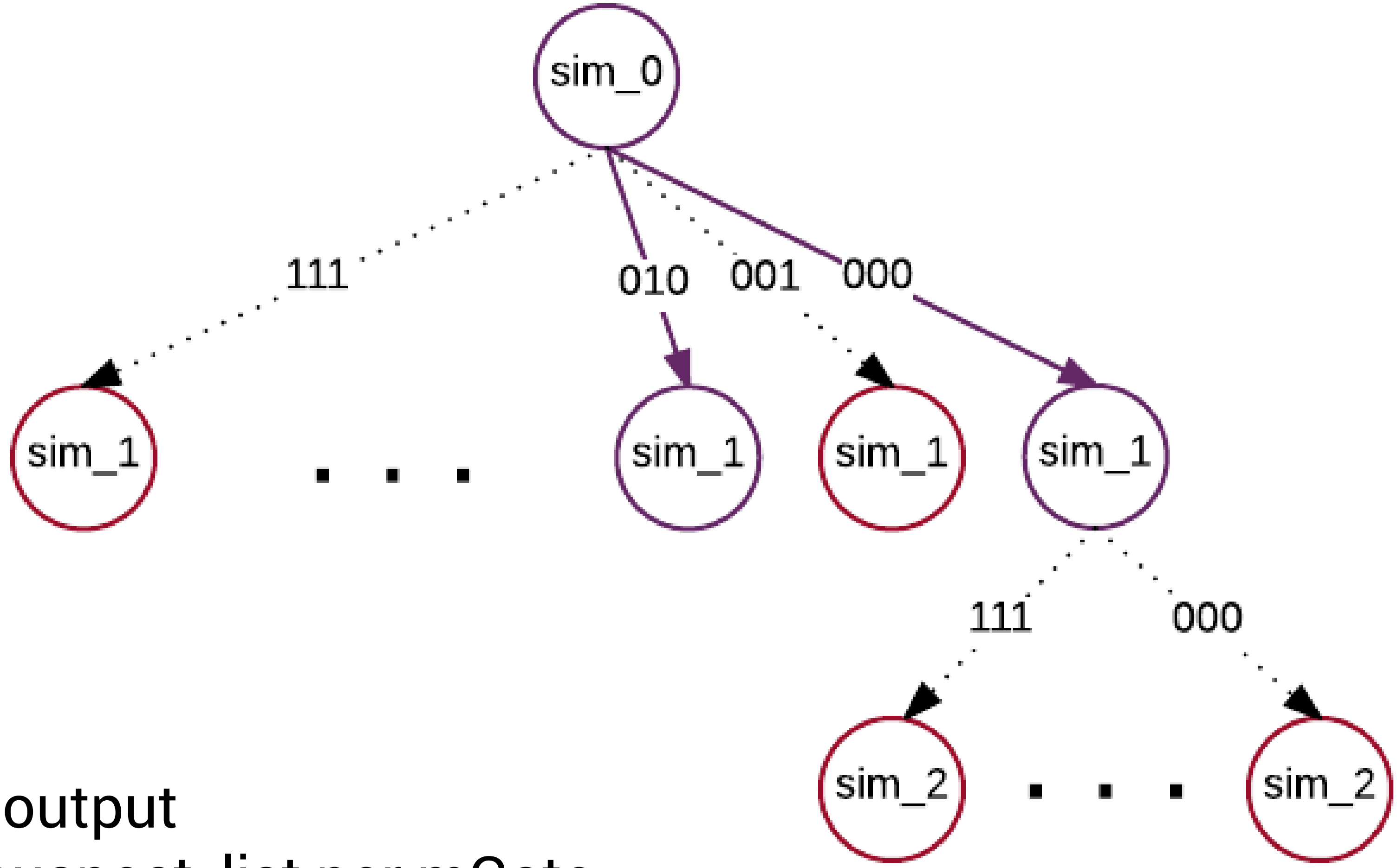
Fucntional **SAT**isfiability



Step In

Say, we have 3 mGates.

