Improved Symbolic Verification Using Partitioning Techniques

Subramanian Iyer[1,2], Debashis Sahoo[1,3], Christian Stangier[1], Amit Narayan[1], and Jawahar Jain[1]

[1] Fujitsu Laboratories of America, Sunnyvale, CA 94085, USA
FAX: (408)530-4515
{siyer,dsahoo,cstangier,amit,jawahar}@fla.fujitsu.com
[2] Dept. of Computer Sciences, University of Texas at Austin, TX 78712, USA
[3] Dept. of Electrical Engineering, Stanford University, CA 94305, USA

Abstract. This paper presents an efficient method to avoid memory explosion in symbolic model checking through the use of partitioning techniques. Dynamic repartitioning of Partitioned OBDDs (POBDDs) is investigated to enhance the efficiency of symbolic verification techniques. New and improved algorithms are presented for reachability based invariant checking and for model checking a fraction of CTL that is found to be most important in practice. These algorithms hinge on dynamically repartitioning the state space and exploit the partitioned nature of the data structure. The effectiveness of the partitioning approach is demonstrated on both proprietary industrial designs as well as public benchmark circuits. Notably, the approach is able to verify, and in some cases falsify, properties of interest in industry on large designs which were otherwise intractable for verification by other state-of-the-art tools.

1 Introduction

Computation Tree Logic (CTL) [6] has proved to be a popular specification language for expressing properties for formal verification of designs, especially hardware. Model checking [6,7] is the prominent automatic formal verification methodology. Reduced Ordered Binary Decision Diagrams (ROBDDs) [4] currently serve as the data structure of choice during symbolic model checking [13], because they have the desirable property of being canonical as well as manipulable. ROBDDs have efficient representations for many functions of practical interest. Unfortunately, some applications require representation of functions that only have exponential ROBDD size. This limits the complexity of problems that can be attacked by ROBDDs.

A more efficient representation was proposed through the use of Partitioned-ROBDDs (POBDDs) [12] especially for large designs. In this approach, different partitions of the Boolean space are allowed to have different variable orderings and only one partition needs to be in memory at any given time. In this paper, we extend and improve this approach to address the following issues.

D. Geist and E. Tronci (Eds.): CHARME 2003, LNCS 2860, pp. 410–424, 2003.

Firstly, we propose the use of *dynamically* Partitioned-OBDDs. This partitioning technique dynamically varies the number of partitions that are created and is thereby able to avoid memory explosion. Theoretical evidence [2] suggests that representations using this approach can be exponentially more compact than an approach using a fixed constant number of partitions. We incorporate this dynamic repartitioning in reachability based invariant checking as well as model checking for a portion of CTL.

Secondly, we also propose a new algorithm for model checking a significant portion of CTL. This portion is defined as those formulae, which can be represented without the use of the greatest fixpoint in existential normal form. More precisely, we efficiently handle the temporal modalities EX, EF and their duals as well as EU. Such formulae are found to be a significant fraction of the properties that are of practical interest to hardware designers. In particular, this includes *invariants* as well as *FSM deadlock avoidance* properties.

It has been previously shown [16,15] that POBDDs can be used analogously to OBDDs for most applications. However, a straightforward implementation using the conventional algorithm leads to excessive overhead in the form of disk accesses, BDD variable reorderings, etc.. The proposed algorithm leverages the partitioned nature of the data structure in order to significantly reduce these overheads. This is, to our knowledge, the first algorithm to take full advantage of the ideas of partitioning at an algorithmic level in the model checking procedure.

Thirdly, though it may not be obvious, use of partitioning based representation is not practical at all if one can not devise a practical and competitive strategy to discover, when appropriate, a path leading to an erroneous state. We provide a novel method to determine the same. In many cases, this method may be able to provide an error trace more efficiently than using classical OBDD based methods.

To our knowledge, this is one of the few papers demonstrating the use of partitioning based data structures in an industrial setting. On many public benchmark circuits also it shows non-linear gains in space and time, often an order of magnitude or more, over the best known state of the art tool (VIS). Thus, we demonstrate that BDD-based verification can be expanded over the limits of classical ROBDD approaches.

1.1 Comparison with Related Work

The use of *partitioned transition relations* [5] was proposed to control the size of symbolic representation of transition relations. The *set of latches* is divided into different groups which control the ROBDD-size of the transition relation and allow early quantification as well. In POBDDs, the entire Boolean space is partitioned. Thus, in order to distinguish the sense in which partitioning is performed, it would be more appropriate to call the former as **clustered-**transition relations. Indeed, the two approaches are orthogonal and these "clustered"-transition relations are used in the image computation of our approach as well.

