Parallel Software Model Checking
Sagar Chaki
October 8, 2015

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Project Introduction and Overview

Scalability = fundamental challenge in software model checking (SMC)
  • Model Checking: My 30-year Quest to Overcome the State Explosion Problem, Prof. Edmund Clarke

Most tools are sequential and do not use the abundant CPU cycles
  • SMC is inherently difficult to parallelize
  • SPIN has been parallelized, but is explicit-state

Develop a parallel symbolic software model checking algorithm
  • Target multi-processors and clusters

Parallelize a recently developed SMC algorithm called Generalized Property Directed Reachability (GPDR)
  • Has inherent parallelization opportunities (promising candidate)
  • Being used in several SMC application domains (wide impact)
void main() {
  int x = 0;
  while(x < 10) x++;
  assert (x == 10);
}

\[ c_1: x = 0 \Rightarrow P(x) \]
\[ c_2: P(x) \land x < 10 \land x' = x + 1 \Rightarrow P(x') \]
\[ c_3: P(x) \land x \geq 10 \land x \neq 10 \Rightarrow \text{Error}() \]

Q: Error()

**SMC Problem**

**Constrained HORN-SAT (CHC) Instance**

- **CHC** = Predicates () + Clauses () + Query ()
- **Solution** = Assignment to predicates that satisfies the clauses such that the Query predicate is assigned
- **Claim**: Solution exists for CHC iff main() never violates assertion
- **SMC** for concurrent programs, real-time software, Lustre programs etc. also being reduced to CHC
- **Idea**: parallelize a recently developed algorithm (GPDR) for solving CHC
Intellectual and Scientific Merit

GPDR: Iteratively compute candidate solutions $P_0, P_1, Q_0, Q_1, R_0, R_1$ etc. till a real solution is found, or it is proved that no solution can exist.
GPDR: Iteratively compute candidate solutions $P_0, P_1, Q_0, Q_1, R_0, R_1$ etc. till a real solution is found, or it is proved that no solution can exist.
Parallel PDR with Lemma Sharing

PDR = GPDR with a single negative predicate per clause
  • Used for hardware model checking
  • Also known as IC3

Parallelized a publicly available reference implementation of IC3
  • Several copies of IC3 running in parallel
  • Sharing facts learned about reachable states (lemmas)
  • Three variants: synchronous, asynchronous, proof-checking
  • Evaluated on benchmarks from the Hardware Model Checking Competition 2014
  • Average speed up over 2x, in some cases over 300x
Unpredictability in Runtime of Parallel PDR

Matches Weibull Distribution = Minimum of iid random variables under Extreme Value Theorem. Solvers “compete” and the fastest one “wins”
Portfolio of Parallel PDRs

Parallelization leads to random runtime

- Information from other copies perturbs the SAT solver and alters the search path in unpredictable ways
- Solution: Use a portfolio
  - Run many solvers in parallel
  - Stop as soon as one finds solution
- How big should the portfolio be?
  - Answer: 20 gives you a .99999 probability of hitting the expected runtime of a single solver
  - Derived using statistical analysis and extreme value theory
    - Runtime of portfolio = \( \min \) (runtime of solvers)
    - Minimum on iid random variables converge to Weibull distribution

Paper under submission. Tools publicly available.
Results: Parallel PDR (4)

| \( \mathcal{B} \) | \( |\mathcal{B}^*| \) | IC3SYNC Mean | IC3SYNC Max | IC3ASYNC Mean | IC3ASYNC Max | IC3PROOF Mean | IC3PROOF Max | IC3RND Mean | IC3RND 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        |
Results: GPDR Strategies

Rewrote our implementation of GPDR (called Spacer)

• Re-design and re-implementation
  • improved the original code written by a student
  • new architecture is similar to IC3 allowing to reuse our existing work on parallelizing IC3

• Implemented three different solution strategies
  • Differ in the way priorities queues are populated and cleared
  • Results indicate that strategies are complementary
  • Each performs well on different subset of benchmarks
  • Good idea to run in parallel with “loose” coupling

• Tool is publicly available
Results: Parallel GPDR

Run different strategies on different machines/cores and share inductive invariants and reachable states (partial solutions)

Use restarts to weed out bad strategies

Observed speedups in some cases, approach has potential
  • Insufficient data to draw solid conclusions
Summary

Parallel Software Model Checking

Sagar Chaki (chaki@sei.cmu.edu)
Arie Gurfinkel (arie@sei.cmu.edu)
Derrick Karimi (dhkarimi@sei.cmu.edu)