

A Performance Study of BDD-Based Model Checking

Bwolen Yang

***Randal E. Bryant, David R. O'Hallaron,
Armin Biere, Olivier Coudert, Geert Janssen
Rajeev K. Ranjan, Fabio Somenzi***

Motivation for Studying Model Checking (MC)

MC is an important part of formal verification

- digital circuits and other finite state systems

BDD is an enabling technology for MC

Not well studied

Packages are tuned using combinational circuits (CC)

Qualitative differences between CC and MC computations

- CC: build outputs, constant time equivalence checking
MC: build model, many fixed-points to verify the specs
- CC: BDD algorithms are polynomial
MC: key BDD algorithms are exponential

Outline

BDD Overview

Organization of this Study

- participants, benchmarks, evaluation process

Experimental Results

- performance improvements
- characterizations of MC computations

BDD Evaluation Methodology

- evaluation platform
 - » various BDD packages
 - » real workload
- metrics

BDD Overview

BDD

- DAG representation for Boolean functions
- fixed order on Boolean variables

Set Representation

- represent set as Boolean function
 - » an element's value is true \iff it is in the set

Transition Relation Representation

- set of pairs (current to next state transition)
- each state variable is split into two copies:
 - » **current state** and **next state**

BDD Overview (Cont'd)

BDD Algorithms

- dynamic programming
- sub-problems (**operations**)
 - » recursively apply Shannon decomposition
- memoization: **computed cache**

Garbage Collection

- recycle unreachable (**dead**) nodes

Dynamic Variable Reordering

- BDD graph size depends on the variable order
- sifting based
 - » nodes in adjacent levels are swapped

Organization of this Study: Participants

Armin Biere: ABCD

Carnegie Mellon / Universität Karlsruhe

Olivier Coudert: TiGeR

Synopsys / Monterey Design Systems

Geert Janssen: EHV

Eindhoven University of Technology

Rajeev K. Ranjan: CAL

Synopsys

Fabio Somenzi: CUDD

University of Colorado

Bwolen Yang: PBF

Carnegie Mellon

Organization of this Study: Setup

Metrics: **17** statistics

Benchmark: **16** SMV execution traces

- ➔ traces of BDD-calls from verification of
 - » *cache coherence, Tomasulo, phone, reactor, TCAS...*
- ➔ size
 - » 6 million - 10 billion sub-operations
 - » 1 - 600 MB of memory