→Local Search : $2^3 = 8$



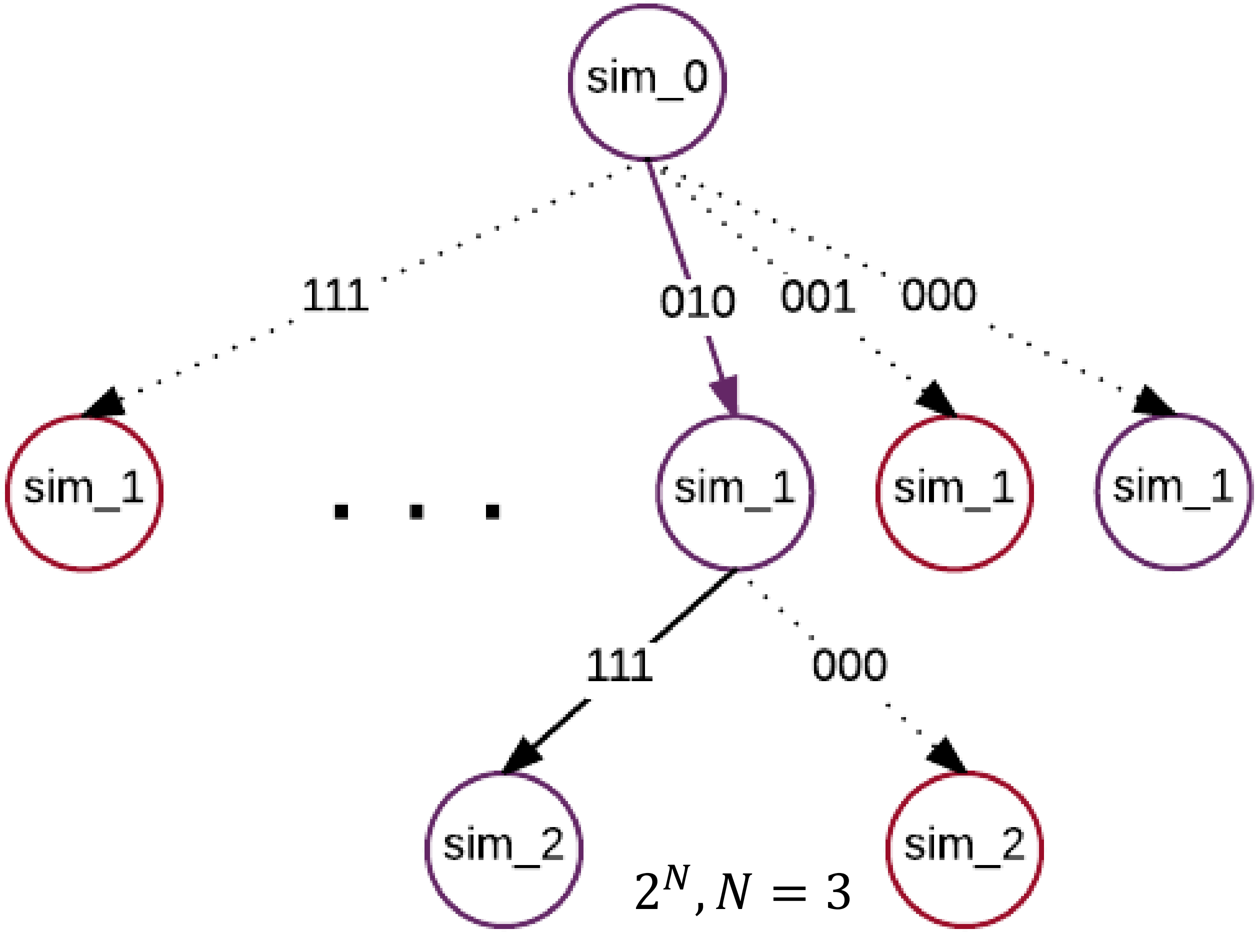
Good Step

- Valid circuit output
- Non-empty suspect_list per mGate

Bad Step

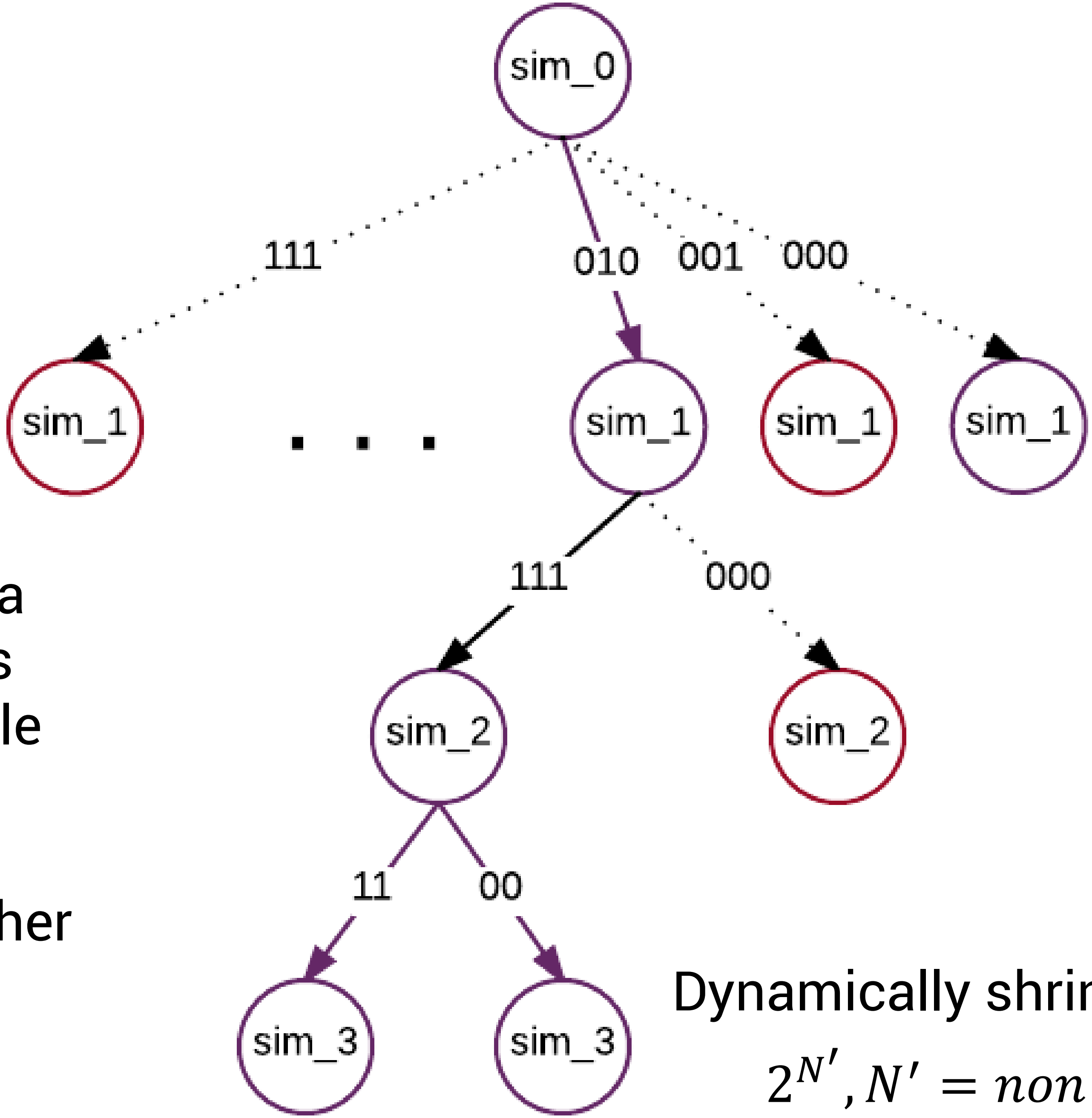
- Invalid circuit output
- Failed to converge,

