On Partitioning and Symbolic Model Checking

Subramanian Iyer[1], Debashis Sahoo[2], E. Allen Emerson[1], and Jawahar Jain[3]

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

Abstract. State space partitioning-based approaches have been proposed in the literature to address the *state-space explosion* problem in model checking. These approaches, whether sequential or distributed, perform a large amount of work in the form of inter-partition (*cross-over*) image computations, which can be expensive. We present a model checking algorithm that aggregates these expensive cross-over images by localizing computation to individual partitions. It reduces the number of cross-over images and drastically outperforms extant approaches in terms of *cross-over* image computation cost as well as total model checking time, often by two orders of magnitude.
Keywords: Symbolic Model Checking, BDD, state partitioning, CTL.

1 Introduction

Model checking is performed by means of successive backwards image computations. Image computation becomes difficult as the data structures representing the state sets grow larger. Large state sets are a direct consequence of the *state-space explosion* problem. Model checking is unable to handle data structures when their size exceeds (roughly by an order of magnitude) what can be reasonably handled in main memory. This frequently happens when handling large designs.

From a practical standpoint, representing the state sets during model checking symbolically using BDDs fails due to this excessive memory requirement. Partitioned symbolic data structures have been proposed in the literature to handle this *memory explosion problem.* Partitioning of the state space is found to balance the trade-off between compactness and canonicity of symbolic BDD representations. In such a framework, each partition of the state space may obey a different variable order.

In a partitioned approach, the state space S is partitioned into subspaces $S_1, S_2, \ldots S_n$. This induces a disjunctive partitioning on the transition relation T into the parts T_{ij} which represents the set of transitions from states in a source partition i to states in the destination partition j. The size of each such transition relation can be further reduced by an implicitly conjoined implementation.

Each partition can be thought of as being the *owner* of a set of states. Transitions from each partition naturally comprise of two components - ones that are wholly local to individual partitions, and ones that span multiple partitions. Correspondingly, the computed image X comprises of a *local* component X_l and

a *cross-over* component X_c. The states corresponding to X_l may be computed locally in each partition. On the other hand, the states in X_c arise out of transitions that originate at a state in one partition and terminate at a state in another, thus, "crossing over" into the destination partition.

Computing cross-over component of the image is often significantly more expensive than the local component for various reasons. Firstly, the cross-over component involves transitions into a potentially larger subspace. Secondly, this incurs the overhead of transporting these states to the partition that "owns" them. Thirdly, the source and destination partitions likely obey different variable orders, and therefore the communicated state set needs to be reordered, which is a known difficult problem as representation sizes become large. Hence even a small reduction in the number of cross-over images can result in a drastic reduction in the total amount of time spent in cross-over images. Our experimental results show that this is indeed the case.

The simplistic way of combining partitioning with the classical model checking algorithm [1, 5], for instance, the distributed model checking algorithm of [2], performs repeated exact images. Each such image computation requires a quadratic number of image computations during each cross-over image computation.

Notice that the set obtained by performing operation EX_l is a subset of the actual image, and in this sense, can be thought of as an under-approximation to EX. This allows for an efficient analysis of reachability [6] and a subset of CTL [3] by replacing a sequence of EX operations by a sequence of the less expensive EX_l operations, interspersed with an occasional EX_c to maintain completeness.

The problem is trickier with greatest fix-points, e.g. the EG operator. The EG operator and its dual AF are important in falsifying and verifying liveness properties. In this case, the final result is the conjunction of successively smaller supersets of the result. If operation EX_c is ignored in pre-image computations, then the result is a subset of the actual pre-image EX. Consequently, some states get pruned early in the greatest fix-point computation for computing the set EG. Since the convergence is on a sequence which is monotonically decreasing, these states pruned early may be lost for ever. Consequently, EX cannot be replaced by EX_l as it compromises on soundness. An important question arises as to how to compute greatest fix-points in the partitioned framework without having to perform repeated frequent cross-over image computations.

In this paper, we propose an alternative *piece-wise* algorithm for model checking CTL formulae in a partitioned setting that addresses these concerns. Our approach exploits the separability of the local and cross-over components of image computation. It performs a number of image computations locally within each partition, and synchronizes occasionally by doing cross-over image computations only when a fix-point is reached locally in each partition.