Recently, a method for distributed model checking was studied by [10,9]. It parallelizes the classical symbolic model checking algorithm using the partition-

ing approach suggested in [15]. This approach uses *slicing*, which is similar to partitioning, with the objective of doing model checking in a distributed fashion. This approach does not address issues related to costs of communication and variable ordering in different partitions. In particular, this approach partitions the computation into a fixed number of fragments equal to the number of processors available in the distributed environment. However as noted in the literature [2], a partitioning scheme with k partitions can be exponentially more succinct than one with just $k-1$ partitions. Thus, the apriori selection of the number of fragments greatly limits the efficiency of the partitioned data structure. Indeed the gain from such a *static* method would be obtained substantially from parallelization rather than from the inherent algorithmic advantages offered by the POBDD data structure.

In contrast, our algorithms effectively capitalize on the partitioned nature of the data structure. We require only one partition to be in memory for any image computation, and each partition can be independently ordered. Significantly, this approach incorporates a dynamic repartitioning scheme which allows for an unbounded number of partitions to be automatically created when necessary. At the same time, we show how to drastically cut down the number of instances of inter-partition communications as compared to the classical approach. This reduces the number of transfers and reorderings of large BDDs between partitions and is found to be a significant gain in practice. We also address the issue of efficient determination of error trace in the presence of partitioning.

In the rest of this paper, we first give an overview of POBDDs and the appropriate verification techniques. Then, we describe the proposed algorithms followed by the experimental results and finally conclusions.

2 Preliminaries

The idea of partitioning was used to discuss a function representation scheme called partitioned-ROBDDs in [12,11] which was extensively developed in [16].
Definition. [16] Given a Boolean function $f : B^n \rightarrow B$, defined over n inputs $X_n = \{x_1, \ldots, x_n\}$, the partitioned-ROBDD (henceforth, POBDD) representation χ_f of f is a set of k function pairs, $\chi_f = \{(w_1, f_1), \ldots, (w_k, f_k)\}$ where, $w_i \colon B^n \rightarrow B$ and $f_i \colon B^n \rightarrow B$, are also defined over X_n and satisfy the following conditions:
1. w_i and f_i are ROBDDs respecting the variable ordering π_i, for $1 \leq i \leq k$.
2. $w_1 \vee w_2 \vee \ldots \vee w_k = 1$
3. $w_i \wedge w_j = 0$, for $i \neq j$
4. $f_i = w_i \wedge f$, for $1 \leq i \leq k$ The set $\{w_1, \ldots, w_k\}$ is denoted by W. Each w_i is called a *window function* and represents a *partition* of the Boolean space over which f is defined. Each partition is represented separately as an ROBDDs and can have a different variable order. Most ROBDD based algorithms can be adapted easily for POBDDs.

Partitioned-ROBDDs are canonical and various Boolean operations can be efficiently performed on them just like ROBDDs. In addition, they can be ex-

ponentially more compact than ROBDDs for certain classes of functions. The practical utility of this representation is also demonstrated by constructing ROBDDs for the outputs of combinational circuits [16]. An excellent comparison of the computational power of various BDD based representations and partitioned-ROBDDs may be found in [2].

2.1 Reachability and Model Checking

We omit the syntax of CTL as it is widely known and readily available in the literature. We shall only note that it is possible to express any CTL formula in terms of the Boolean connectives of propositional logic and the existential temporal operators EX, EU and EG. Such a representation is called the *existential normal form*.

Model Checking is usually performed in two stages: In the first stage, the finite state machine is reduced with respect to the formula being model checked and then the reachable states are computed. The second stage involves computing the set of states falsifying the given formula. The reachable states computed earlier are used as a *care set* in this step.

Since there exist computational procedures for efficiently performing Boolean operations on symbolic BDD data structures, including POBDDs, model checking of CTL formulas primarily is concerned with the symbolic application of the temporal operators. EXq is a backward image and uses the same machinery as image computation during reachability, with the adjustment for the direction. $EpUq$ (resp. EGp) has been traditionally represented as the least (resp. greatest) fixpoint of the operator $\tau(Z) = q \vee (p \wedge EXZ)$ (resp. $\tau(Z) = p \wedge EXZ$).

Invariants are CTL formulas of the form AGp, where p is a proposition, and can therefore be checked during the initial reachability computation itself.

