Model Checking with Multi-Threaded IC3 Portfolios

Sagar Chaki, Derrick Karimi

January 19, 2016

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0003195

Overview

IC3/PDR recent algorithm for hardware model checking

- Showing lot of promise
- But most focus on sequential implementations

We parallelized it by having several IC3 copies sharing lemmas

- Showed speedup, but runtime is unpredictable
  - Sensitive to “lemma-discovery order” between copies

Investigated runtime distribution of portfolios of parallel IC3 solvers

- Leveraged observation that runtime of each parallel IC3 follows the “Weibull” probability distribution
- Used statistical analysis to compute good portfolio sizes

Outline

Sequential IC3

Parallel IC3 – Three Variants

- Synchronous, Asynchronous, Proof-Checking

Runtime Variability and Weibull Distribution

Portfolio and Statistical Analysis

Experimental Results

Sequential IC3

Solves the reachability problem in finite state machines

Input : Problem (I, T, S)

- I = set of initial states
- T = transition relation
- S = set of safe states

Output : $\top$ if $\neg S$ is unreachable from I via T, $\bot$ otherwise

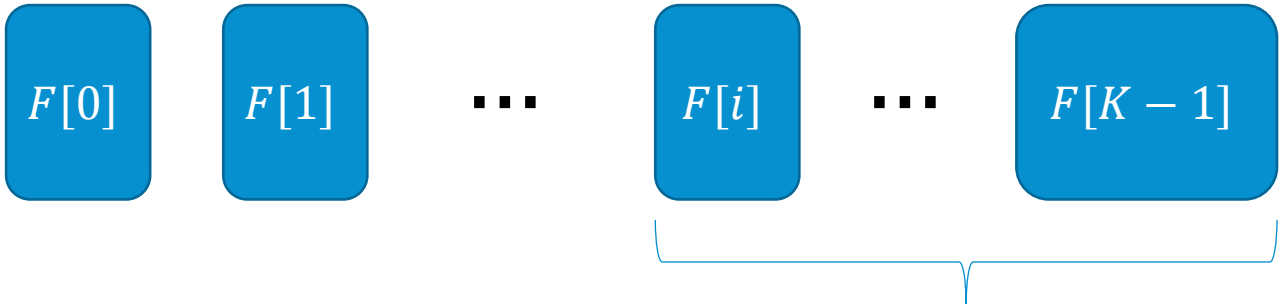
- $T^{\leq i}(I) =$ set of states reachable from I via T in at most i steps

Symbolic: Everything encoded as clauses/minterms over propositional variables

Sequential IC3

$Variables$:
(I, T, S) – $input\ problem$
$F[0..K-1] = array\ of\ frames \quad F[i] = \ set\ of\ clauses \quad K = size\ of\ F$
$bug = boolean\ flag\ to\ indicate\ if\ a\ bug\ has\ been\ found$

Key Invariant



$F[i] = \emptyset \Rightarrow f(i)$ is an inductive invariant

$$\bigcup_{j=i}^{K-1} F[j] = overapprox.\ of\ set\ of\ states\ reachable\ from\ I\ in\ i\ steps\ or\ less$$

$$Formally, \bigcup_{j=i}^{K-1} F[j] = f(i) \supseteq T^{\leq i}(I)$$

One SAT Solver per frame
$f(i)$ added as clauses to solver for $F[i]$
Responsible for all queries involving $f(i)$

Sequential IC3

$Variables$:
(I, T, S) – $input\ problem$
$F[0..K-1] =\ array\ of\ frames \quad F[i] =\ set\ of\ clauses \quad K = size\ of\ F$
$\bigcup_{j=i}^{K-1} F[j] = f(i) \supseteq T^{\leq i}(I)$
$bug = boolean\ flag\ to\ indicate\ if\ a\ bug\ has\ been\ found$

```
bool IC3() {
 check if ¬S reachable in 0 or 1 steps;
 K := 3; F[0] := I; F[1] := F[2] := ∅; bug :=⊥;
 while(⊤)
  strengthen();
  if(bug) return ⊥;
  propagate();
  if(∃i ∈ [1, K − 2]. F[i] = ∅) return ⊤;
  F[K] := ∅; K := K + 1;
}
```