If during state space traversal, each partition requires many steps of image computation to reach a local fix-point, then the proposed algorithm shows significant gain (which is proportional to the depth of the fix-point).

In section 2, we recall the notions of state space partitioning and the definition of model checking. We present a simple partitioned version of the classical model

checking algorithm in section 3. Section 4 describes our modified algorithm designed to localize computation by postponing cross-over image computations. In the final section, we present our experimental results documenting the increased efficiency of our technique.

2 Preliminaries

In this section, we briefly look at some background related to state space partitioning and image computation, leading up to a description of the classical model checking algorithm in a partitioned framework.

2.1 State Space Partitioning

The idea of partitioning was used to discuss a function representation scheme called partitioned-ROBDDs in [4] which was further extensively developed in [7].
Definition. [7] 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 exponentially 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 [7].

In the rest of this paper, we only consider such *window-based state partitioning*. The reason for this is that this representation is canonical, and allows negation to be performed locally in each partition. Other schemes for dividing the state sets, notably that of [2], need to perform a global synchronization operation to perform negation and this can be expensive.

2.2 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. These two stages can be performed either one after the other by -computing the reachable states first, or in an interleaved manner, where the reachable states are computed on demand. For the purpose of this paper, and to keep the discussion restricted to the model checking algorithm, we shall assume that the set of reachable states is computed and provided *a priori*.

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) fix-point of the operator $\tau(Z) = q \vee (p \wedge EXZ)$ (resp. $\tau(Z) = p \wedge EXZ$).

We now examine the classical model checking algorithm, modified for a partitioned representation of the state sets. This is a simple algorithm, along the lines of the distributed model checking algorithm of [2].

3 Classical Model Checking with Partitioning

First, a word on our terminology. Each partition *owns* states that are in its subspace, as defined by its window function. Conversely, such states *belong* to the partition. We say that a partition performs operations on sets that it owns. The result of such operations may lie in a different subspace and may then need to be transferred to one or more other partitions. It is important to make this distinction between the partition where the operation is performed and the partition to whom the result finally belongs, because they may obey different variable orders[4] and variable reordering is known to be expensive.

Since backward image computation is the basic unit operation in performing model checking, we first examine image computation in the presence of partitioning.

3.1 Partitioned Image Computation

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*. Similarly, the *backward image computation*, which calculates the set of states $N(s)$ from which the system can reach given set of states $R(s')$, uses the equation $N(s) = \exists_{s',i'}[T(s, s', i) \wedge R(s')]$. The computation of EXp can be done using the backward image computation. State

[4] Further, in case of a parallel implementation, such partitions may be physically on different processors. For now, we ignore this detail.

space partitioning into n disjoint parts induces a partitioning of the transition relation T into n^2 parts T_{jk} consisting of transitions from a state in partition j to a state in partition k. We can derive T_{jk} by conjoining T with the respective window functions as $T_{jk}(s, s', i) = w_j(s)w_k(s')T(s, s', i)$. Thus we can express the transition relation $T(s, s', i) = \bigvee_j \bigvee_k T_{jk}(s, s', i)$ as an induced disjunctive partitioning.

Figure 1 shows how to calculate EXp separately on each partitions. Here the

```
ComputeEX(Set R, Transition Relation T) {
  foreach (partition j)
    foreach (partition k)
      PreImg^jk(s) = ∃_{s',i}[T_jk(s, s', i) ∧ R_k(s')]
      reorder BDD PreImg^jk(s) from partition order k to order j
    end for
    N_j(s) = ⋁_k PreImg^jk(s)
  end for
  output N
}
```

Fig. 1. Backwards Image Computation with Partitioning

set R_j is the set of states that represent p in partition j, and the set N_j represents EXp in partition j which are computed by application of the transition relation $T_{jk}(s, s', i)$.