The standard reachability algorithm is based on a breadth-first traversal of finite-state machines [8,13,19]. The algorithm takes as inputs the set of initial states, $I(s)$, expressed in terms of the present state variables, s, and a transition relation, $T(s, s', i)$, relating the set of next states, $N(s')$, that a system can reach from a state s on an input i. The transition relation, $T(s, s', i)$, is obtained by taking a conjunction of the transition relations, $s_k^{'} = f_k(s, i)$, of the individual state elements, i.e., $T(s, s', i) = \prod(s_k^{'} = f_k(s, i))$. Given a set of states, $R(s)$, that the system can reach, the set of next states, $N(s')$, is calculated using the equation $N(s') = \exists_{s,i}[T(s, s', i) \wedge R(s)]$. This calculation is also known as *image computation*. The set of reached states is computed by adding $N(s)$ (obtained by replacing variables s' with s) to $R(s)$ and iteratively performing the above image computation step until a fixed point is reached.

2.2 Reachability Using POBDDs

In the context of Partitioned-OBDDs, we can derive a transition relation, T_{jk}, from partition j into partition k by conjoining T with the respective window functions as $T_{jk}(s, s', i) = w_j(s)w_k(s')T(s, s', i)$.

```
preImgPart(Bdd, j) {
   return preImage(Bdd, T_jj)
}

preImgComm(S){
   result = ∅
   foreach (partition j)
      temp = preImage(S_j, T_jj-bar)
      foreach (partition k ≠ j)
         temp_k = temp restricted to w_k
         reorder BDD temp_k from partition order j to order k
         result_k = result_k ∨ temp_k
      end for
   end for
   return result
}
```

Fig. 1. Image Computation Algorithm

The Partitioned-ROBDD based traversal algorithm uses the ROBDD based algorithm in its inner loop to perform fixed point on individual partitions. Let us assume that we are given a partitioned-ROBDD representation $\chi_R = \{(w_j(s), R_j)|1 \leq j \leq k\}$. If we take the image of R_j under T_{jj}, we obtain $N_j(s') = \exists_{s,i}[w_j(s)w_j(s')T(s,s',i)R_j(s)]$. Since $w_j(s')$ is independent of the variables that are to be quantified, it can be taken out of existential quantification, giving us $N_j(s') = w_j(s')[\ \exists_{s,i}[w_j(s)T(s,s',i)R_j(s)]\ ]$

The image of R_j under T_{jj} lies completely within partition j. Similarly, the image, N_l of R_j under T_{jl} will lie completely within partition l. This observation motivates us to define the image computation in terms of the image computed within the same partition and the image *communicated* to another partition. The former will be called *ImgPart* and the latter will be called as *ImgComm*. Analogously, we define the *pre-image* computations *preImgPart* and *preImgComm*. They are illustrated in the pseudo-code shown in Fig 1.

The pre-image, i.e. *computeEX*, is then obtained by their union, as $preImage(p) := \bigvee_i preImgPart(p_i, i) \vee preImgComm(p)$.

The pseudo-code for computeEX, as applied to POBDD, is in Fig 2a.

Notice that two approaches are possible for the computation of the communicated image: In the first, an image is computed from partition j into each partition $k \neq j$ separately, using the transition relation T_{jk}. Alternately, one can compute the image from partition j into the boolean space that is the complement of partition j, denoted by $\overline{j}$. The former has the advantage that the BDD representations of the transition relations T_{jk} are much smaller, but in return it has to perform $O(n^2)$ image computations. We use the second method in defining *imgComm*. This method requires only $O(n)$ image computations, but each of these is followed by $O(n)$ restrict operations.

3 Improved State Space Traversal

In this section, we will describe the use of a dynamic partitioning scheme where the number of partitions can be increased or decreased as the computation progresses. This can be shown to be exponentially more succinct than the use of a fixed constant number of partitions. We also present a novel algorithm for computing a path from a state with an error to the initial state.

3.1 Dynamic Repartitioning

Dynamic repartitioning of the state space is triggered whenever the size of any partition under observation crosses a certain threshold. The partitioning variables are selected using the history of previously computed windows. Repartitioning is performed by splitting the given partition by cofactoring the entire state space based on one or more splitting variables until the blow-up has been ameliorated for each partition, which was created so far. Initially, the partitioning is done using one splitting variable. The choice of this variable is as explained before. At this point, each new partition is checked to see whether the blow-up has subsided. If not, repartitioning is called again on that partition until the blow-up has subsided in all partition.

Sometimes it is found that the blow-up in the BDD-sizes during an intermediate step of image computation is a temporary phenomenon which eventually subsides by the time the image computation is completed. In such a case the invocation of dynamic global repartitioning of the state space could create a large number of partitions, whose BDD-sizes become eventually very small. These partitions create an unnecessary amount of computational overhead. Hence, it is advantageous to create these partitions *locally* only for that particular image computation and then recombine them before the end of the image computation. To create these local partitions, we can cofactor the state space using the ordered list of splitting variables that was generated earlier.

