Error detection using BMC in a parallel environment

Subramanian K. Iyer[1], Jawahar Jain[2], Mukul R. Prasad[2], Debashis Sahoo[3], and Thomas Sidle[2]

[1] University of Texas at Austin, Austin, TX 78712, USA
[2] Fujitsu Labs of America, Sunnyvale, CA 94085, USA
[3] Stanford University, Stanford CA 94305, USA

Abstract. In this paper, we explore a parallelization of BMC based on state space partitioning. The parallelization is accomplished by executing multiple instances of BMC independently from different seed states. These seed states are *deep* states, selected from the reachable states in different partitions. In this scheme, all processors work independently of each other, thus it is suitable for scaling verification to a grid-like network. Our experimental results demonstrate improvement over existing approaches, and show that the method can scale to a large network.

1 Introduction

Satisfiability based Bounded Model Checking (SAT-BMC) [2] approaches are the preferred method for detecting error states that are not very deep. However, these techniques can become quite expensive when many time-frames are required to be analyzed. BDD based approaches are better choices for those "deep cases" where the image BDDs remain moderately small as constructing large BDDs for many image steps can be very expensive. Thus the class of problems which may require many steps of image analysis to detect the error, *but* where BDD sizes grow large, remain an attractive research target.

Our approach is to create a method that can find various candidate deep states which can be *seeds* from which SAT-BMC can be run in *parallel* to explore the adjacent state space. Starting from such potential deep seed states, multiple BMC runs may be able to reach further deep states, and locate errors, which may be out of reach for existing methods.

Generating Seed States: For a few initial steps of reachability, rapid progress can be made using BDDs. To control the size of BDDs using state space analysis we use state-space partitioning [5]. Deep states provided from such local BFS traversals can be used to provide initial seed states to subsequent BMC runs. Since the BDD runtime is directly proportional to the size of the graphs, we further limit the size of partitions using an under-approximation based method on top of partitioned BDDs.

Using Seed States: We augment our ideas of combining Partitioning and BMC by generating multiple instances of BMC and run each such case in parallel on a grid of computers. This idea looks even more attractive when we consider that large computing grids are slowly becoming available in many computing environments [1].

For a detailed description of background, survey of related work, and explanation of terminology, the reader may refer to the full version of this paper. [4]

2 Algorithm

We believe there are two key ideas for deep exploration. The first is to go deep using BDDs at the expense of completeness, by ensuring that BDD sizes remain tractable. This is accomplished by the use of partitioning and under-approximation. The second idea is starting multiple BMC runs, one from each seed. To keep the runtime practical we make these runs in parallel by using the computing power of a grid. This is a non-conventional way of parallelizing BMC. For the circuits where the BDD based exploration is able to build the transition relation cheaply our method appears to overcome the main shortcoming of classical SAT-BMC which is its inability to perform deep state exploration.

To summarize, our algorithm has following two stages:

1. Generate deep seed-states using partitioning and approximation techniques
2. Distribute seeds on the grid focussing on minimizing unnecessary runs.

Generate Deep Seeds: We perform a full traversal of the state space by partitioning the transition relation, as well as the computed sets of states so that both the graphs and associated calculations remain tractable. When the BDD calculations are no longer manageable, we perform successive under-approximations. At each step of image computation, we use a subset of the actual set of states. Such massive under-approximation may result in successive traversal not always leading to a deeper state. However, probabilistically speaking, if the number of states in any computed image set is more than the sum in the previous steps, as is often the case, then there is a high probability that with successive application of "smaller" image function obtained from a partition of the transition relations, most nodes in our path of deep-traversal will indeed be deep.

Parallel Seed SAT: In order to determine the initial seed states for SAT, we use the following two approaches: Firstly, a small number of BDD based partitions are explored fully and CNF clauses are written out at regular intervals, say every 5 steps. Alternatively, a large number of partitions are explored very rapidly with under-approximation, and the resulting deep states are used to seed SAT. By making multiple BMC runs, starting from various points along the state traversal, we can ensure that at least a subset of the BMC executions start from a deep state. Since all BMC runs can be made in parallel so this leads to a non-traditional method of parallelizing BMC.

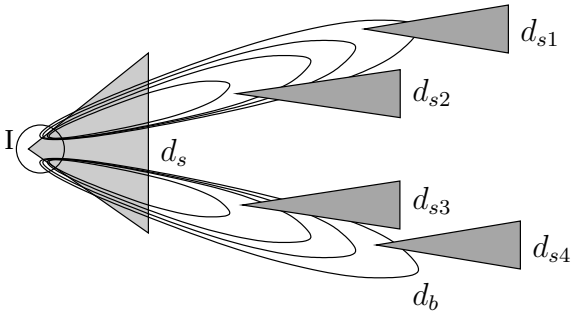

Fig. 1. Seeding multiple SAT-BMC runs from POBDD reachability

The Proposed Algorithm:

1. *Partition_reach*: Use state partitioning in reachability to get different and divergent paths exploring state space.
2. *Approx_Partition_reach*: Do reachability analysis with under-approximation – during each image computation, pick a subset of the newly found reachable states and add it to reachable set in order to avoid BDD blowup problems.
3. *Generate_seed*: At regular intervals, whenever a threshold is crossed, store the seeds and pass it to a new instance of the SAT solver.
4. *Start_Seeded_SAT*: From each of these seeds, run an instance of SAT-based BMC up to a small enough depth.
5. *Run_in_Parallel*: Run one SAT instance on each machine of the grid.
6. *Termination_condition*: Allow BDD exploration and all SAT explorations to continue in parallel until bug is seen or timeout is reached.