To compute the pre-image, the n^2 computations $T_{jk}(R_k)$ need to be performed, followed by n disjunctions as shown. Recall that when using a partitioned-BDD to represent the set of states, each partition is maintained separately in memory, under differing variable orders. It is therefore natural that the pre-image of states in partition k under the transitions leading to each partition j, i.e. the computation $T_{jk} \wedge (R_k)$, is performed in partition k. Each partition k thus computes states that potentially belong to every other partition. Subsequently the disjunction to obtain the pre-image lying with partition j, i.e. the computation of $\bigvee_k PreImg^{jk}$, is performed by partition j. As a consequence, the set $PreImg^{jk}$ needs to be transferred from partition k to partition j, when j and k differ.

We call these $n^2 - n$ computations as *cross-over image* computations, in the sense that the source and destination partitions are different. It must be emphasized that *Cross-over image computation is expensive* for various reasons: First, a quadratic number of image computations need to be performed as above and the BDDs need to be accessed from every partition. 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 secondary memory. Then, the BDD variable order of the computed image set must be changed from the order of the source partition to that of each of its target partitions, before the new states can be added to the reached set in the target. Reordering large

BDDs can be very expensive. Finally, there may also be other overhead, for eg., in the case of a parallel implementation there is the overhead of physically transmitting a large number of these BDDs over the network.

Thus the cost of a cross-over image computation may be significantly greater than that of a local image computation.

Next, we consider a simple partitioned model checking algorithm for the fix-point operators.

3.2 Partitioned Computation of fix-points

The classical fix-point algorithms for $E(pUq)$ and EGp as modified to use a partitioned data structure are illustrated in Fig. 2. Notice that these rely on

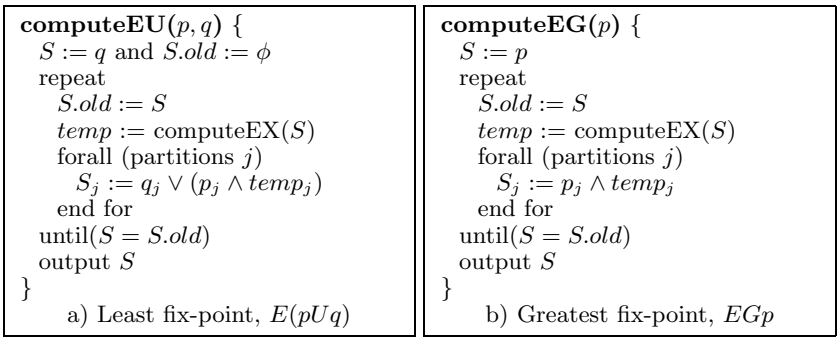

Fig. 2. Classical Model Checking of Fix-points in presence of Partitioning

the partitioned image computation and therefore perform one set of cross-over images in each iteration. In other words, each iteration until the fix-point is reached performs a number of image computations, quadratic in the number of partitions. As discussed in Section 3.1, this can get rather expensive.

In the next section, we present a model checking algorithm that localizes computation to individual partitions by postponing these cross-over image computations.

4 Partitioned Model Checking

In this section we present a new partitioned model checking algorithm which works by postponing cross-over image computations. When the design is defective and is falsified, this algorithm discovers bugs faster, by virtue of computations being localized to individual partitions. Even when the design is correct and is verified, this algorithm converges after fewer cross-over image computations. We show that in the worst case, this algorithm has at most as many cross-over image computations as the partitioned version of the classical algorithm, presented in the previous section.

Model checking of boolean connectives is well-known for the partitioned approach, so we will only describe the image and fix-point computations. It must however be mentioned that all boolean operations - conjunction, disjunction as well as negation - are local to individual partitions[5] and involve no interaction between them. Also, it suffices to consider the existential temporal operators EX, EG and EU, as these with the propositional connectives form a basis for all CTL formulae.

4.1 Image Computation

The main computation in the partitioned form of the classical model checking algorithm is image computation. As noted in the previous section, the computation of EXp from p comprises of n^2 image computations, re-orderings and state set transfers between partitions and this can get expensive. Even though our focus is on trying to avoid computing the entire image *at every step*, it may still be necessary to perform the full image computation in two cases – firstly, for the occasional cross-over images, and secondly, when the property is expressed in terms of the EX or AX operators. In this section, we look at some of the issues in computing the image.