Our algorithm for checking invariants performs successive steps of image computation on each R_j under T_{jj}. Since these steps, *imgPart*, of image computation add states only within the same partition, and since different partitions are disjoint, we are guaranteed that the same state is not being visited multiple times within different partitions. Once a fixpoint is reached within a partition j, the procedure *imgComm* is used to communicate the new set of states to the partition l for for $1 \leq l \leq k$ and $l \neq j$. At any stage, where new states are added into the reached states set, we check for the violation of the invariant presented. If failure is detected, we stop and call the error trace mechanism to retrieve a path from the initial states to an error state. Otherwise, we proceed with traversing more states until the entire state space is exhausted, at which point, the formula has passed.

3.2 Tracing Erroneous Paths

In order to obtain a path from an error state e back to an initial state i, the naive idea would be to compute successive preImages beginning with e, until

i is reached. After a few steps of computing backward images, one would be faced again with a rapidly increasing BDD size. In order to avoid this blow-up in BDD-size, we need to be able to isolate a set of candidate predecessors for the current state so that the next preImage computation does not have have to handle too large BDDs. In the case of ROBDDs, this is accomplished by keeping the so called "onion rings" or the frontier of states encountered during each image computation.

In the partitioned setting, the set of possible predecessors may be spread across multiple partitions. Thus it is possible to store these frontier states in a partitioned manner. Therefore the backward image can be computed with respect to only a portion of the frontier states.

Thus, the image computations need to be recorded in a tree-like data structure in order to be able to find the correct subspace for the backward image. For each state s in the set of reachable states S, this tree contains the image computation when the state s was first added to the reachable set S. The structure stores the information required to trace a backward path as follows: For each partition of the boolean space, its *frontier* is defined as the states added to this partition by the most recent invocation of imgComm and the subsequent imgPart operations. Each such frontier is actually a collection of sets, each represented as a BDD, whose set union represents the set of all states that have been reached in this partitions for the first time, but have not yet been used for communication to other partitions. Thus, the number of BDDs in this frontier can be, in the worst case $O(M + d_i)$ where M is the number of partitions, and d_i is the depth of the fixpoint in partition i. For the entire graph this can, in the worst case be, $O(M * (M + d_{max}))$.

To retreive a path from an initial state to a state s, we do the following:

1. Obtain the location in the computation tree that contains s.
2. Take the predecessor frontier of this location in the tree, and compute a backward image into this frontier to find one or more predecessor states.
3. Pick one such predecessor state.
4. Repeat steps 2 and 3 on successive states until an initial state is reached.

This gives us the backward path from state s with an error to an initial state.

Advantages of partitioned error trace: Notice that in the case of ROBDDs, the onion rings can get large in size. An effect of having these large sized representations is that image computations get more expensive. As noted before, ignoring the frontier states and performing a backward reachability is even more expensive, and in that case the backward path can be longer in length too.

Observe that partitions can often be assymetric with respect to the space and time required for performing image computations on them. Therefore, in the presence of multiple paths from an error state to the initial states, it would be advantageous to compute the shortest path in terms of computational effort rather than the length of the path. In order to do this, we annotate the nodes of the tree with information about the amount of time the corresponding image computation required. These annotations can be used as an indicator of how much time the backward image would take, and thus, in step 3 above, they can

assist in reducing the time spent in finding a more practical path back to the initial states.

4 Model Checking Fixpoint Formulas

As mentioned in section 2.2, the modalities EX, EU and EG suffice to represent any CTL formula in existential normal form.

In particular, we note that the deadlock property $AG(p \rightarrow EFq)$ can be represented in the "greatest fixpoint free" fragment of CTL Since invariant checking and deadlocks form a large fraction of formulas that are of practical interest to designers, we will first look at the least fixpoint operator $E(pUq)$. Note that, p and q are not restricted to propositions and can be any CTL formulae.

4.1 Why Communication Is Expensive

It is important to notice that there are fundamental differences between the two image operations - *imgPart* and *imgComm*. Observe that imgPart(R_j) is in the same partition j as the original BDD R_j and therefore only one partition needs to be in memory for its computation. On the other hand, imgComm(R_j) computes an image into $\overline{j}$, i.e., *every partition other than j*, therefore it needs to finally access and modify every partition. This gives rise to two important issues with respect to communication.

Firstly, the reached state set of every partition needs to be accessed. In the case of large designs, where the BDDs of even a single partition can run into millions of nodes, this usually means accessing stored partitions from the disk.