  i.e, at least one mGate has an empty suspect_list



$2^N, N = 3$

Proceed with local Search



Unit Clause

The suspect_list for a particular mGate has convereged, i.e. single suspect

Implies that any further search **has to have** mGate = suspect

Dynamically shrinking local search,

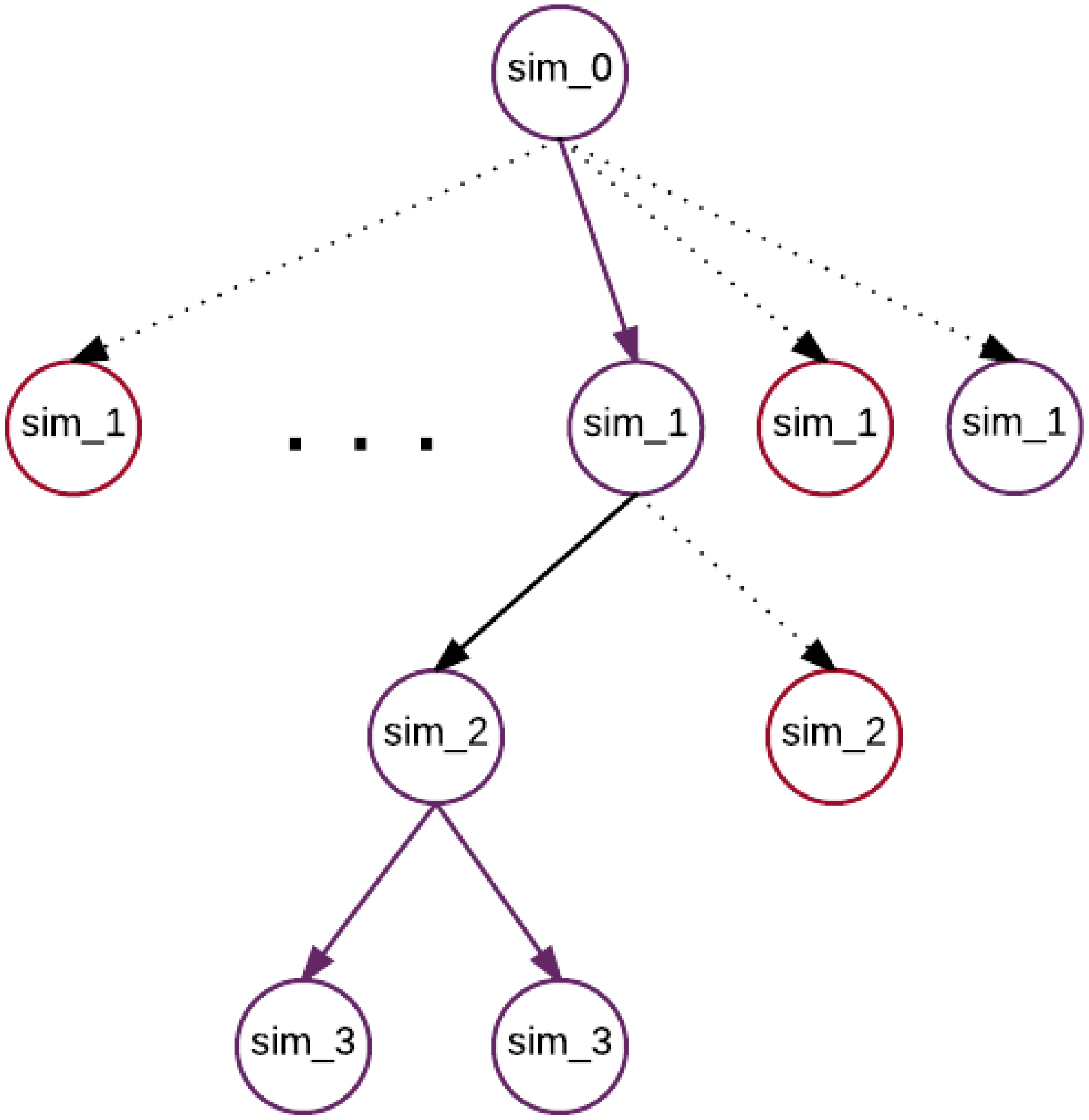
$$2^{N'}, N' = non - unit\ clause\ mGates$$