We find that performing the cross-over images one partition at a time is memory intensive and often the intermediate BDDs get very large for many examples. Therefore, we advocate performing these cross-over image computations from each partition into many partitions at a time.[6]

In order to perform cross-over images efficiently, we maintain a *transfer manager* M. Given the set p, in order to compute EXp, each partition i computes the image $T_{ii}(p_i)$ which it keeps locally and the set of unowned states $U_i = T_{i\bar{i}}(p_i)$ which is communicated to the manager M. M uses the window functions w_j to calculate the sets $S_j = \bigvee_{i \neq j} U_i * w_j$ and then transmits the states S_j to partition j. Thus EXp is computed by doing $2n$ image computations and $2n$ transfers between partitions, although the number of re-orderings remains n^2.

It should be mentioned here that in a multiprocessor environment, such a manager can become a bottleneck, and should perhaps be dispensed with. But the point is that, in each partition, only a constant number of image computations be performed, rather than a number linear in the number of partitions. Thus the total number of image computations is linear rather than quadratic in the number of partitions.

We call the fraction $T_{ii}(p_i)$ that is computed locally using T_{ii} as the i^{th} projection of the local image EX_l. The rest of the images comprise the cross-over image EX_c. The algorithms to compute EX_l and EX_c are shown in Figure 3.

[5] This is an important consequence of window-based partitioning.

[6] Here, it must be noted that we address the case of verification using uniprocessor systems. The partitioned approach easily extends to distributed and parallel computing environments and our improvements are expected to scale accordingly.

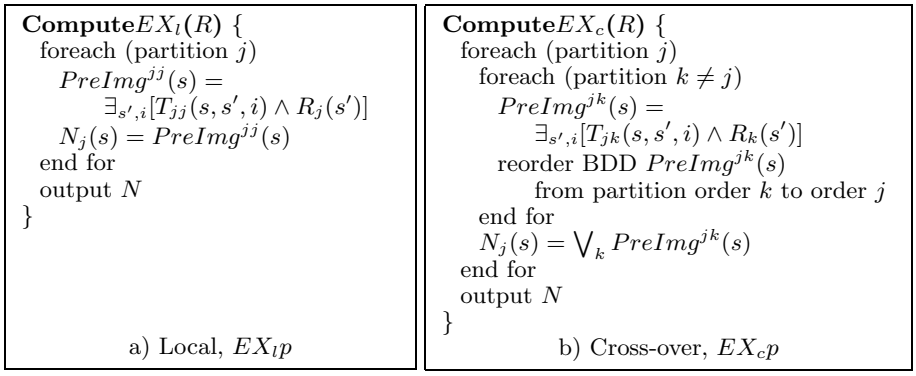

Fig. 3. Local and Cross-over Components of Image Computation with Partitioning

4.2 Fix-point computations

The main idea for model checking fix-points is that the computations can be significantly localized to individual partitions by postponing the cross-over image computations EX_c, which are then aggregated and performed infrequently. Accordingly, we define the fix-point operators in terms of two operations – local image computations EX_l and cross-over image computations EX_c, rather than the classical definition in terms of just the image computation operation, EX.

The algorithms for computing $E(pUq)$ and EGp are shown in Figure 4. The key idea is to create an under-approximation (resp. over-approximation) to EXp, which can be wholly calculated locally within individual partitions, so that the least (resp. greatest) fix-point computation can be localized.