Secondly, the BDD variable order of the computed imageset must be changed from the order of the j^{th} partition to that of each of its target partitions, before the new states can be added to the reached set in the target. Again, for large designs, reordering a large BDD can be an extremely expensive operation.

In this context, image computation within a partition, *ImgPart*, is a relatively inexpensive operation as compared to communication between partitions, ImgComm. Therefore, in the interest of minimising transfer of BDDs from one partition to another, we need a new algorithm that would decrease the number of invocations of ImgComm whenever possible.

An associated advantage of performing image computation repeatedly within a partition before communicating, is that it allows some errors to be caught much earlier. When a formula fails in any partition, it becomes unnecessary to explore the other partitions any further. In this manner, it may be possible to locate the error by exploring a smaller fraction of the state space than otherwise necessary.

In the rest of this section, we will present, in the context of POBDDs, the improved model checking algorithm designed to take advantage of partitioning.

4.2 Evaluating the Least Fixpoint $E(pUq)$

The classical algorithm for the least fixpoint operator is presented in Figure 2a in terms of the POBDD data structure.

```
computeEX(p) {
   R ← p
   forall (partitions j)
      S_j ← preImgPart(R_j, j)
   end for
   S ← S ∨ preImgComm(R)
   output S
}
computeEU(p, q) {
   S ← q and S.old ← φ
   repeat
      S.old ← S
      S ← q∨(p∧computeEX(S))
   until(S = S.old)
   output S
}
```

a) Classical Algorithm

```
computeEU(p, q) {
   S ← q and S.old ← φ
   repeat
      S.old ← S
      forall (partitions j)
         repeat
            S_j.old ← S_j
            S_j ← S_j ∨ (p_j ∧ preImgPart(S_j, j))
         until(S_j = S_j.old)
      end for
      S ← S ∨ (p ∧ preImgComm(S))
   until(S = S.old)
   output S
}
```

b) New Algorithm

Fig. 2. Algorithms for $E(pUq)$ using Partitioned-OBDDs

Notice that in the computation of $E(pUq)$, the preImage computation forms the bulk of the work performed by the algorithm. As noted in section 4.1, the cost of performing communication during every preImage is quite large. This penalty is due to resources required to transfer BDDs between partitions, to reorder the BDDs before such transfer can occur and to fetch the partitions from storage in order that the new states can be conjuncted with p and disjuncted with q. Therefore, it is important to *postpone the invocation of preImgComm*, i.e., to perform as many image computations as possible locally within each partition before communication is performed across partitions.

A New Algorithm for $E(pUq)$

In this section we describe a new algorithm for model checking least fixpoint CTL formulas and sketch a proof of its correctness. Algorithm 2b for computing the set $E(pUq)$ is designed to take advantage of the partitioned nature of the data structure. Notice that we explore each partition independently of the others until they reach a fixpoint individually. Then, we perform the communication across partitions.

This allows us to keep just one partition in memory at any given time. It also greatly reduces the number of communication induced BDD transfers, disk accesses and variable reordering calls.

Before proving the correctness of the new algorithm, we define some notation. Let the set of states S at the end of the k^{th} iteration of the outermost repeat-until loop in algorithm 2b be represented by S^k.

For every state $s \models E(pUq)$, either $s \models q$ or there exists a sequence of states $s_0, s_1, \ldots, s_k$ that has the smallest length $k \neq 0$ such that $s_0 = s$, $s_k \models q$,

$\forall i < k : s_i \models p$ and $\forall i < k : s_i \in preImage(s_{i+1})$. Such a sequence of states is called a *witness* for the inclusion of s in $E(pUq)$, and k is its *length*.

For the sake of convenience, we will use the symbol for a formula to also represent the set of states it represents. We first show that algorithm 2b terminates.

Lemma 1. *(Termination) For any integer i, $S^{i+1} \supseteq S^i$. The inequality is strict unless a fixpoint is reached.*

The proof is evident from the construction of sets S^k. Since any step of the procedure must add at least one new state to the set S, we have termination at the end of at most as many iterations as there are states in the space under consideration.

Theorem 1. *The procedure computeEU of algorithm 2b, given the set of states corresponding to formulas p and q as inputs, terminates with the output S being precisely the set of states that model the formula $E(pUq)$.*

Proof: **Soundness:** We prove by induction on the sets S^k that the procedure is sound, i.e., at all times $S \models E(pUq)$. This clearly holds for any state in the initial set $S^0 = q$, since any state satisfying q also satisfies $E(pUq)$.