Corresponds to reference implementation we used:
https://github.com/arbrad/IC3ref

Sequential IC3

$Variables$:
(I, T, S) – $input\ problem$
$F[0..K-1] =\ array\ of\ frames \quad F[i] =\ set\ of\ clauses \quad K = size\ of\ F$
$\bigcup_{j=i}^{K-1} F[j] = f(i) \supseteq T^{\leq i}(I)$
$bug = boolean\ flag\ to\ indicate\ if\ a\ bug\ has\ been\ found$

```
bool IC3() {
 check if ¬S reachable in 0 or 1 steps;
 K := 3; F[0] := I; F[1] := F[2] := ∅; bug :=⊥;
 while(⊤)
  strengthen();
  if(bug) return ⊥;
  propagate();
  if(∃i ∈ [1, K − 2]. F[i] = ∅) return ⊤;
  F[K] := ∅; K := K + 1;
}
```

strengthen() = adds new clauses to $F[0..K-1]$ till it finds either:

(i) a real CEX => sets $bug = \top$ and returns;

(ii) $f(K-2) \wedge \neg S = \bot$ => returns.

Uses SAT solver heavily.

Sequential IC3

$Variables$:
(I, T, S) – $input\ problem$
$F[0..K-1] =\ array\ of\ frames \quad F[i] =\ set\ of\ clauses \quad K = size\ of\ F$
$\bigcup_{j=i}^{K-1} F[j] = f(i) \supseteq T^{\leq i}(I)$
$bug = boolean\ flag\ to\ indicate\ if\ a\ bug\ has\ been\ found$

```
bool IC3() {
 check if ¬S reachable in 0 or 1 steps;
 K := 3; F[0] := I; F[1] := F[2] := ∅; bug :=⊥;
 while(⊤)
  strengthen();
  if(bug) return ⊥;
  propagate();
  if(∃i ∈ [1, K − 2]. F[i] = ∅) return ⊤;
  F[K] := ∅; K := K + 1;
}
```

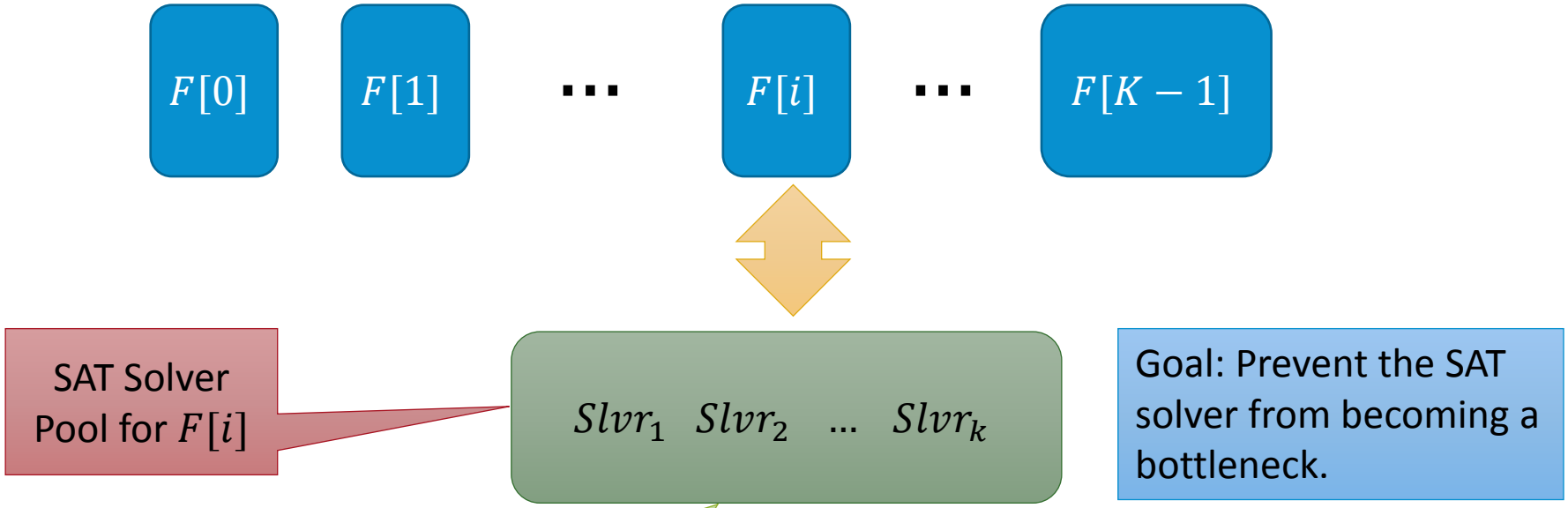
$propagate() = \forall i \in [0, K-2]$ pushes inductive clauses in $F[i]$ to $F[i+1]$;