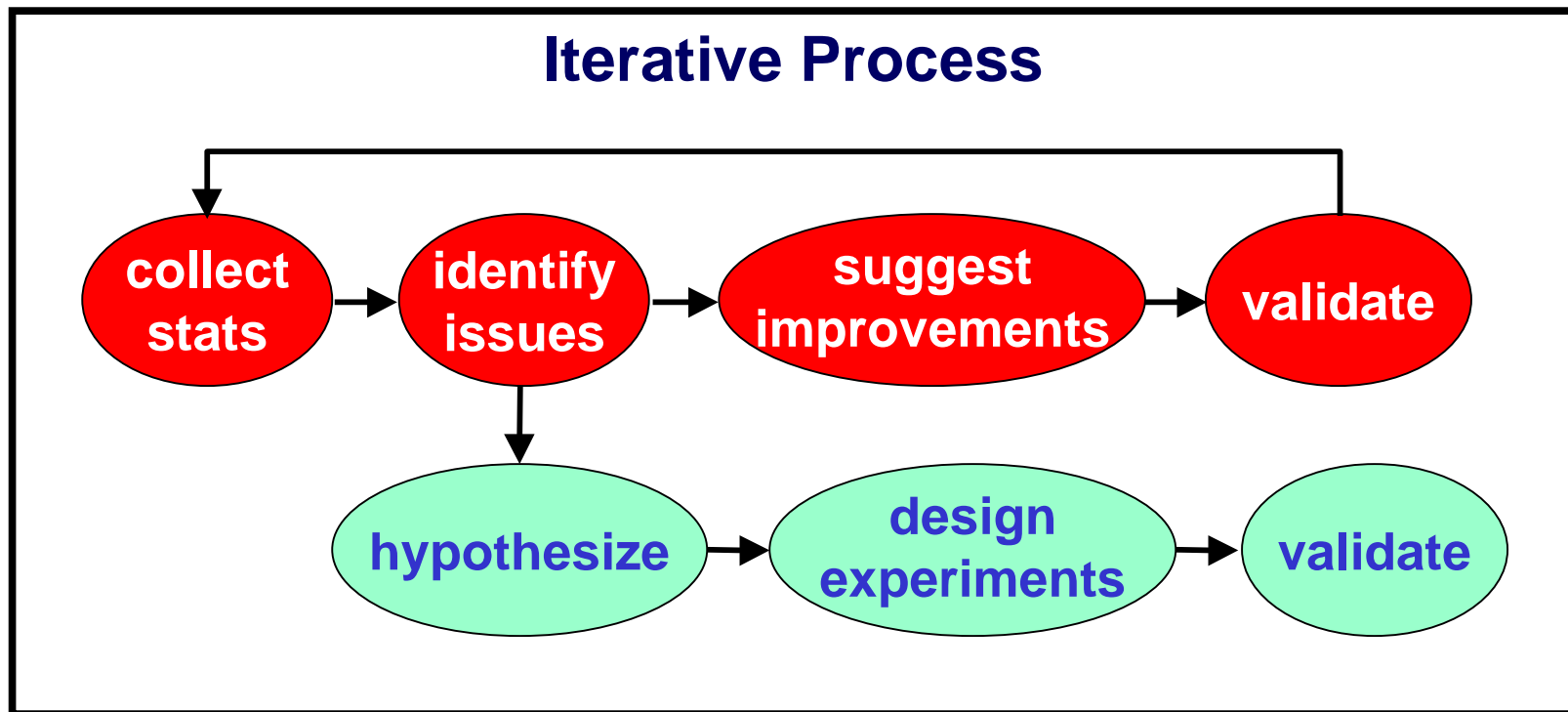
Evaluation platform: *trace driver*

- ➔ “drives” BDD packages based on execution trace

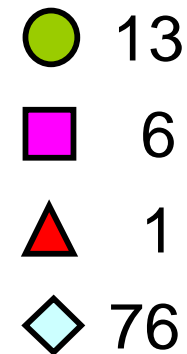
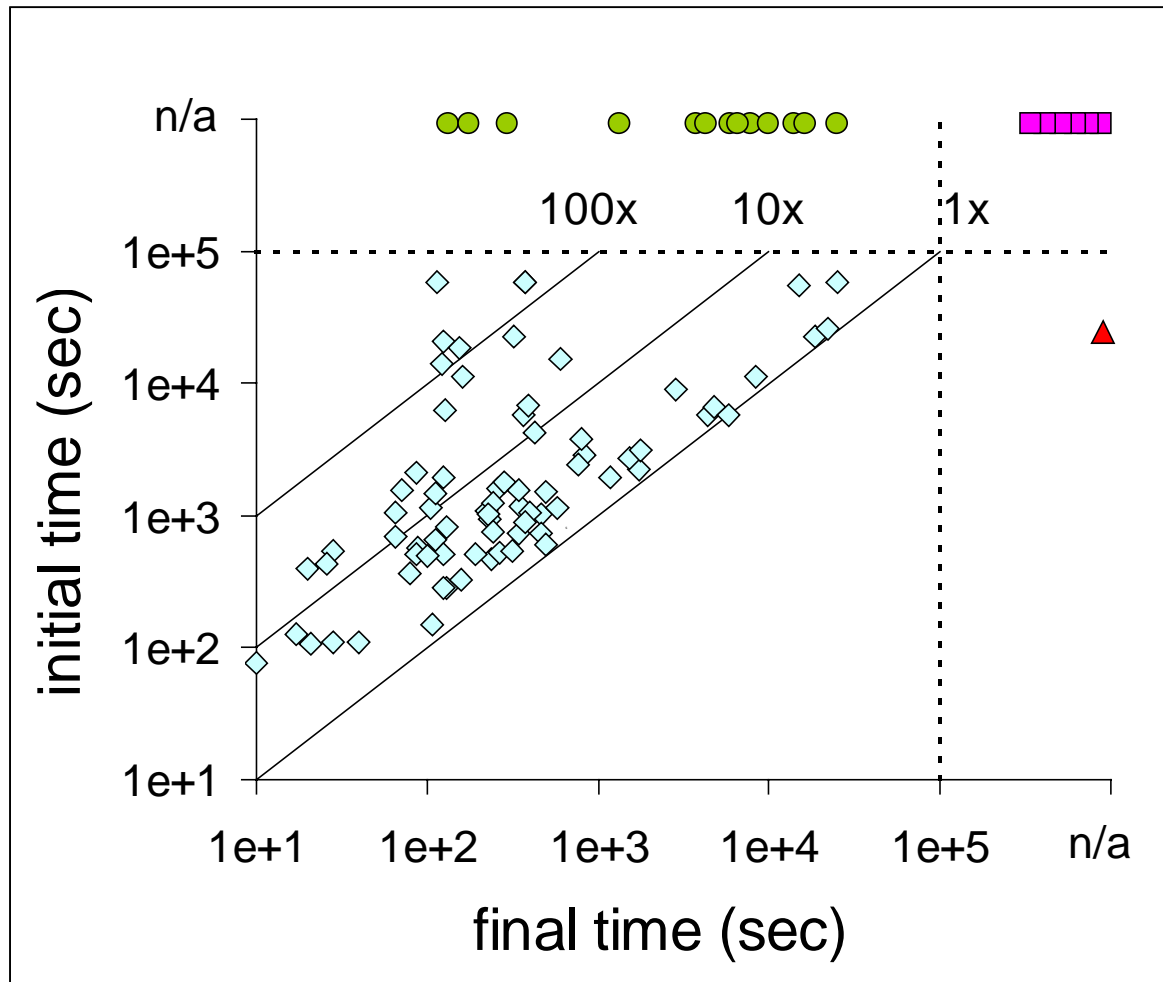
Organization of this Study: Evaluation Process

Phase 1: **no** dynamic variable reordering

Phase 2: **with** dynamic variable reordering



Phase 1 Results: Initial / Final



speedups	
> 100:	6
10 - 100:	16
5 - 10:	11
2 - 5:	28

Conclusion: collaborative efforts have led to significant performance improvements

Phase 1:

Hypotheses / Experiments

Computed Cache

- effects of computed cache size
- amounts of repeated sub-problems across time

Garbage Collection

- reachable / unreachable

Complement Edge Representation

- work
- space

Memory Locality for Breadth-First Algorithms

Phase 1: ***Hypotheses / Experiments (Cont'd)***

For Comparison

ISCAS85 combinational circuits (> 5 sec, < 1GB)

- c2670, c3540
- 13-bit, 14-bit multipliers based on c6288

Metrics depends only on the **trace and **BDD** algorithms**

- machine-independent
- implementation-independent

Computed Cache: Repeated Sub-problems Across Time

Source of Speedup

- increase computed cache size

Possible Cause

- many repeated sub-problems are far apart in time

Validation

- study the number of repeated sub-problems across user issued operations (**top-level operations**).