Assume, it holds for S^i, i.e., that $S^i \models E(pUq)$. Consider a state $s \in S^{i+1} - S^i$. Then, by construction of S^{i+1} from S^i, we have $s \models p$. Either s is added during some step of the inner fixpoint loop or it is added in a step of communication, i.e., $s \in preImgComm(S^i)$.

Suppose s is added in the inner fixpoint loop of some partition j. Since S^i is a POBDD, let us call the projection of S^i in partition j as S^i_j. From before, we know $preImgPart(S^i_j, j) \subseteq preImgPart(S^i) \subseteq preImage(S^i)$. Also notice that the variable for the inner fixpoint is initialized to S^i_j. Therefore, every state added in the first step of the inner fixpoint models $p \wedge EX(E(pUq))$ and therefore models $E(pUq)$. Consequently, we can show by induction that any state added in the inner fixpoint loop for partition j must model $E(pUq)$.

In the second case, s was added in some step of the communication. Considering that $preImgComm(S^i) \subseteq preImage(S^i)$, any state added in the communication step models $p \wedge EX(E(pUq))$, and therefore $E(pUq)$. In particular, $s \models E(pUq)$.

Consequently, $S^{i+1} - S^i \models E(pUq)$ and the soundness of the procedure follows by induction.

Completeness: We next show the completeness, i.e., that every state of $E(pUq)$ is indeed in set S. Let T^k be the set of states, whose inclusion in $E(pUq)$ is witnessed by a path of length at most k. We prove by induction on k that $T^k \subseteq S$. In the base case, this trivially holds because $T^0 = q = S^0 \subseteq S$.

Now, let us assume that $T^i \subseteq S$. For any state $s \in T^{i+1}$ consider the sequence of states $s_0 = s, s_1, \ldots, s_{i+1}$ that witnesses its inclusion in $E(pUq)$. We will show that $s \in S$.

Now, the sequence $s_1, \ldots, s_{i+1}$ is a witness for s_1, therefore $s_1 \in T^i \subseteq S$. In particular, there exists a smallest j so that $s_1 \in S^j$. We know that $s \models p$ and

$s \in preImage(s_1) \subseteq preImage(S^j)$. From the definition of S^j and Algorithm 2b, we have that

$$\begin{aligned} S^{j+1} &\supseteq S^j \vee (p \wedge preImgPart(S^j)) \vee (p \wedge preImgComm(S^j) \\ &= S^j \vee (p \wedge (preImgPart(S^j) \vee preImgComm(S^j))) \\ &= S^j \vee (p \wedge (preImage(S^j))). \end{aligned}$$

Therefore, $s \in S^{j+1} \subseteq S$, whereby $T^{i+1} \subseteq S$. By induction, this gives us $E(pUq) \subseteq S$.

Together with lemma 1, this proves that algorithm 2b terminates with the set $S = E(pUq)$.

■

4.3 Evaluating the Greatest Fixpoint *EGp*

The model checking of EGp is done by computation of the greatest fixpoint of the operator $\tau(Z) = p \wedge EXZ$. As in the case of least fixpoint, one would like to postpone the communication until after each partition has reached its individual fixpoint independent of the other partitions. However, the description of this is considerably more complex and thus far we have only implemented a simple, classical, version of the greatest fixpoint algorithm for EGp in terms of POBDDs.

Even so, most specifications of interest in practice are expressible in the fragment of CTL free of greatest fixpoints. For e.g., deadlock avoidance properties of the form $AG(p \rightarrow EFq)$ and invariants can both be expressed in existential normal form using only least fixpoints. Therefore, we find that the inability to postpone communications for the greatest fixpoint does not impose a great disadvantage in most practical applications.

5 Experiments

We implemented dynamic partitioning-based model checking using the CUDD-package [18] (version 2.3.0) for OBDD representation. We use the routines from VIS [3] (version 1.4) for reading in the design and to build the initial transition relation using the IWLS95 method [17]. Our implementation can be thought of as building on top of VIS and therefore a comparison with VIS is natural.

We found empirically that for our benchmarks VIS-2.0 using the MLP [14] method performs worse than VIS-1.4 using the IWLS95 method, probably due to known problems in preimage computation. Thus, we compared our methods to VIS by using the IWLS95 method for both.