clause $\alpha \in F[i]$ is inductive if:

$$f(i) \wedge T \Rightarrow \alpha'$$

Uses SAT solver heavily.

SAT Solver Pool



SAT Solver Pool API:

- An IC3 requests a SAT solver.
- If one is available, it is removed from the pool and given to the IC3.
- The IC3 uses the SAT solver and then returns it.
- It is added back to the pool.
- New lemmas learned by this solver are added to everyone else.
- If a solver is currently lent out, the new lemmas are added to it when it is returned.

Parallel IC3: IC3Sync

$n = \#of\ parallel\ IC3\ solvers, each\ using\ its\ own\ frames$
$\forall i = [1,n].F_i[0..K-1]\ K = size\ of\ each\ F_i$
$bug = boolean\ flag\ to\ indicate\ if\ a\ bug\ has\ been\ found$

$bool\ IC3Sync()\ \{$
$check\ if\ \neg S\ reachable\ in\ 0\ or\ 1\ steps;$
$\forall i \in [1,n].F_i[0] \coloneqq I; F_i[1] \coloneqq F_i[2] \coloneqq \emptyset;$
$K \coloneqq 3; bug \coloneqq \bot;$
$while(\top)$
$\{ strengthen_1(\ \ ); propagate_1(\ \ );$
$\| \cdots \|$
$strengthen_n(\ \ ); propogate_n(\ \ );\ \}$
$if(bug)\ return\ \bot;$
$if\left(\exists i \in [1,K-2].\forall j \in [1,n].F_j[i] = \emptyset\right)\ return\ \top;$
$\forall i \in [1,n].F_i[K] \coloneqq \emptyset;$
$K \coloneqq K+1;$
$\}$

(a) Each copy has its own frames.
(b) Use common SAT solver pool to share information.
(c) Runs asynchronously most of the time.
(d) Only synchronization is for the termination check.

Synchronous Parallel Execution – Terminates only when all components terminate

Proof of correctness in paper

Parallel IC3: IC3ASync

$n = \#of\ parallel\ IC3\ solvers, each\ using\ its\ own\ frames$
$\forall i = [1,n].F_i[0..K-1], K_i = size\ of\ F_i$
$bug = boolean\ flag\ to\ indicate\ if\ a\ bug\ has\ been\ found$

```
bool IC3ASync() {
  check if ¬S reachable in 0 or 1 steps;
  bug :=⊥;
  IC3Copy_1( ) ;⋄ ⋯ ⋄ IC3Copy_n( ) ;
  return bug? ⊥: ⊤;
}
```

```
void IC3Copy_i( ) {
  F_i[0] := I; F_i[1] := F_i[2] := ∅; K_i := 3;
  while(⊤)
    strengthen_i( ) ;
    if(bug) return;
    propogate_i( ) ;
    if(∃i ∈ [1, K_i − 2]. ∀j ∈ [1, n]. F_j[i] = ∅)
      return;
    F_i[K_i] := ∅; K_i := K_i + 1;
}
```

(a) Each copy runs completely asynchronously.
(b) Termination check distributed over multiple copies.
(c) Use common SAT solver pool to share information.