Hypothesis: Top-Level Sharing

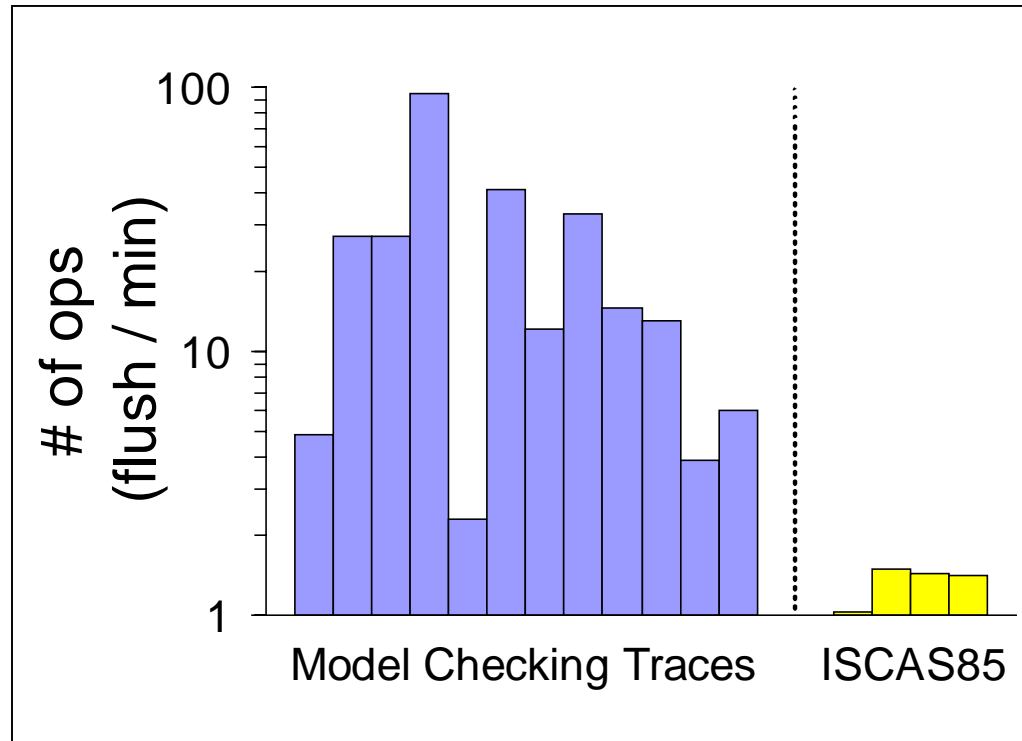
Hypothesis

MC computations have a large number of repeated sub-problems across the top-level operations.

Experiment

- ➔ measure the minimum number of operations with GC disabled and complete cache.
- ➔ compare this with the same setup, but cache is flushed between top-level operations.

Results on Top-Level Sharing



flush: cache flushed between top-level operations

Conclusion: large cache is more important for MC

Garbage Collection: Rebirth Rate

Source of Speedup

- reduce GC frequency

Possible Cause

- many dead nodes become reachable again (**rebirth**)
 - » GC is delayed till the number of dead nodes reaches a threshold
 - » dead nodes are reborn when they are part of the result of new sub-problems

Hypothesis: Rebirth Rate

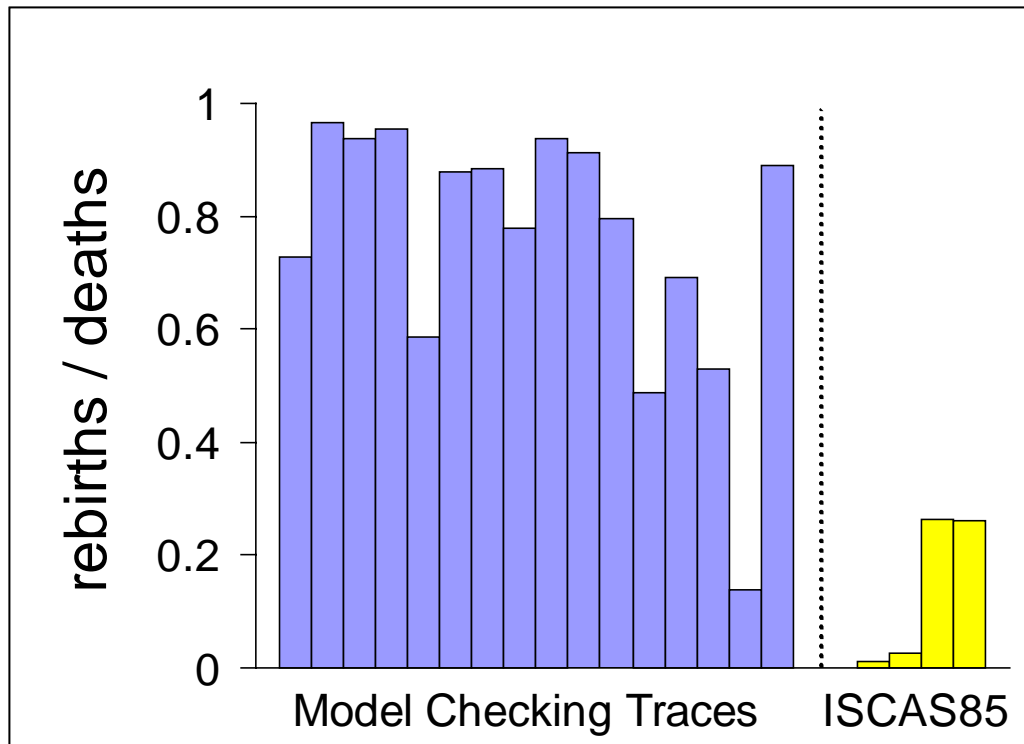
Hypothesis

MC computations have very high rebirth rate.