Parallelization

Shared Memory: OpenMP

```
#pragma omp task
```

Exploratory Decomposition
- Recursive call
- Non-deterministic work size
- On-the fly generation of work

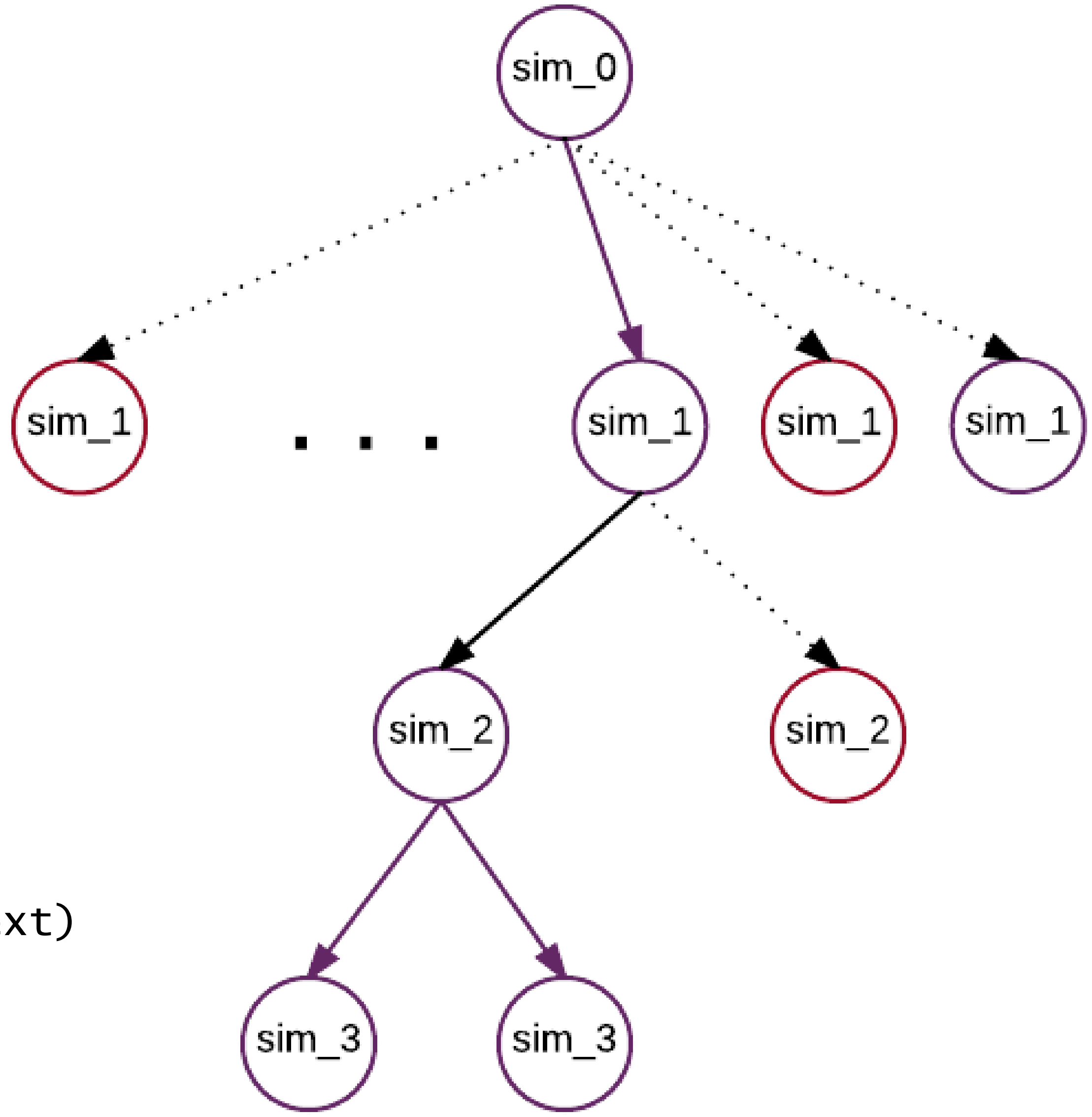


Parallelization

Dynamic Programming

- Each level of search is a single Step
  i.e one input vector
- Sim Object, cloned at every node and passed
  to the next function call (task)
- Each node (task):

```
foreach (faultvector in localSearch)
        sim_next = sim.clone()
        introduce fault vector
        Step-In(sim_next)
        if (Good_Step)
                #pragma omp task
                recursive function call(sim_next)
```