Asynchronous Parallel Execution – Terminates as soon as any one component terminates

Proof of correctness in paper

Parallel IC3: IC3Proof

$n = \#of\ parallel\ IC3\ solvers, each\ using\ its\ own\ frames$
$\forall i = [1, n]. F_i[0..K-1], K_i = size\ of\ F_i$
$bug, safe = boolean\ flag\ to\ indicate\ bug\ or\ safety\ proof$

```
bool IC3Proof() {
 check if ¬S reachable in 0 or 1 steps;
 bug := ⊥; safe := ⊥;
 IC3PrCopy_1( );⋄ ··· ⋄ IC3PrCopy_n( );
 return bug? ⊥: ⊤;
}
```

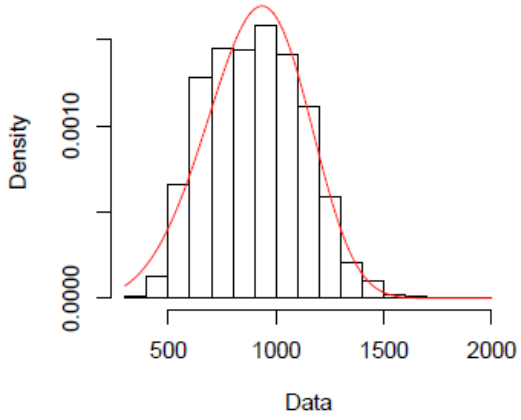
```
void IC3PrCopy_i( ) {
 F_i[0] := I; F_i[1] := F_i[2] := ∅; K_i := 3;
 while(⊤)
  strengthen_i( );
  if(bug) return;
  propagateProof_i( );
  if(safe)return;
  F_i[K_i] := ∅; K_i := K_i + 1;
}
```

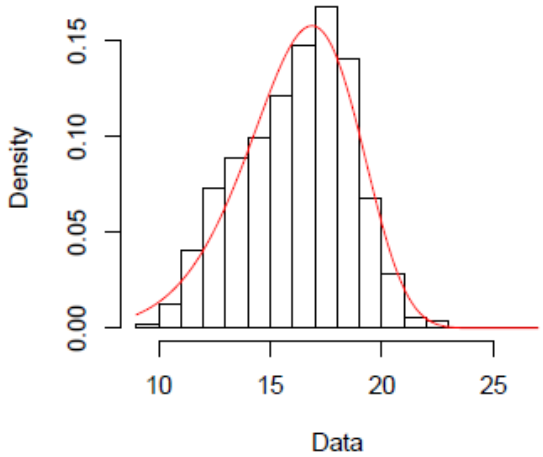
```
void propagateProof_i( ) {
 (a) propagate inductive lemmas
     forward;
 (b) if a frame becomes empty,
     check if the lemmas at that level
     over all the copies form an
     inductive invariant
 (c) if so set flag := ⊤ and return
 (d) otherwise, return after all lemmas
     have been pushed forward
}
```

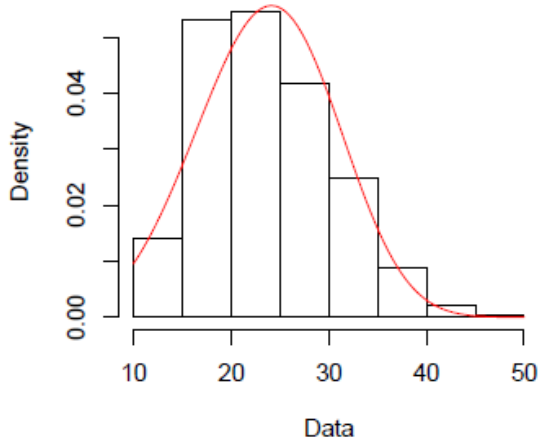
Details and Proof of correctness in paper