Experiment

measure the number of deaths and the number of rebirths

Results on Rebirth Rate



Conclusions

- delay garbage collection
- triggering GC should not base only on # of dead nodes
- delay updating reference counts

BF BDD Construction

On MC traces, **breadth-first** based BDD construction has **no** demonstrated advantage over traditional **depth-first** based techniques.

Two packages (CAL and PBF) are BF based.

BF BDD Construction Overview

Level-by-Level Access

- operations on same level (variable) are processed together
- one queue per level

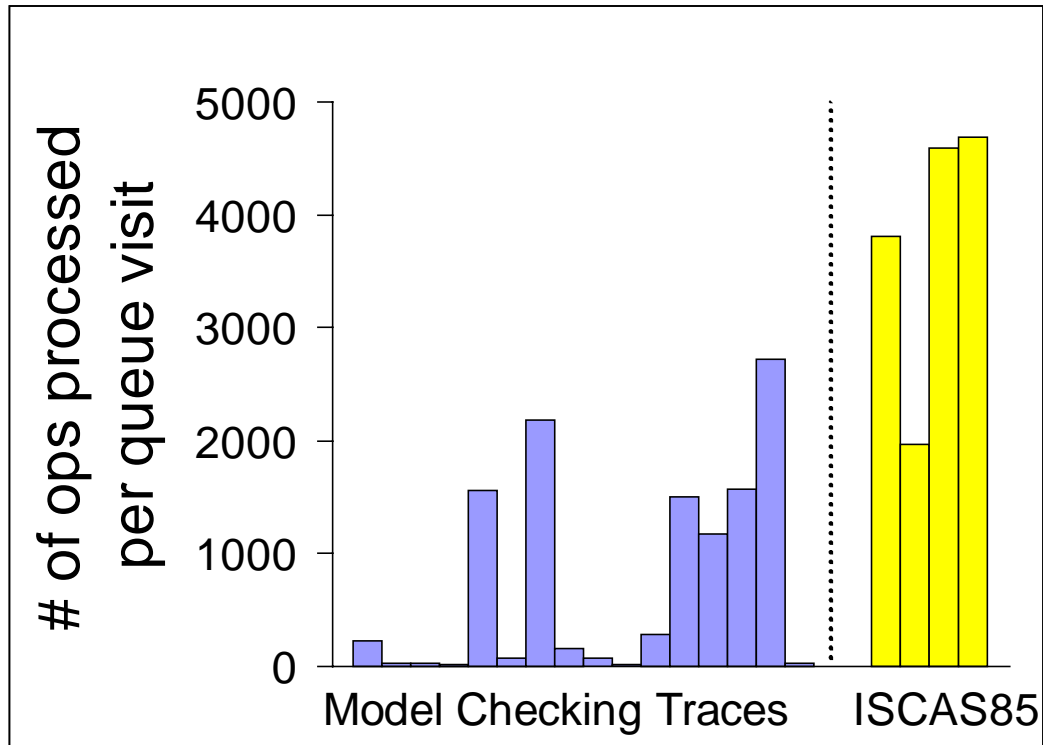
Locality

group nodes of the same level together in memory

Good memory locality due to BF ==>

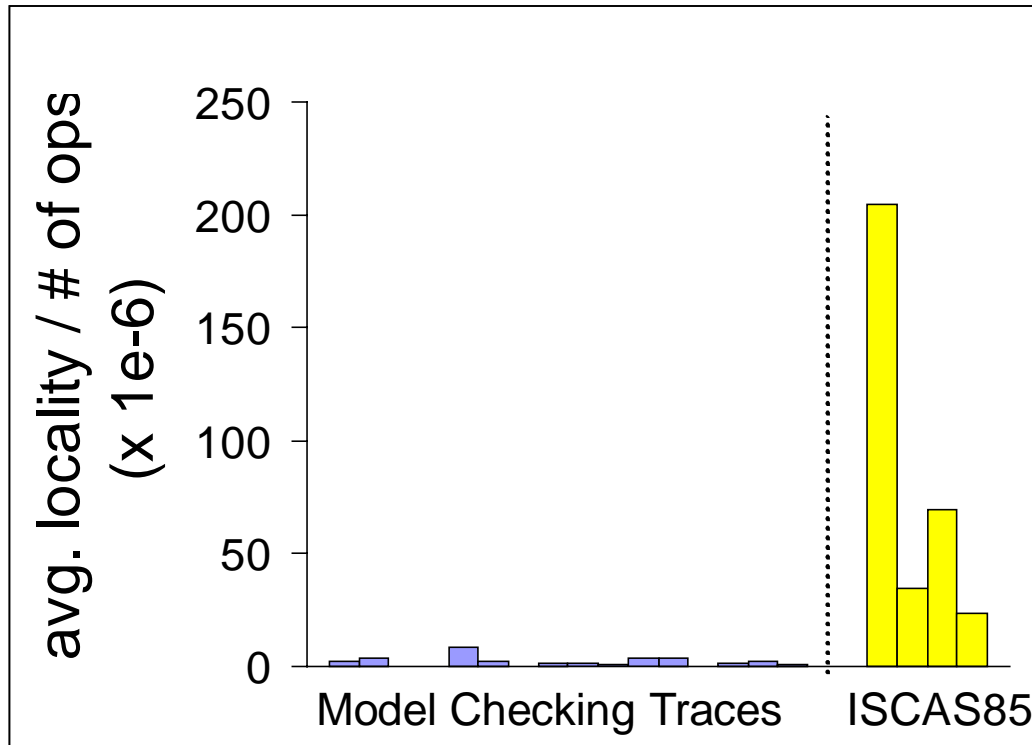
of ops processed per queue visit must be high

Average BF Locality



Conclusion: MC traces generally have less BF locality

Average BF Locality / Work



Conclusion: For comparable BF locality,
MC computations do much more work.