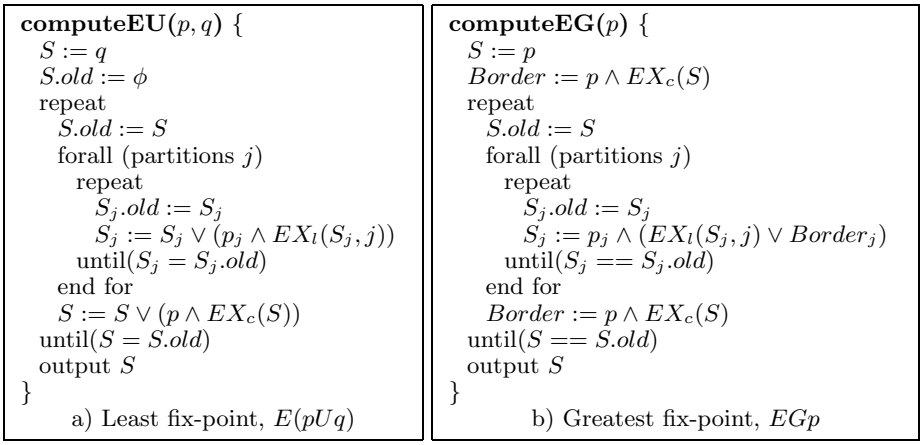

Fig. 4. Fix-point Computations localized by postponing cross-over images

Definition 1. *Each iteration of the outermost repeat-until loop in Algo.2 (shown in Fig. 2) and Algo.4 (resp. Fig. 4) is called a* phase *of the respective algorithm.*

From this definition, we note the following.

Lemma 1. *Every phase has one and only one cross-over image.*

We will show that Algo.4 terminates with the correct result and that the number of its phases is at most the number of phases in Algo.2. Since each such phase has precisely one cross-over image computation, we have that the number of cross-over images computed by the new algorithm is, in the worst case, no more than that for the existing algorithm. However in practice, the new algorithm computes a number of "local" images in each phase. Therefore it has fewer phases than the algorithm of Fig.2 almost always. As noted before, even a small reduction in the number of cross-over images can result in a drastic reduction in the total amount of time spent in cross-over images.

Theorem 1. *a)[3] The procedure computeEU of Fig 4a, 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 $E(pUq)$.*
b) The number of its phases does not exceed the number of phases for Algo.2a.

Proof: Let the set of states S at the end of the i^{th} phase be called S^i. The termination is guaranteed because the sequence of sets S^i is strictly monotonic increasing.

We first show the soundness of Algo.4a, i.e., at all times $S \models E(pUq)$. We show this by induction on the sets S^k. This clearly holds for any state in S^0, since every state in S^0 satisfies q and therefore $E(pUq)$. Assume 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 in the local image computation EX_l for some partition j or in the cross-over image computations EX_c. In either case, $s \models p$. It remains to show that s is the predecessor of a state that models $E(pUq)$. In the first case, such a state is in the same partition as s and in the second case, such a state exists in partition k such that s was added in the cross-over image computation from k to j. Thus in either case, s models $EX(E(pUq))$. Consequently, Algo.3a is sound.

Next, we show completeness, i.e., that every state of $E(pUq)$ is indeed in set S. For every state $s \models E(pUq)$, there exists a sequence of states $s_0, s_1, \ldots, s_k$ that has the smallest length $k \geq 0$ such that $s_0 = s$, $s_k \models q$, $\forall i < k : s_i \models p$ and $\forall i < k : s_i \in EX(s_{i+1})$. This sequence of states is called a *witness* for the inclusion of s in $E(pUq)$, and k is its *length*. 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, 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 $E(pUq)$. 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 $s \in EX(s_1) \subseteq EX(S^j)$. From the definition of S^j and Algo.4a, we have that $s \in S^{j+1}$, whereby $T^{i+1} \subseteq S^{j+1} \subseteq S$. By induction, this gives us $E(pUq) \subseteq S$.

This proves that Algo.4a terminates with the set $S = E(pUq)$. Notice that the set of states at the end of the k^{th} phase of Algo.2a is precisely T^i. As above, $\forall i, T^{i+1} \subseteq S^{j+1} \subseteq S^{i+1}$. Hence Algo.4a has at most as many phases as Algo.2a. ■

Before proving an analogous result for the greatest fix-point operator EGp, we briefly motivate its construction. As $EX_l p$ is a subset of EXp, the result of localizing the computation by performing repeated EX_l operations yields an under-approximation at every step. Since the greatest fix-point operator converges by a sequence of monotonically decreasing sets, under-approximation leads to some states being pruned too early and being lost for ever. States that may be incorrectly pruned early in the computation of EG comprises of states, each of which lies in a different partition from its predecessor, and can therefore be discovered only by performing the operation EX_c, which is the expensive component of image computation.