Benchmarks and Experimental Setup

For our experiments, we used the designs from the Vis Verilog benchmark suite [1]. This suite also contains properties given in CTL formulas for verification. We pick the properties which when expressed existentially are “greatest

Table 1. Invariant Checking on Large Designs

Circuit	Number of Partitions	Peak Nodes			Time (seconds)		
		VIS	POBDD	Gain	VIS	POBDD	Gain
palu	4	371 K	150 K	2.5	253	102	2.5
product	4	919 K	116 K	7.9	1394	546	2.6
am2910	4	>1.52 M	187 K	>81	>24h	1.2 K	>72
rotate32	43	>825 K	640 K	>1.3	>24h	8 K	> 8.6
spinner32	60	>1.61 M	362 K	>4.4	>24h	10 K	>11.3
vsa16a	4	>1.02 M	722 K	>14	>24h	22.6 K	>3.8

fixpoint free". On the entire benchmark suite this is found to cover about 80 % of all properties, which is believed to be typical. Finally, we also used proprietary designs that were made available by Fujitsu designers.

The parameters of VIS and CUDD are left unchanged at their default values. Experiments on the public benchmarks were performed on dual-processor Xeon 2.2Ghz workstations with 2 GB of RAM running Linux. The invariant checking as well as model checking experiments used dynamic partitioning. Both were run with a timeout limit of 24 hours.

The peak number of live nodes is given by Peak Node. The CPU time is measured in seconds and given as Time. The column denoted with Time Gain (resp. Space Gain) describes the gain in time (space) of POBDDs over VIS.

Results on Invariant Checking. We compare our POBDD method to the standard VIS approach on invariant checking in Table 1. Note that this table is restricted to the largest entries (BDD-nodes > 300K) in the benchmark suite. Our partitioned approach clearly outperforms the state-of-the-art VIS in time as well as in space. Especially for the larger circuits the improvement is drastic, since we complete the verification of four circuits that timed out using VIS.

Comparison with Static Partitioning It is natural to analyse what benefit dynamic partitioning offers over static partitioning. In Fig. 3, we compare the performance of the proposed dynamic partitioning based invariant checking approach with invariant checking based on the static partitioning method of [15]. In particular, note that in the last case, *vcrc32_8*, the previous approach timed out after 86,400 seconds whereas we are able to complete in about 12,000 seconds.

Results on Model Checking. The results on runtime and space requirements in model checking are presented in Table 2.

POBDDs may not sometimes show their full potential on the smaller circuits due to the overhead of creating and maintaining partitions. Nevertheless, the results show that POBDD-based model checking can out-perform VIS even on such cases in time as well as in space.

But, more important are the last few entries in the table, showing the harder benchmarks. Here, the POBDD-based model checking clearly outperforms the

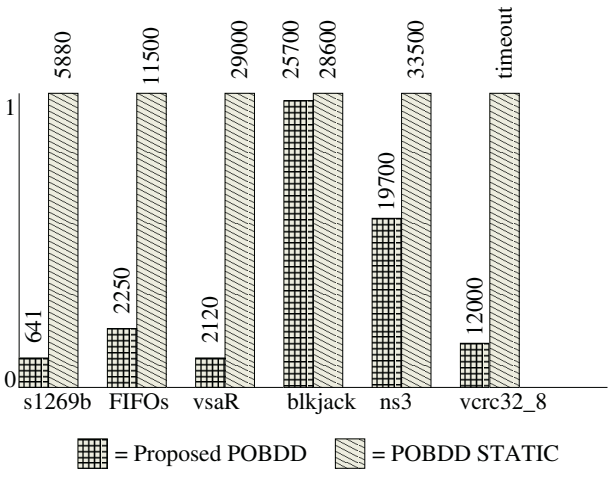

Fig. 3. Comparison of Times taken (Normalized) by Different Partitioning Approaches for Invariant Checking on some Large designs

Table 2. Model Checking on Large Designs

Circuit	Number of Partitions	Peak Nodes			Time (seconds)		
		VIS	POBDD	Gain	VIS	POBDD	Gain
product	4	919 K	108 K	8.5	1450	437	3.3
s1269b	4	2.3 M	317 K	7.2	7340	170	43.0
am2910	4	>4.9 M	127 K	>38.2	>24h	324	>266
twoQ	6	>5.5 M	1.8 M	>3.1	>24h	11.4 K	>7.6
palu	12	>10.5 M	3 M	>3.5	>24h	40.1 K	>2.2
am2901	5	>5.7 M	1.94 M	>2.9	>24h	45.4 K	>1.9

classical approach and is able to even finish four of the designs that cannot be finished within the given 24 hour timeout when using VIS.

It is also noteworthy, that the maximum peak BDD-size of one partition is often an order of magnitude smaller than the maximum peak node size for ROBDDs. We have observed that this reduction is in many cases more than the number of partitions created.