Phase 1:

Some Issues / Open Questions

Memory Management

- ➔ space-time tradeoff
 - » computed cache size / GC frequency
- ➔ resource awareness
 - » available physical memory, memory limit, page fault rate

Top-Level Sharing

- ➔ possibly the main cause for
 - » strong cache dependency
 - » high rebirth rate
- ➔ better understanding may lead to
 - » better memory management
 - » higher level algorithms to exploit the pattern

Phase 2: Dynamic Variable Reordering

BDD Packages Used

- CAL, CUDD, EHV, TiGeR
- improvements from phase 1 incorporated

Why is Variable Reordering Hard to Study

Time-space tradeoff

- how much time to spent to reduce graph sizes

Chaotic behavior

e.g., small changes to triggering / termination criteria
can have significant performance impact

Resource intensive

- reordering is expensive
- space of possible orderings is combinatorial

Different variable order ==> different computation

e.g., many “*don't-care* space” optimization algorithms

Phase 2: Experiments

Quality of Variable Order Generated

Variable Grouping Heuristic

- keep strongly related variables adjacent

Reorder Transition Relation

- BDDs for the transition relation are used repeatedly

Effects of Initial Variable Order

- **with** and **without** variable reordering

Only CUDD is used

Variable Grouping Heuristic: Group Current / Next Variables

Current / Next State Variables

- for transition relation, state variable is split into two:
 - » current state and next state

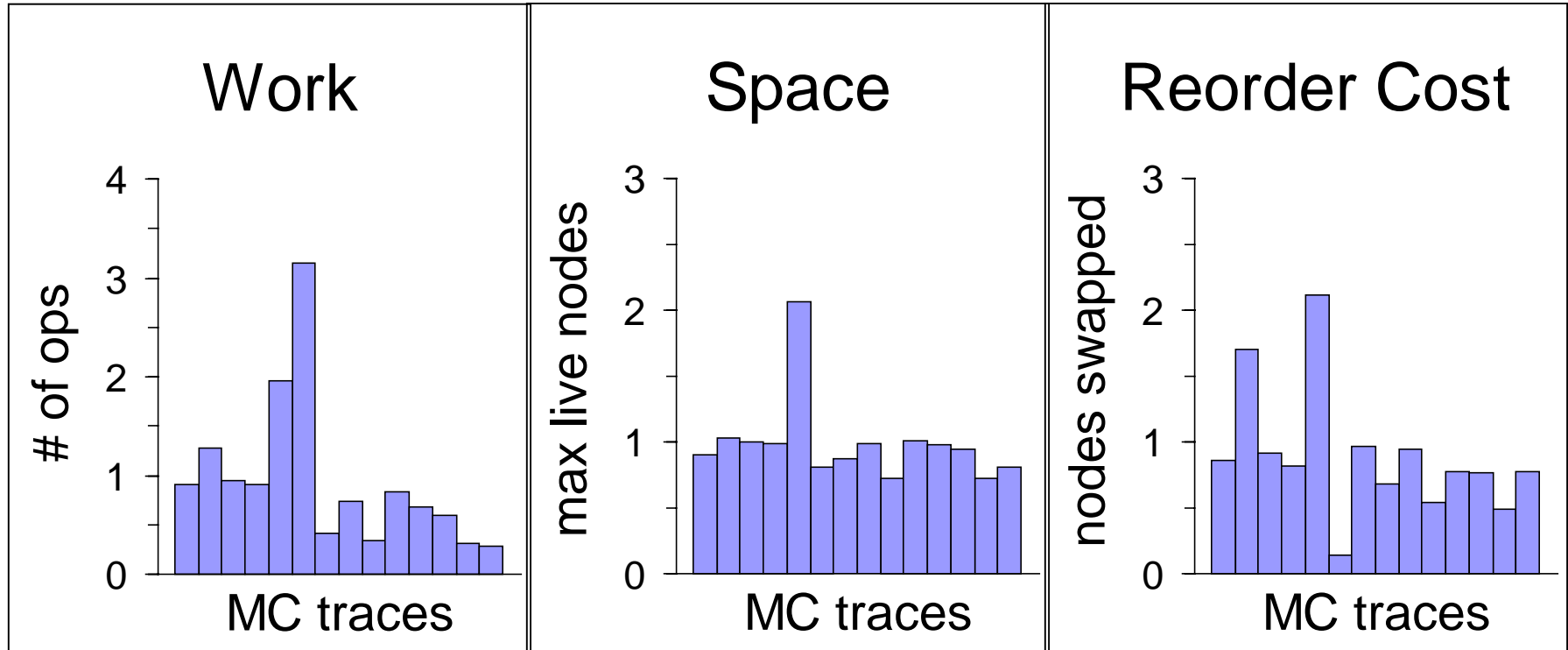
Hypothesis

Grouping the corresponding current- and next-state variables is a good heuristic.

Experiment

- for both **with** and **without** grouping, measure
 - » **work** (# of operations)
 - » **space** (max # of live BDD nodes)
 - » **reorder cost** (# of nodes swapped with their children)

Results on Grouping Current / Next Variables



All results are normalized against no variable grouping.