No barriers/taskwait
No roll-backs
No critical sections*

Performance

Constraints:

- Test circuits from ISCAS-89 benchmark
- Random simulation, K = 128
- Random obfuscation of gates of type, ≥2-input
- Machine: rlogin.cs.vt.edu

Goal: Maximal SAT (all possible solutions), timeout: 10 minutes

mgs_sat : Mystery Gate Solver, with Functional SAT

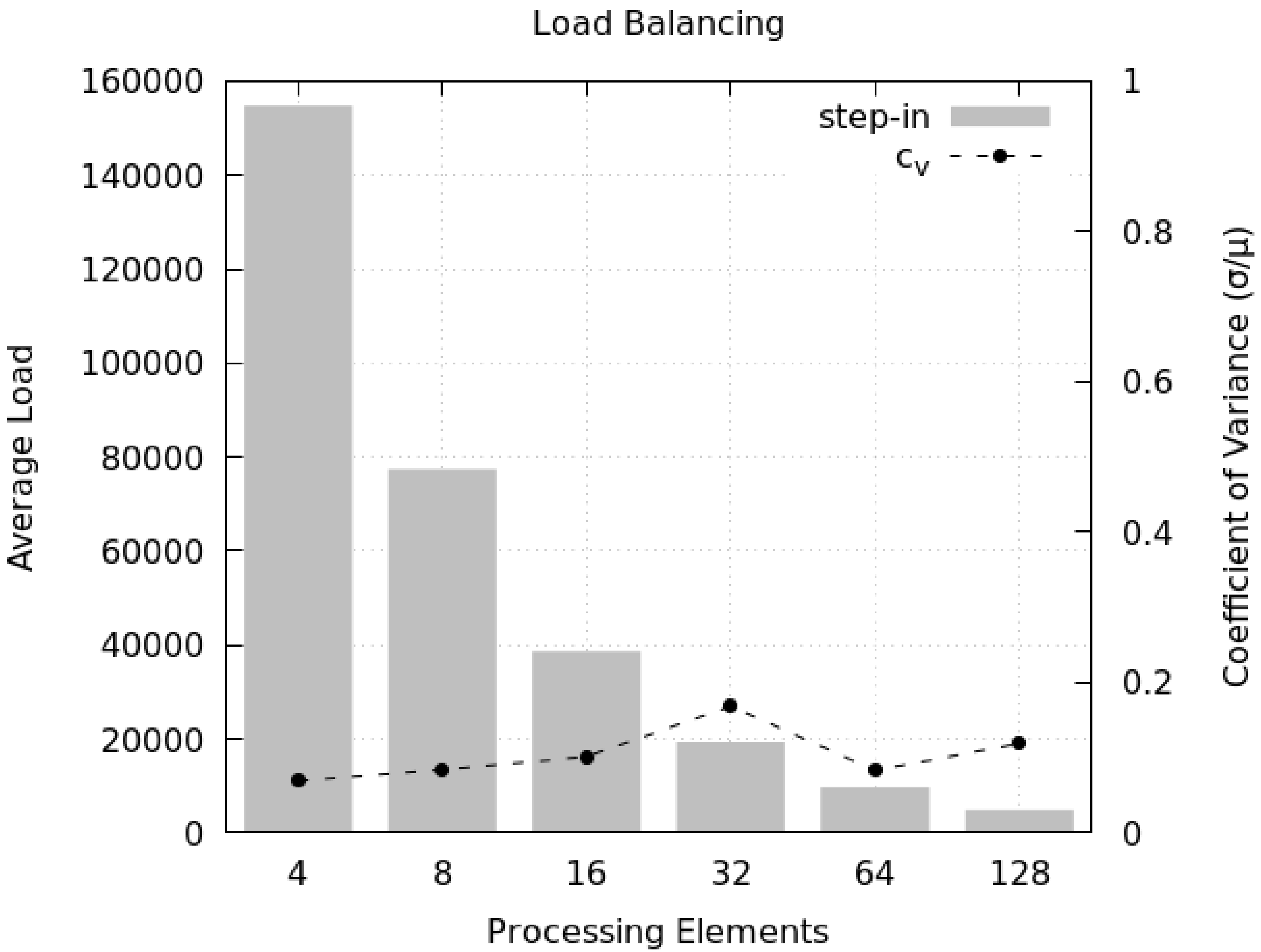
If, K=128, then for 8 mGates, the global search space for mgs_sat is 2^{1024}, which can under-approximated to 10^{300}

Performance: Load Balancing

Circuit = c432 (204 gates)