Algo.4b compensates for this by maintaining a set *Border*, which is the set of all states which have a successor in a different partition than themselves. This is, clearly an over-approximation to EX_c in each partition. This superset of EX_c is used to calculate a superset of EX at every image. This *Border* is updated only once in each phase, when each partition has reached a fix-point with respect to local images EX_l. These over-approximations are monotonically decreasing, and so the computed set eventually converges to the desired set EG.

We now prove the following theorem.

Theorem 2. *a) The procedure computeEG of Fig 4b, given the set of states corresponding to formula p as input , terminates with the output S being precisely the set of states that model the formula EGp.*
b) The number of its phases does not exceed the number of phases for Algo.2b.

Proof: Again, let the set of states S at the end of the i^{th} phase be called S^i. The termination is guaranteed because the sequence of sets S^i is strictly monotonic decreasing.

We first show the soundness of Algo.4b, i.e., the algorithm only deletes states which do not satisfy EGp. Note that a state can be deleted only in the two circumstances. The first is if it does not satisfy p and is deleted in the very beginning. We can therefore assume that all states under consideration satisfy p. The second way a state may be deleted is during some phase, when it is not a predecessor to any state in its own partition, and it is not on the Border, i.e., it has been determined previously that this state is not a predecessor to any state in another partition. Thus all successors to such a state satisfy $\neg p$, and therefore any deleted state is not in EGp.

Next we show completeness, i.e., the algorithm deletes all states that do not satisfy EGp. Consider a state $s \not\models EGp$. Then there exists a sequence of states $s_0, s_1, \ldots, s_k$, which is cycle-free that has the greatest length $k \geq 0$ such that $s_0 = s$, $s_k \models \neg p$, $\forall i < k : s_i \models p$ and $\forall i < k : s_i \in EX(s_{i+1})$. This sequence of states is called a *witness* for the exclusion of s from EGp, and k is its *length*. Now, let T^k be the set of states whose exclusion from EGp is witnessed by a

longest cycle-free path of length at most k. We prove by induction on k that $T^k \cap S^k = \phi$. In the base case, this trivially holds because $T^0 = \neg q$ and $S^0 = q$. Now, assume that $T^i \cap S^i = \phi$. 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 exclusion from EGp. The sequence $s_1, \ldots, s_{i+1}$ is a witness for s_1, therefore $s_1 \in T^i$, and therefore $s_1 \notin S^i$. In particular, there exists a smallest j so that s_1 was deleted in the j^{th} stage of the algorithm. Two cases arise, either both s_0 and s_1 are in the same partition or they are in different partitions. If they are in the same partition, then s_0 is deleted in the j^{th} stage also when a fix-point is computed locally in that partition. If they are in different partitions, then s_0 is in the border set for its partition, and is deleted from this border set at the end of the j^{th} stage because its last successor s_1 is deleted and no other successors can exist because this is the longest witness. Therefore s is deleted in the $j + 1^{th}$ stage, as required to be proved.

This proves that Algo.4b terminates with the set EGp. Notice that the set of states T^i is precisely the set of states deleted in phase i of Algo.2b. As above, states in T_i have all been deleted by the end of i phases of the algorithm. Hence Algo.4b has at most as many phases as Algo.2b. ∎

4.3 Comparison

In the worst case, Algo.4 requires at most as many phases as Algo.2. However, in practice, Algo.4 outperforms Algo.2, because when computing the least (resp greatest) fix-point by localizing computation to individual partitions, Algo.4 often discovers (resp. prunes) many more states than when performing just one image computation in each phase. Thus the postponement of cross-over images affords a significant benefit in overall faster convergence of the model checking algorithm, often reducing the number of phases.

We now analyze the benefit of reducing the number of cross-over images. Consider a simple model where the number of image computations performed is the same in each partition, say n. Further, assume the time for computing EX_l is L and that for EX_c is C.

Thus Algo.2 performs n phases, each with one computation of EX_l and EX_c, and incurs a total time $C_{old} = n*(L+C)$. Algo.4 needs potentially fewer phases, say $m \leq n$. Each such phase has one EX_c computation and a number of EX_l computations, for concreteness say there are $k \geq 1$ of them. This gives a total time $C_{new} = m * (k * L + C)$.