Industrial Circuits The properties for industrial circuits were taken from actual Fujitsu designs with sizes ranging from 2000 to 10000 flip-flops. Table 3 shows the summarized results for the comparison of POBDD-based model checking with VIS for three different types of properties. For the first two properties, *Index range* and *full-case*, the POBDD method is able to finish 11 (resp. 5) more properties than the OBDD method.

Table 3. Model Checking of Industrial Circuits (2,000 to 10,000 flip-flops)

Property Type	Method	Pass	Fail	Timeout
Index out of range	POBDD	678	0	0
	VIS	667	0	11
Full Case	POBDD	16	0	0
	VIS	11	0	5
Synchronizer data stability	POBDD	2	4	0
	VIS	0	2	4

For the third property, *data stability*, the POBDD method is actually able to detect 2 failures more in addition to the passing properties than the OBDD approach.

6 Conclusions

In this paper we addressed the memory explosion problem associated with model checking through the use of dynamically Partitioned-OBDDs. We have shown that it can be significantly better for problems, where the state of the art can require impractically large computational resources. The significant advantage of the proposed verification technique is its ability to control the memory required. Usually, this has the added advantage of improvement in run-time, which is primarily governed by the BDD-sizes. On large circuits we find that the computational savings offered by the proposed partitioning based model checking can be significant. We have shown cases, where our proposed method could finish in just a few thousand seconds, whereas other approaches timed out after a day. Importantly, a new algorithm for invariant checking and for model checking the fragment of CTL free of greatest fixpoint in the existential normal form are presented. This can handle many more properties of practical interest and truly exploit the theoretical and practical benefits of dynamically partitioned-OBDDs.

Acknowledgment. The authors would like to thank Prof. E. Allen Emerson and Prof. David Dill for their advice and encouragement.

References

1. Vis verilog benchmarks http://vlsi.colorado.edu/ vis/. Technical report.
2. B. Bollig and I. Wegener. Partitioned bdds vs. other bdd models. In *Proc. of the Intl. Workshop on Logic Synthesis*, 1997.
3. R. K. Brayton, G. D. Hachtel, A. L. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. Cheng, S. A. Edwards, S. P. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, and T. Villa. VIS: A System for Verification and Synthesis. In *Computer Aided Verification*, 1996.
4. R. E. Bryant. Graph based algorithms for Boolean function representation. *IEEE Transactions on Computers*, C-35:677–690, August 1986.

5. J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic Model Checking with Partitioned Transition Relations. In *Proc. of the Design Automation Conf.*, pages 403–407, June 1991.
6. E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. IBM Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.
7. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.
8. O. Coudert, C. Berthet, and J. C. Madre. Verification of Sequential Machines Based on Symbolic Execution. In *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, 1989.
9. Orna Grumberg, Tamir Heyman, and Assaf Schuster. Distributed symbolic model checking for μ-calculus. In *Computer Aided Verification*, pages 350–362, 2001.
10. Tamir Heyman, Daniel Geist, Orna Grumberg, and Assaf Schuster. Achieving scalability in parallel reachability analysis of very large circuits. In *Computer Aided Verification*, pages 20–35, 2000.
11. J. Jain. On analysis of boolean functions. *Ph.D Dissertation, Dept. of Electrical and Computer Engineering, The University of Texas at Austin*, 1993.
12. J. Jain, J. Bitner, D. S. Fussell, and J. A. Abraham. Functional partitioning for verification and related problems. *Brown/MIT VLSI Conference*, March 1992.
13. Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
14. I. Moon, G. D. Hachtel, and F. Somenzi. Border-Block Triangular Form and Conjunction Schedule in Image Computation. In *Proc. of Formal Methods in CAD (FMCAD'00)*, volume 1954 of *Lecture Notes in Computer Science*, 2000.
15. A. Narayan, A. Isles, J. Jain, R. Brayton, and A. Sangiovanni-Vincentelli. Reachability Analysis Using Partitioned-ROBDDs. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 388–393, 1997.
16. A. Narayan, J. Jain, M. Fujita, and A. L. Sangiovanni-Vincentelli. Partitioned-ROBDDs - A Compact, Canonical and Efficiently Manipulable Representation for Boolean Functions. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 547–554, 1996.
17. R. K. Ranjan, A. Aziz, R. K. Brayton, C. Pixley, and B. Plessier. Efficient BDD Algorithms for Synthesizing and Verifying Finite State Machines. In *Proc. of the Intl. Workshop on Logic Synthesis*, 1995.
18. Fabio Somenzi. CUDD: CU Decision Diagram Package ftp://vlsi.colorado.edu/pub. Technical report.
19. H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Implicit State Enumeration of Finite State Machines using BDD's. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 130–133, November 1990.