Conclusion: grouping is generally effective

Effects of Initial Variable Order: Experimental Setup

For each trace,

- ➔ find a good variable ordering O
- ➔ perturb O to generate new variable orderings
 - » fraction of variables perturbed
 - » distance moved
- ➔ measure the effects of these new orderings with and without dynamic variable reordering

Effects of Initial Variable Order: Perturbation Algorithm

Perturbation Parameters (p , d)

- p : probability that a variable will be perturbed
- d : perturbation distance

Properties

- in average, p fraction of variables is perturbed
- max distance moved is $2d$
- ($p = 1$, $d = \text{infinity}$) \implies completely random variable order

For each perturbation level (p , d)

generate a number (**sample size**) of variable orders

Effects of Initial Variable Order: Parameters

Parameter Values

- **p**: (0.1, 0.2, ..., 1.0)
- **d**: (10, 20, ..., 100, infinity)
- **sample size**: 10

==> for each trace,

- **1100** orderings
- **2200** runs (w/ and w/o dynamic reordering)

Effects of Initial Variable Order: Smallest Test Case

Base Case (best ordering)

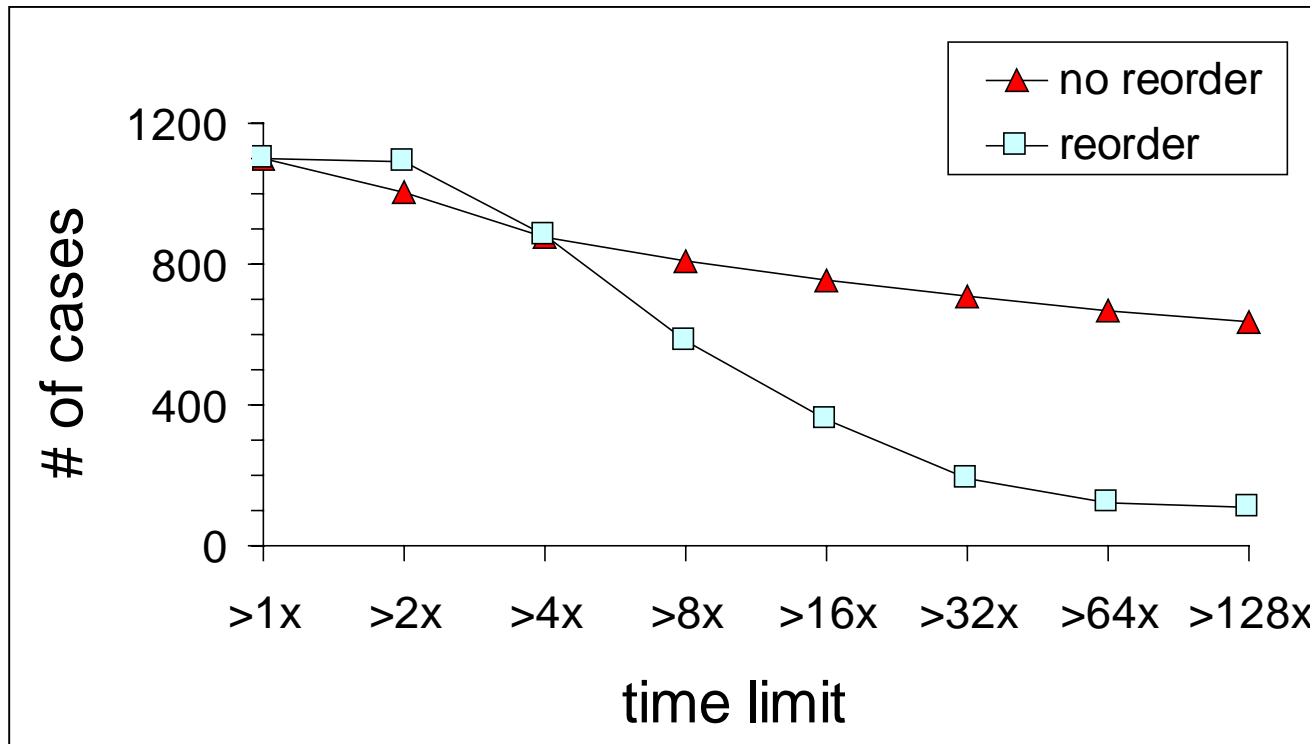
- time: **13** sec
- memory: **127** MB

Resource Limits on Generated Orders

- time: **128x** base case
- memory: **500** MB

Effects of Initial Variable Order: Result

of unfinished cases



At 128x/500MB limit, “no reorder” finished **33%**,
“reorder” finished **90%**.