Recall from Section 3.1 that $C >> L$. Thus for reasonable k, the reduction in the number of cross-over images is directly reflected in the reduction of the total model checking time. Further, in the best case scenario, when $m * k = n$, the reduction may be by as much as a factor of k.

In practice a significant gain is observed, as borne out by the experimental results that are described in the next section.

5 Experimental Results

We implemented the algorithm of Fig. 3 on top of the CUDD package for BDDs using the VIS-2.0 verification environment, which is a state-of-the-art public domain BDD-based formal verification package. We have chosen VIS for its Verilog support and its powerful OBDD-package (i.e. CUDD [8]). As our techniques affect only the BDD-data structures and algorithms, they can – with moderate effort – be implemented in other packages as well. These techniques work with any method of image computation; all experiments here are conducted using the IWLS95 method.

We use the partitioning scheme detailed in [3] for performing reachability analysis. Once the reachable states are computed, the model checking algorithms use the same partitions created during reachability analysis.

Benchmarks
We chose the public domain circuits and their model checking properties from the VIS-Verilog [9] benchmark suite. For sake of brevity, results are omitted for some of the smaller examples.

Results
We notice that crossover image computation is indeed a bottleneck in verification. On a uniprocessor machine, in the VIS-Verilog benchmark suite, there are examples where the program runs out of memory while performing the crossover images. Thus a reduction in the number and frequency of such cross-over images is critical for the full utilization of computing resources in a multi-processor environment.

The run-times for our sequential implementation are shown in Table 1. The first column of Table 1 has the name of the circuit and the property being checked. This is followed by the data for cross-over image computation. Firstly, the number of cross-over images is shown for the Naive algorithm, labeled *Old* and for our proposed algorithm, labeled *New*. The next two columns show the respective time taken. This is followed by the speedup achieved by the proposed algorithm over the older one. The last two columns show the total time taken by the model checking, after reachability has finished.

Experimentally, the proposed algorithm converges faster, both in terms of total time, as well as in terms of number of expensive cross-over image computations that are performed. Further, the time taken by cross-over images as a percentage of total time is reduced. These are demonstrated in the table that follows. In numerous cases (e.g. s1269, soap, ghg, etc.), we find that total cross-over image time is reduced by two orders of magnitude or more.

Using the data in Table 1, we can compare the total time taken for model checking by the two methods. Notice that the proposed algorithm reduces, often dramatically, the number of cross-over images and the proportion of the total time that is spent in doing them. In almost all the examples, this leads to a direct improvement in the total time.