N = 8

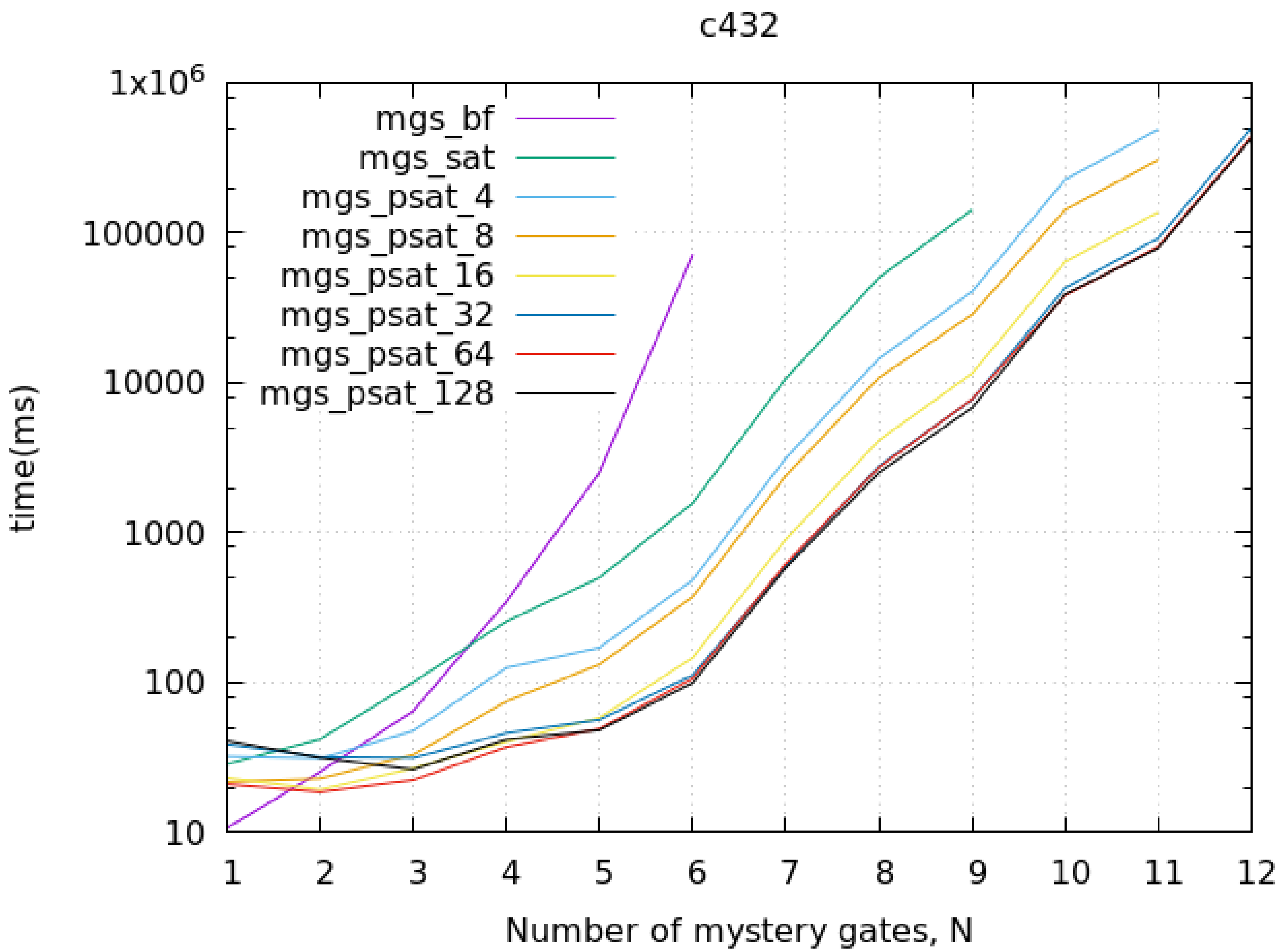
Total Work: 619032



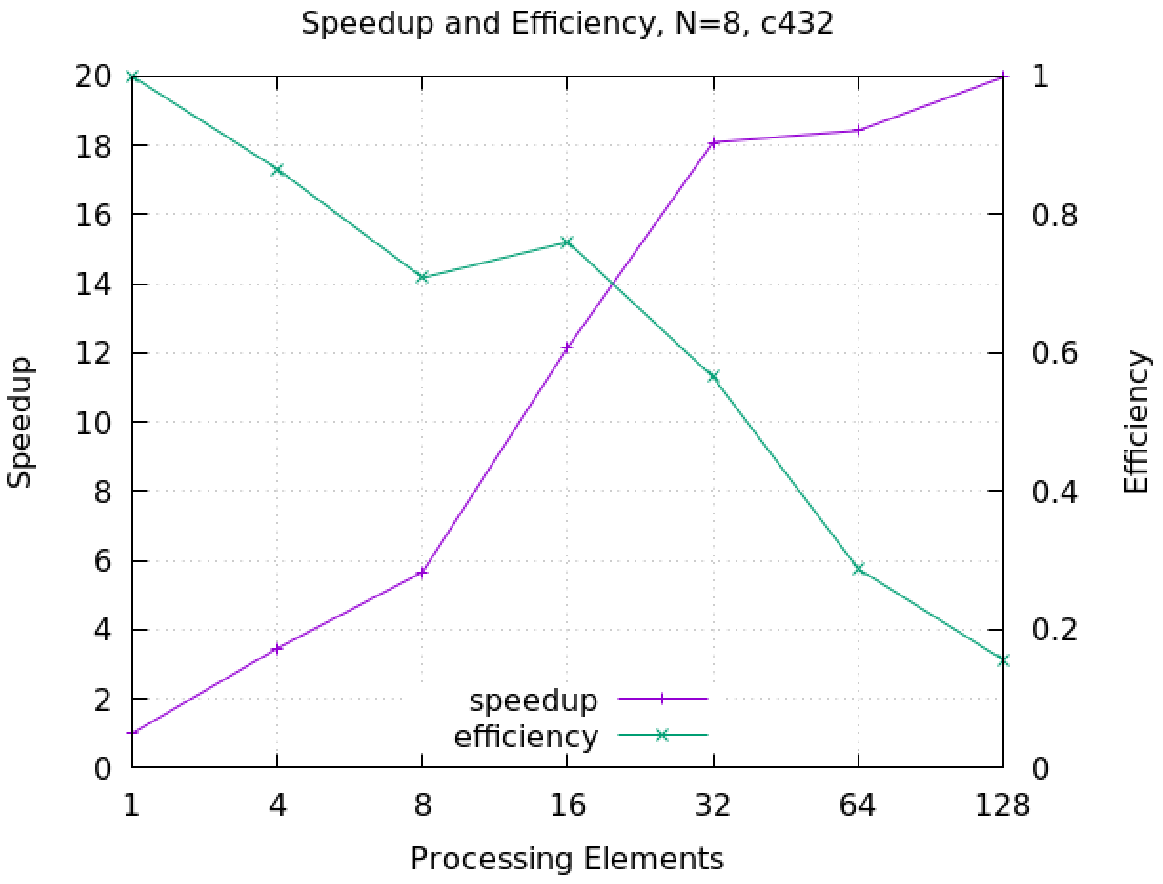
Performance: Time

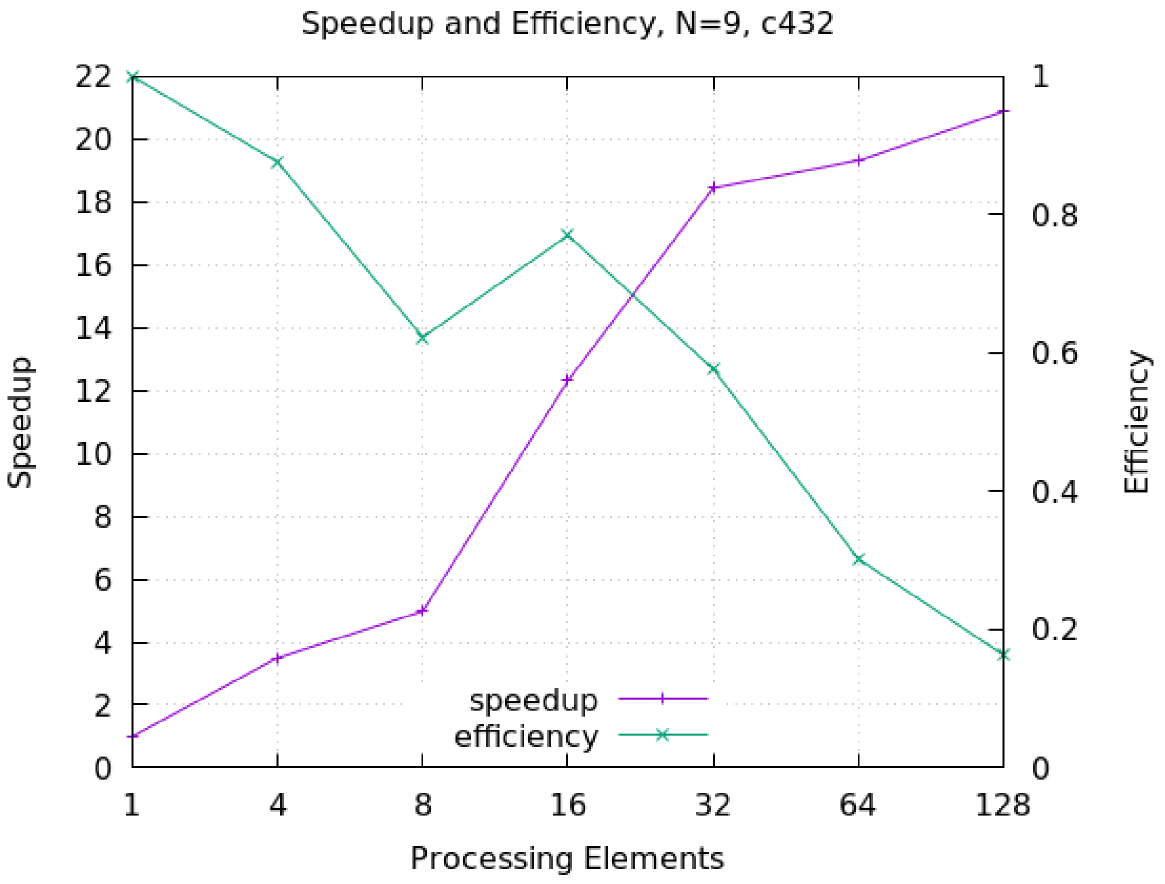
N=8 in c432

Algorithm	Approx. Time
mgs_bf	1 hour
mgs_sat	1 minute
mgs_psat_32	3 seconds



Performance: Speedup/Efficiency

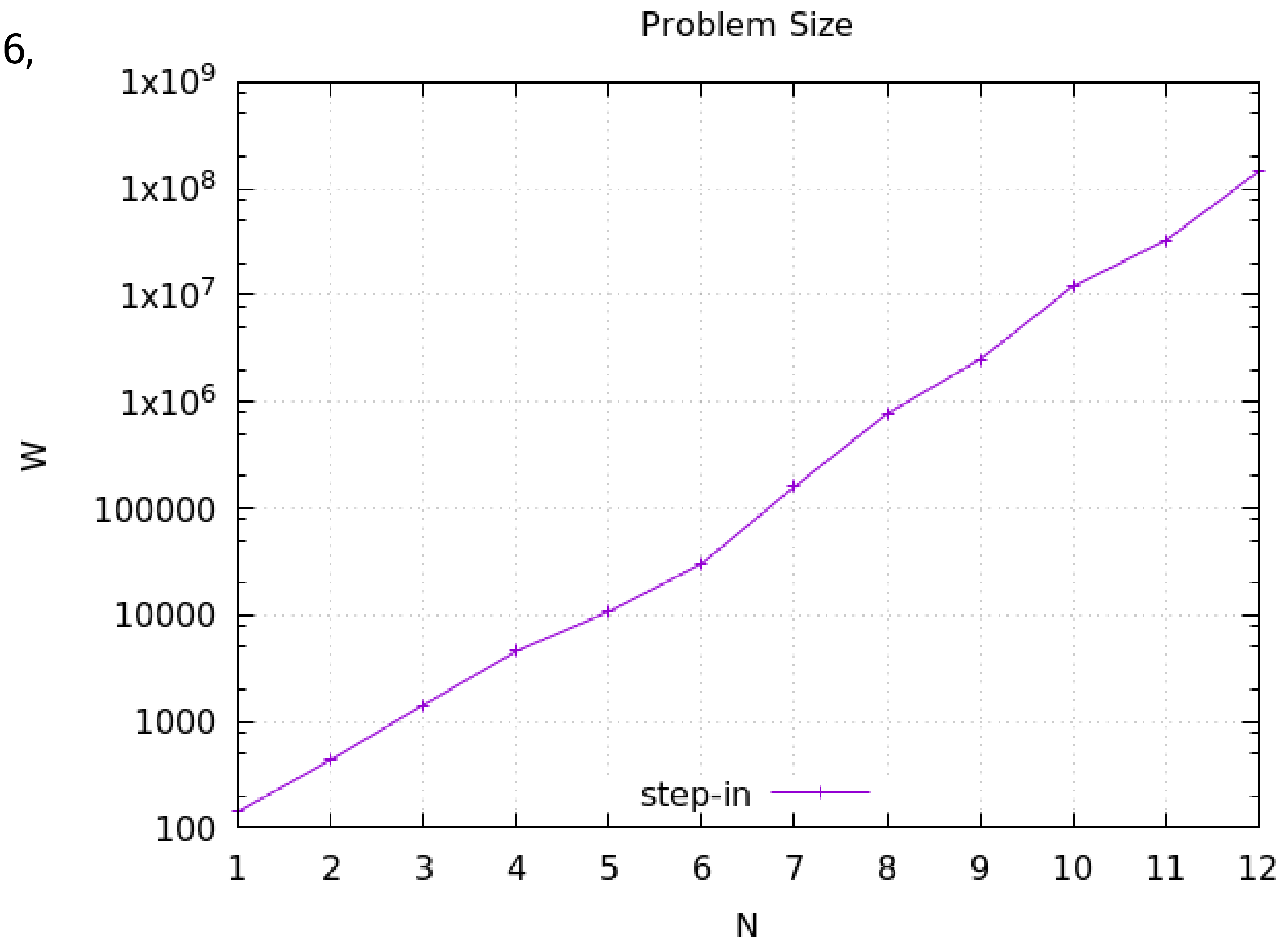




Performance: Scaling

Curve Fitting to find W vs N
Least error for the polynomial of degree 26,

$$W \approx 1.9e{-17}\ (N^{26}) + 1.18e{-15}\ (N^{25}) - 7e{-14}\ (N^{24}) + 1.2e{-12}\ (N^{23}) - 8.72e{-12}\ (N^{22}) + 2.32e{-11}\ (N^{21})$$



Performance: Scaling

Isospeed Scalability

For an increasing workload, the average execution time should remain constant, by increasing the number of processing elements

Timeout: T = 10 minutes

Performance Scaling: Isospeed

maximixe(N), for given p,
within time T

p	N
1	9
4	11
8	11
16	11
32	12
64	12
128	12
256	12

Not Scalable!

minimize(p), for given N,
within time T

N	P
1	1
2	1
3	1
4	1
5	1
6	1
7	2
8	2
9	2
10	2
11	4
12	32
13	>256

Conclusion

mgs_sat: parallel implementation – not scalable, but is a good starting point for future work.

Exploratory decomposition + Dynamic Programming = good template for parallelization

The algorithm can possibly be improved with optimal input simulation (mystery gate input coverage), dominant path simulation, translation to Boolean SAT, etc

That's all Folks!

Any Question ?