3 Results

In this section, we present our experimental results on some industrial circuits. Several of these properties are deep and pose some difficulty for SAT-BMC as well as simulation based methods. The experiments are run on a grid of computers that include up to 100 independent Xeon CPUs (1.5 to 2.3 GHz) running linux. We use an in-house grid middle-ware, CyberGrip [1], developed at Fujitsu Labs Limited, Japan, for managing jobs executed on the grid. Our program is implemented on top of VIS-2.0 and uses CUDD BDD package and zchaff SAT-solver. The POBDD algorithm is run on a single processor but the CNF files generated are transfered to different nodes on the grid where a BMC run is fired in parallel.

Details of Experiments: Random simulation, using VIS-2.0 upto 100,000 steps is unable to find a bug in any of the circuits in the benchmark. We perform simulation to find deep states and seed BMC from there. This is similar to the approach of [3], except that we use a different random seed for each simulation depth. For each circuit, we run simulation, in steps of 1,000 from 2,000 to 10,000. When the depth is reached, we pick the state reached at the end of the simulation and seed SAT from there.

			Existing					Proposed			
	Num.	Error	Total Time (sec)					Time (sec)			Num.
Ckt	latches	Depth	BDD	POBDD	BMC	Sim	Sim/BMC	Seed	BMC	Total	CPU
b1	125	59	7	**3.2**	T/O	NB	167	3.2	N/A	**3.2**	1
b2	70	85	3.4	**2**	T/O	NB	115	2	N/A	**2**	1
b3	66	23	1.9	**1.3**	T/O	NB	268	1.3	N/A	**1.3**	1
b4	66	59	1.9	**1.3**	T/O	NB	3097	1.3	N/A	**1.3**	1
b5	170	36	T/O	T/O	T/O	NB	2758	27	36	**63**	9
b6	201	29	3148	2857	T/O	NB	1407	156	20	**176**	3
b7	123	60	**258**	976	T/O	NB	T/O	35	429	464	14
b8	169	23	T/O	T/O	T/O	NB	T/O	198	55	**253**	28
b9	148	27	T/O	T/O	T/O	NB	T/O	280	1580	**1860**	70

"T/O" is a timeout of 2 hrs, "NB" means no bug found.

Table 1. Comparison of the time taken in seconds by various approaches.

Table 1 shows the time taken by different methods: Existing approaches are invariant checking using BDDs and POBDDs; SAT-BMC; simulation to 5,000 steps and an application of SAT solver after 5,000 steps. The last four columns of Table 1 shows the details of time spent by the proposed method: the time taken for (a) POBDD based reachability to discover the seed state, (b) the SAT-solver to find the bug from there, (c) the total time and (d) the number of CPUs of the grid that are actually used. We allow each method to run for 2 hours. The results for all the methods are shown in table 1. Note that the proposed method is the only one that is able to find the error in benchmarks b8 and b9.

4 Conclusions

Based upon our analysis of the experimental results, we believe that the proposed hybrid method has various benefits. It is computationally inexpensive in terms of overhead and an alternate way of parallelizing SAT-based BMC – each of many processors can execute a BMC from a different set of initial states. The only data that is passed over the network is at the very beginning, after that no synchronization is required, until termination. Such parallelization has no interdependence at all, and can therefore very effectively utilize a number of processors in a large grid, without creating communication overhead between the processors. This method also effectively exploits the advantage of symbolic BDD based search as well as SAT. If there are a large number of partitions or if certain partitions are difficult, performing cross-over images between them can be difficult, and this may be the bottleneck in getting to the error. This can be overcome by SAT based BMC, which is "locally complete" from its originating point and does not compute sets of states.

Although a very large grid was available, in typical experiments only a small number of CPUs were used. This suggests significant scope to improve the quality of results and possibility to tackle larger problems with further research.

References

[1] Akira Asato and Yoshimasa Kadooka. Grid Middleware for Effectively Utilizing Computing Resources: CyberGRIP. In *Fujitsu Scientific and Technical Journal*, volume 40, pages 261–268, 2004.

[2] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1):7–34, July 2001. Kluwer Academic Publishers.

[3] Pei-Hsin Ho, Thomas Shiple, Kevin Harer, James Kukula, Robert Damiano, Valeria Bertacco, Jerry Taylor, and Jiang Long. Smart Simulation Using Collaborative Formal and Simulation Engines. In *Proc. of the IEEE/ACM International Conference on Computer-Aided Design*, pages 120–126, November 2000.

[4] Subramanian Iyer, Jawahar Jain, Mukul Prasad, Debashis Sahoo, and Thomas Sidle. Error Detection using BMC in a Parallel Environment. In *Technical Report, Department of Computer Sciences, University of Texas at Austin*, 2005.

[5] Debashis Sahoo, Subramanian Iyer, Jawahar Jain, Christian Stangier, Amit Narayan, David L. Dill, and E. Allen Emerson. A Partitioning Methodology for BDD-based Verification. In *Formal Methods in Computer-Aided Design*, volume 3312 of *Lecture Notes in Computer Science*, pages 399–413. Springer-Verlag, January 2004.