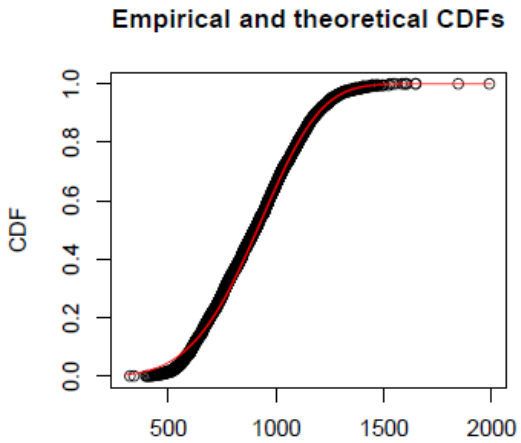
Refer to IC3Sync, IC3Async, and IC3Proof collectively as ParIC3

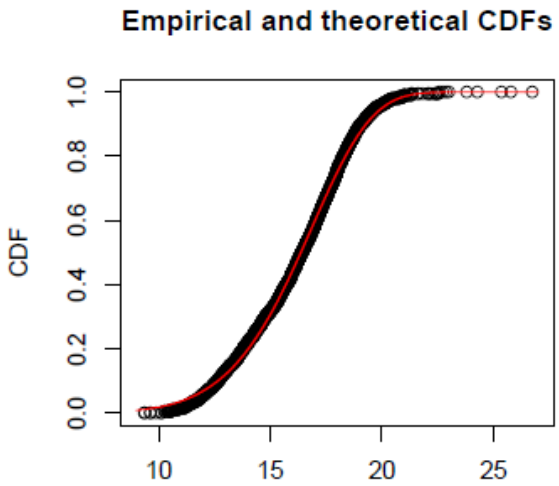
Unpredictability in Runtime of Parallel IC3

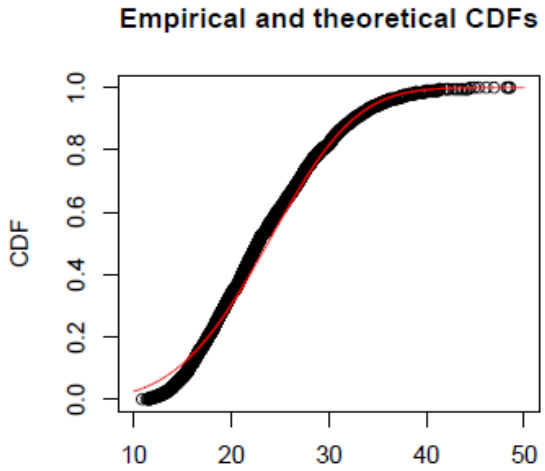












Matches Weibull Distribution = Minimum of iid random variables under Extreme Value Theorem

Intuition: IC3 copies "compete" and the fastest one "wins"

Goodness of Fit to Weibull – IC3Sync

	IC3SYNC (4)			
Example	k	λ	μ, μ^*	σ, σ^*
6s286	4.07	1119	1015,1015	280,274
intel026	2.71	49.0	43.6,44.2	17.3,14.6
6s273	3.80	26.1	23.6,23.6	6.93,6.57
intel057	6.58	16.0	14.9,15.1	2.66,2.11
intel054	7.82	24.3	22.8,23.0	3.46,2.94
6s215	2.38	7.69	6.82,7.03	3.05,2.34
6s216	1.95	35.1	31.1,31.0	16.6,16.9
oski3ub1i	5.98	54.9	50.9,51.4	9.90,7.90
oski3ub3i	5.71	52.4	48.5,48.9	9.84,8.00
oski3ub5i	5.08	66.8	61.4,61.9	13.8,11.6
SAFE	5.00	246	224,224	62.1,60.2
BUG	4.22	43.4	39.7,40.0	10.6,9.37
ALL	4.61	145	131,132	36.4,34.7