Circuit_ Property	Cross-over images					Model Checking time(sec)	
	Number		Time (s)				
	Old	New	Old	New	Speedup	Old	New
bpbs	4	1	24	1	24	398	313
gcd_1	15	7	19.11	.7	27	68.97	108.07
gcd_2	15	7	18.27	.16	114	27.56	9.06
gcd_3	10	8	37.13	4.29	8.6	134.65	56.32
gcd_4	10	8	37.41	3.44	11	108.76	42.11
gcd_5	11	9	37.13	46.3	0.8	107.31	92.19
gcd_6	12	9	42.96	3.79	11	121.66	53.69
gcd_7	13	9	42.98	3.99	11	132.7	50.51
gcd_8	14	9	35.68	1.41	25	128.04	48.94
gcd_9	15	9	31.77	0.91	35	119.63	48.04
gcd_10	16	9	28.72	0.57	50	111.89	46.47
ghg	9367	6	166.12	0.15	1107	280.75	27.31
idu32_1	3	3	12.35	89.96	0.13	294.49	406.61
idu32_2	2	2	0.07	0.02	3.5	0.12	0.03
idu32_3	3	3	0.07	0.02	3.5	0.06	0.02
idu32_4	8	4	0.61	0.02	30	0.82	0.1
idu32_5	8	4	0.61	0.03	20	0.83	0.11
idu32_6	7	2	0.83	0.02	41	1.27	0.1
idu32_7	7	4	1.31	0.02	65	2.11	0.22
idu32_8	8	4	13.63	0.03	454	14.15	0.28
idu32_9	8	4	0.38	0.02	19	0.52	0.05
idu32_10	23	9	0.58	0.04	14	0.8	0.15
luckySeven	64	35	80.12	55.89	1.4	114.64	82.03
nosel	7	3	106.01	10.2	10	270.18	130.87
product	1	1	3	3	1	1798	418
s1269b_1	1	1	9.42	0.01	942	15	13
s1269b_2	8	8	67	1.01	67	93	1
soap_44	53	5	592.09	1.2	493	714.81	28.24
soap_45	80	8	106.76	1.86	57	224.19	104.11
soap_46	53	5	92.9	1.14	81	187.79	28.76
soap_47	52	5	41.87	1.11	37	94.89	31.83
soap_48	60	5	42.3	0.76	55	98.91	56.41
soap_49	79	9	94.68	1.61	58	207.18	73.78
soap_50	60	5	199.6	1.05	190	299.4	22.9
sppint2_1	5	4	86.26	45.06	1.9	100.39	58.26
sppint2_4	1	1	2.8	0.01	280	3.06	2.82
sppint2_5	7	3	4.4	0.01	440	4.94	0.75
sppint2_6	5	3	0.2	0.13	1.5	0.68	0.28
sppint2_7	16	6	4.23	0.7	6	24.66	2.27
sppint2_8	5	4	1.22	0.37	3.3	1.87	0.71
sppint2_9	14	6	1.29	0.74	1.7	5.01	1.81
sppint2_10	5	4	1.31	0.17	7.7	1.83	0.32
two	38	24	30.6	18.8	1.6	46	28
usb_phy_1	49	23	16	19	0.8	43	29
usb_phy_3	40	19	108.51	11.83	9	24.89	28.97
usb_phy_4	21	11	5.6	2.32	2.4	12.01	8.26
usb_phy_6	5	5	0.97	1.05	0.9	2	2.99
usb_phy_7	39	17	10.96	2.14	5	24.32	8.72

Table 1. Comparison of existing and proposed algorithms for partitioned model checking CTL properties on circuits in the VIS Verilog benchmark suite. For each circuit and property, the first pair of columns shows the number of inter-partition cross-over images performed by the two methods, the second set shows the time required for these cross-over images, and the speedup achieved by the new method and the final set shows the total model checking time.

6 Conclusion

We have presented a model checking algorithm in the presence of state space partitioning, that aggregates and postpones cross-over image computations, allowing for significant localization of image computations. This is also found in practice to reduce the number of iterations in fix-point computations.

If during state space traversal, each partition requires many steps of image computation to reach a local fix-point, then the proposed algorithm shows significant gain (which is proportional to the depth of the fix-point). In the worst case, this method would be identical to the naive one, with strict alternation between localized (EX_l) and cross-over (EX_c) image operations in every fix-point calculation. However, this is extremely unlikely because it corresponds to a case where every "path" corresponding to a formula comprises of states *each* of which lies in a different partition from its predecessor. This does not happen in practice when partitioning is done properly.

Our experiments have been conducted on uniprocessor machines, but this algorithm can be easily parallelized and we believe its benefits would scale to an implementation in a multi-processor environment.

We believe this algorithm can be generalized to more expressive logics like the full μ-calculus with a few modifications. A parallel form of the proposed algorithm can also provide better resource usage than existing distributed model checking algorithms.

References

1. 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.
2. Orna Grumberg, Tamir Heyman, and Assaf Schuster. Distributed symbolic model checking for μ-calculus. In *Computer Aided Verification*, pages 350–362, 2001.
3. S. Iyer, D. Sahoo, C. Stangier, A. Narayan, and J. Jain. Improved symbolic Verification Using Partitioning Techniques. In *Proc. of CHARME 2003*, volume 2860 of *Lecture Notes in Computer Science*, 2003.
4. J. Jain. On analysis of boolean functions. *Ph.D Dissertation, Dept. of Electrical and Computer Engineering, The University of Texas at Austin*, 1993.
5. Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
6. 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.
7. 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.
8. Fabio Somenzi. CUDD: CU Decision Diagram Package ftp://vlsi.colorado.edu/pub, 2001.
9. VIS. Vis verilog benchmarks http://vlsi.colorado.edu/ vis/, 2001.