Conclusion: dynamic reordering is effective

Phase 2: ***Some Issues / Open Questions***

Computed Cache Flushing

→ cost

Effects of Initial Variable Order

→ determine sample size

Need New Better Experimental Design

BDD Evaluation Methodology

Trace-Driven Evaluation Platform

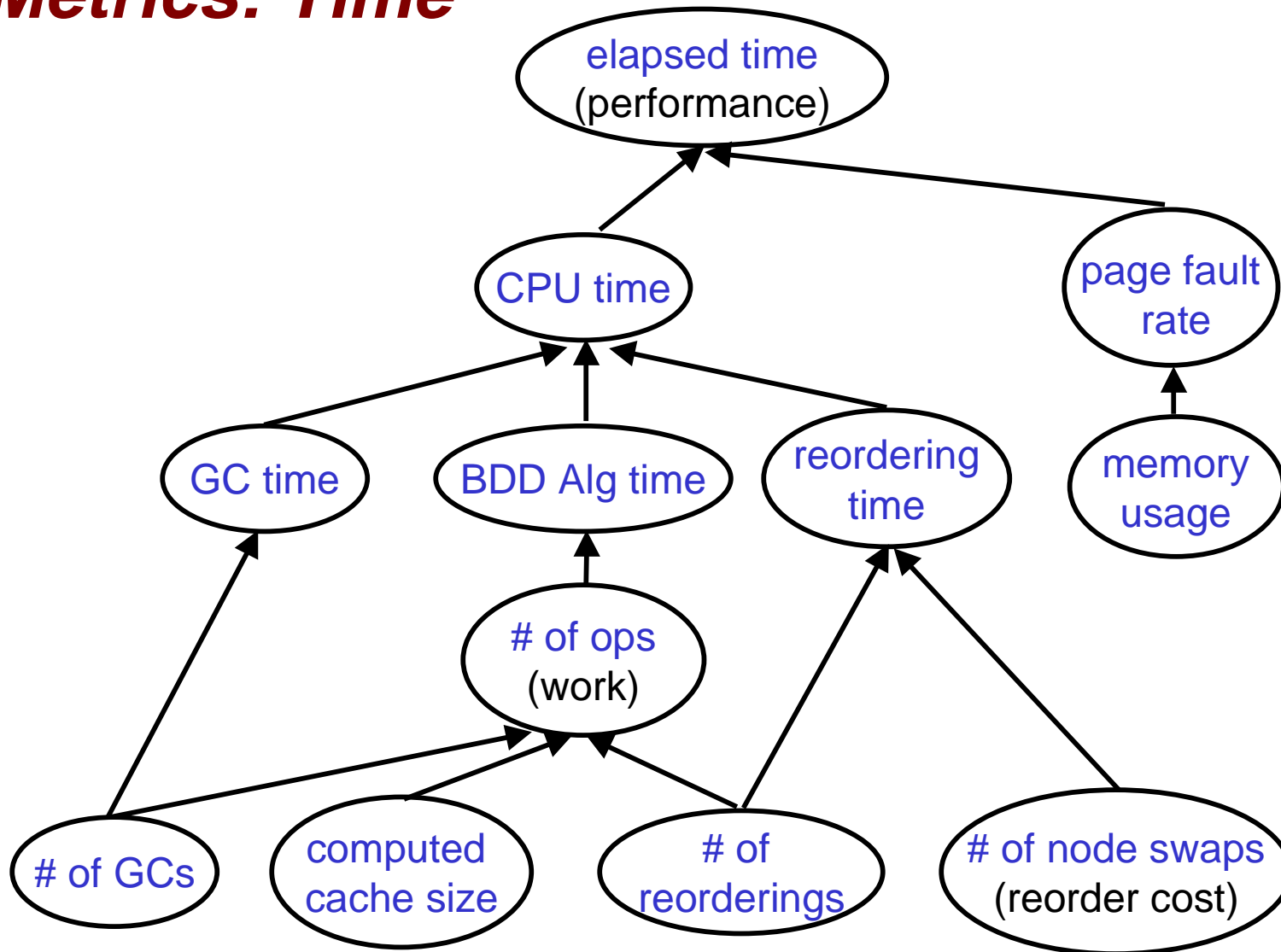
- real workload (BDD-call traces)
- study various BDD packages
- focus on key (expensive) operations