(a) Used a cluster of 11 machines, each with 16 cores@2.4GHz and [48,190]GB of RAM.
(b) Selected 10 examples from HWMCC'14 (5 safe, 5 buggy)
(c) Solved each 3000 times
(d) Collected solving time
(e) Extracted Weibull parameters
-- k : shape
-- λ : scale
(f) Compared predicated mean and variance from k and λ with observed from the actual runtimes

Goodness of Fit to Weibull – IC3ASync

	IC3ASYNC (4)			
Example	k	λ	μ, μ^*	σ, σ^*
6s286	4.44	990	902,903	230,220
intel026	3.70	50.2	45.3,46.2	13.6,10.1
6s273	4.11	23.5	21.3,21.4	5.85,5.36
intel057	7.31	17.2	16.1,16.1	2.60,2.46
intel054	8.69	26.1	24.6,24.8	3.38,2.84
6s215	4.71	6.75	6.17,6.21	1.49,1.34
6s216	3.56	27.5	24.8,24.9	7.74,6.97
oski3ub1i	7.02	52.3	48.9,49.2	8.20,6.71
oski3ub3i	5.51	52.2	48.2,48.6	10.1,8.51
oski3ub5i	4.94	67.2	61.6,62.0	14.2,12.4
SAFE	5.65	221	202,202	51.1,48.3
BUG	5.15	41.2	37.9,38.2	8.36,7.20
ALL	5.40	131	120,120	29.7,27.7

(a) Cluster of 11 machines, each with 16 cores@2.4GHz and [48,190]GB of RAM.
(b) Selected 10 examples from HWMCC'14 (5 safe, 5 buggy)
(c) Solved each 3000 times
(d) Collected solving time
(e) Extracted Weibull parameters
-- k : shape
-- λ : scale
(f) Compared predicated mean and variance from k and λ with observed from the actual runtimes

Goodness of Fit to Weibull – IC3Proof

	IC3PROOF (4)			
Example	k	λ	μ, μ^*	σ, σ^*
6s286	4.35	980	892,892	232,228
intel026	3.70	50.1	45.2,46.1	13.6,10.3
6s273	4.17	23.3	21.2,21.3	5.73,5.29
intel057	7.52	17.8	16.7,16.9	2.63,2.07
intel054	9.26	26.1	24.7,24.8	3.20,2.92
6s215	4.72	6.38	5.84,5.90	1.41,1.21
6s216	2.78	28.1	25.0,25.1	9.74,9.05
oski3ub1i	4.78	54.8	50.2,50.8	11.9,9.53
oski3ub3i	5.66	52.2	48.2,48.5	9.87,8.39
oski3ub5i	4.93	66.2	60.7,61.1	14.0,12.1
SAFE	5.80	219	200,200	51.4,49.7
BUG	4.58	41.5	38.0,38.3	9.42,8.07
ALL	5.19	130	119,119	30.4,28.9

(a) Cluster of 11 machines, each with 16 cores@2.4GHz and [48,190]GB of RAM.
(b) Selected 10 examples from HWMCC'14 (5 safe, 5 buggy)
(c) Solved each 3000 times
(d) Collected solving time
(e) Extracted Weibull parameters
-- k : shape
-- λ : scale
(f) Compared predicated mean and variance from k and λ with observed from the actual runtimes

Portfolio of IC3Pars

Run several IC3Pars in parallel

- Completely independent
  - no data sharing among different IC3Pars
  - data shared only between IC3 copies within the same IC3Par
- Stop as soon as one IC3Par completes

Intuition: With a large enough portfolio, we can get lucky

- But how large should the portfolio be?
  - e.g., if we want to beat the average performance of a single IC3Par solver with 0.99999 probability?
  - can statistical analysis provide an answer?
    - Yes, answer=20

Statistical Analysis of ParIC3 Portfolio

Consider portfolio $P_1, \ldots, P_m$ of m ParIC3s working on a problem

Y_i = time taken by the i-th solver $\sim WEI(k, \lambda)$

Let $t^* = E[Y_i]$ = expected solve time by single IC3Par = $\lambda\Gamma(1 + \frac{1}{k})$

$Y = \min(Y_1, \ldots, Y_m)$ = solve time by portfolio $\sim WEI(k, \frac{\lambda}{\frac{1}{m^k}})$

Let $p(m) = P[Y \leq t^*]$ =probability that portfolio does better than expected solving time of a single IC3Par

Result: $p(m) > 1 - e^{-\frac{m}{e^\gamma}}$ where $\gamma = 0.57721$ is the Euler-Mascheroni constant.