Evaluation Metrics

- more rigorous quantitative analysis

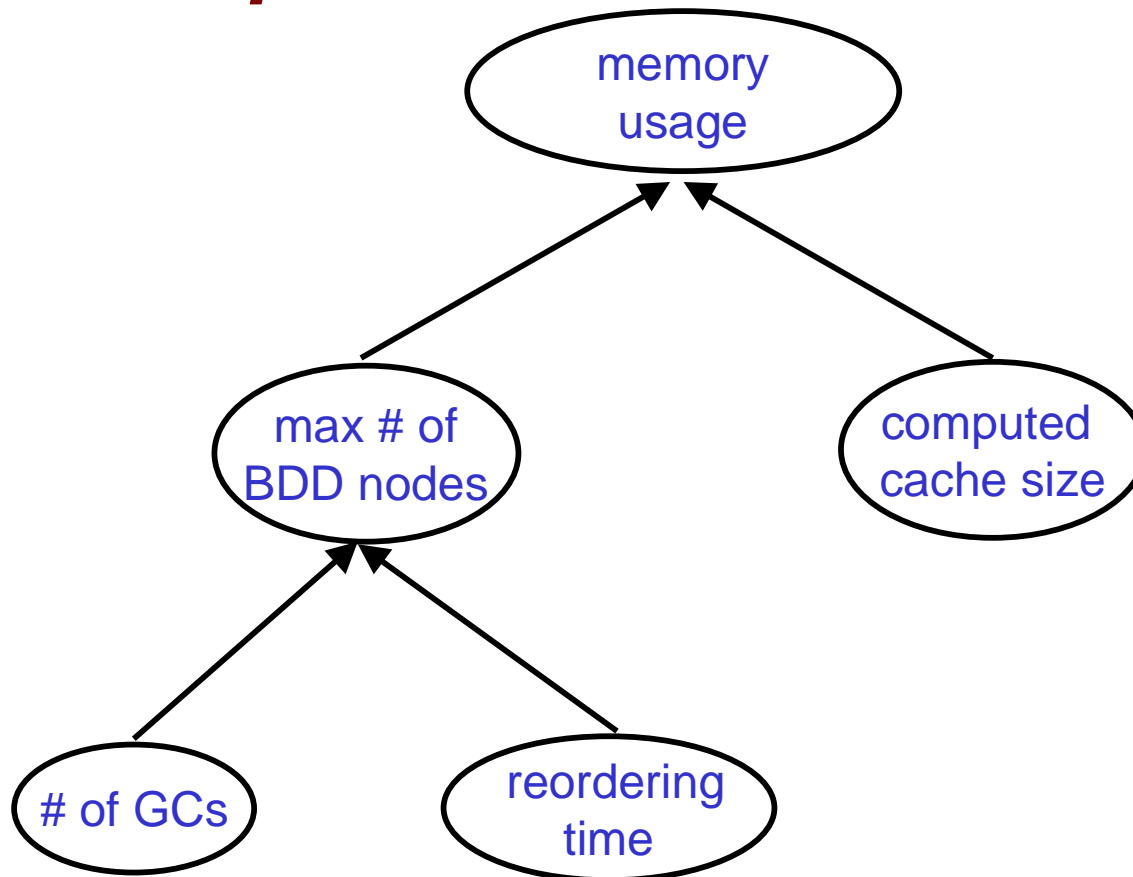
BDD Evaluation Methodology

Metrics: Time



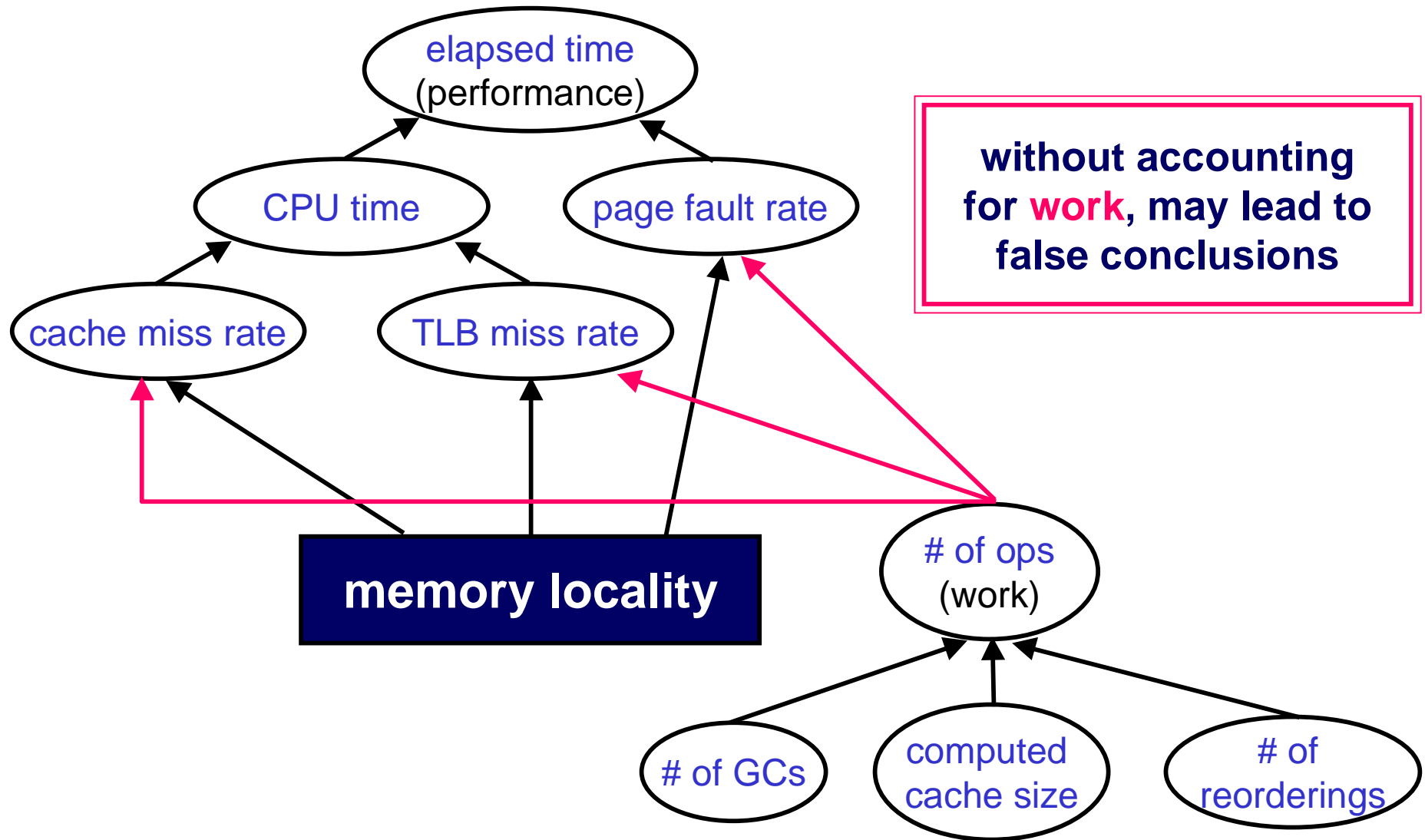
BDD Evaluation Methodology

Metrics: Space



Importance of Better Metrics

Example: Memory Locality



Summary

Collaboration + Evaluation Methodology

- ➔ significant performance improvements
 - » up to 2 orders of magnitude
- ➔ characterization of MC computation
 - » computed cache size
 - » garbage collection frequency
 - » effects of complement edge
 - » BF locality
 - » reordering heuristic
 - current / next state variable grouping
 - » effects of reordering the transition relation
 - » effects of initial variable orderings
- ➔ other general results (not mentioned in this talk)
- ➔ issues and open questions for future research

Conclusions

Rigorous quantitative analysis can lead to:

- ➔ dramatic performance improvements
- ➔ better understanding of computational characteristics

Adopt the evaluation methodology by:

- ➔ building more benchmark traces
 - » for IP issues, BDD-call traces are hard to understand
- ➔ using / improving the proposed metrics for future evaluation

**For data and BDD traces used in this study,
<http://www.cs.cmu.edu/~bwolen/fmcd98/>**