Plugging in$1 - e^{-\frac{m}{e^\gamma}} = 0.99999$ we get $m = 20$

- A portfolio of 20 IC3Pars will beat the average IC3Par in a single attempt with probability 0.99999

Results: Parallel PDR Speedup

		IC3SYNC		IC3ASYNC		IC3PROOF		IC3RND	
$\mathcal{B}$	$\lvert\mathcal{B}^*\rvert$	Mean	Max	Mean	Max	Mean	Max	Mean	Max
HWCSAFE	31	1.30	5.61	1.58	5.47	1.60	4.08	1.17	4.64
HWCBUG	14	2.49	18.7	14.3	151	25.1	309	1.07	1.49
TIPSAFE	14	1.28	4.50	2.61	11.1	2.29	12.8	1.37	3.80
TIPBUG	9	2.23	5.35	2.82	7.32	3.50	12.1	1.16	2.17
SAFE	44	1.30	5.61	1.93	11.1	1.83	12.8	1.24	4.64
BUG	23	2.38	18.7	9.58	151	16.3	309	1.11	2.17
ALL	67	1.67	18.7	4.74	151	6.79	309	1.19	4.64

Portfolio of 20 IC3Par solvers. Each IC3Par Solver has 4 IC3 copies. SAT solver pool size = 3. Experiments done on a 128 Core machine running at 2.67GHz and 1TB RAM. IC3Proof performs best overall – looking for inductive invariants intermittently pays off. IC3Rnd = Portfolio of 20 IC3s with a randomized SAT solver. Some speedup but not as good as IC3Par.

Results: Parallel PDR vs IC3Par2010

		IC3SYNC		IC3ASYNC		IC3PROOF			IC3PAR2010	
$\mathcal{B}$	$\lvert\mathcal{B}^*\rvert$	Mean	Max	Mean	Max	Mean	Max	$\lvert\mathcal{B}^+\rvert$	Mean	Max
HWCSAFE	31	1.30	5.61	1.58	5.47	1.60	4.08	20	2.67	14.40
HWCBUG	14	2.49	18.7	14.3	151	25.1	309	15	1.62	3.91
TIPSAFE	14	1.28	4.50	2.61	11.1	2.29	12.8	14	.89	1.82
TIPBUG	9	2.23	5.35	2.82	7.32	3.50	12.1	7	1.32	1.67
SAFE	44	1.30	5.61	1.93	11.1	1.83	12.8	34	1.94	14.40
BUG	23	2.38	18.7	9.58	151	16.3	309	22	1.52	3.91
ALL	67	1.67	18.7	4.74	151	6.79	309	56	1.77	14.40

IC3Par2010 – parallel version of IC3 presented in original (VMCAI'11) IC3 paper
Parallel PDR better for unsafe cases – i.e., better at finding counterexamples
Difference is less clear for safe cases, each better in some cases

Concluding Thoughts

- We have some new ways of parallelizing IC3
  - Modest speedups with portfolios – improves over previous attempts in some cases
  - Statistical analysis gives numeric values to good portfolio size
- Connections with parallelizing other verification tools?
  - we rely on data structures peculiar to IC3
  - monotonicity and invariants maintained by the algorithm
- Can the statistical analysis be done for other portfolios?
  - Our result tied to a specific distribution (Weibull) of runtime
  - Does this hold for parallel SAT/SMT solvers?
  - If not, can we derive similar statistical results?