BDD-BASED CONJUNCTIVE DECOMPOSITION USING A GENETIC ALGORITHM AND DEPENDENT VARIABLE AFFINITY

Lun Li, Mitchell A. Thornton, Stephen Szygenda
Department of Computer Science and Engineering, Southern Methodist University
{lli,mitch,szygenda}@engr.smu.edu

ABSTRACT

Decomposition is an important strategy in synthesis and verification. BDD-based conjunctive decomposition has been successfully used in verification, especially in image computations in Finite State Machine (FSM) state-space traversals. Conjunction decomposition also has applications in other areas of CAD. A "two-step" algorithm for representing a large function as a conjunctive decomposition of BDDs is described where a Genetic Algorithm (GA) approach for ordering individual bit functions is given followed by an affinity-based clustering technique. Experimental results are given that show the effectiveness of the algorithm.

1. INTRODUCTION

Binary Decision Diagrams (BDD) can be very efficient data structures in synthesis and verification applications. However, the size of BDD's that arise in synthesis and verification computations continues to pose a problem in many cases. Various strategies have been proposed to avoid size explosion. Decomposition is an important strategy to reducing the size of large BDDs.

Decomposition of a BDD is closely related to finding efficient partitioned representations of a given Boolean function. Partitioned representations may be derived in the process of building a BDD or by decomposing a given BDD. The former approach can yield compact decompositions when some structure information, such as an underlying network, is provided. Our decomposition technique belongs to the former approach. We noticed that some Boolean functions, like Characteristic functions, can be represented as product form from circuit structure, such as $F = D_1 \wedge D_2 \wedge \cdots \wedge D_n$, where each D_j is a product term. We are targeting at this type of Boolean functions and trying to find a good way to cluster some of smaller product term. The motivation for the decomposition of this type of Boolean function comes from the following aspect: forming a monolithic BDD for such F is impractical when the circuit is complicated; however, working on every single term directly without clustering causes too many iterations and is time consuming. Past research [1] has shown that there are some tradeoffs between the number of product terms and the size of each product term. Clustering some of the product terms and also keeping the size of each clustered function within a reasonable threshold reduces the number of iterations and improves efficiency.

A two-step process strategy is deployed to determine an efficient partition. Original product terms are ordered according to the correlations of their supporting variables first. The purpose of ordering is to keep the product terms that have a similar support set close so that few new variables are introduced by clustering. Secondly, ordered product terms are conjoined to form clustered product terms whose size are within a given threshold.

A Genetic Algorithm (GA) based ordering algorithm is presented in the paper to address the ordering problem. The overall objective is to reduce the shared size (to occupy less storage space) and the individual size (for easier manipulation) of the decomposed set as compared to the original BDD size as well as the number of product terms (few iterations).

The rest of the paper is organized as follows. In section 2, we review background material and related work. Section 3 describes our GA approach for conjunctive decomposition. Section 4 describes experimental results and Section 5 concludes.

2. PRELIMINARIES AND RELATED WORK

2.1 Characteristic Function and BDD

Notation: A Boolean network N with n primary inputs $X = \{x_1,...x_n\}$ and m primary outputs $Z = \{z_1,...z_m\}$ can be viewed as a set of m Boolean functions defined as f:$\mathbf{B}^n \rightarrow \mathbf{B}^m$. Let λ represent the set $\{\lambda_1(X),...\lambda_m(X)\}$ where each λ_i is an output function. A characteristic function of Boolean network N is defined as a Boolean function $C(X, Z, \lambda)$ such that $C(X, Z, \lambda)=1 \Leftrightarrow z_i \equiv \lambda_i(X)$. In other words, a characteristic function maps every valid input/output combination to '1', and every invalid combination to '0'. Computationally, the characteristic function can be derived by the following formula [2]:

$$C(X, Z, \lambda)=\prod_{i=1}^{m} C_i(z_i, X, \lambda_i)=\prod_{i=1}^{m}(z_i \equiv \lambda_i(X)) \quad (1)$$

where $(a \equiv b)$ corresponds to $(ab+\bar{a}\bar{b})$, $C_i(X, z_i, \lambda_i)$ is also called a *bit function*.

BDD: BDDs are data structures used to represent Boolean functions. Bryant [3] introduced the concept of

reduced, ordered BDDs (ROBDDs) along with a set of efficient operators for their manipulation and proved the canonicity property of ROBDDs. In the rest of the paper, BDD refers to a ROBDD.

Decomposition: An early systematic approach to functional decomposition was proposed by Ashenhurst [4] and Curtis [5]. According to decomposition, a Boolean function $f(X)$ can be represented as $f(X) = h(g(X_1), X_2)$ where $X_1 \cup X_2 = X$. Here X_1 is referred to as a *bound* set and X_2 is a *free* set.

Bi-decomposition [6] is a restricted class of functional decompositions that has the form $f(X) = g_1(X_1) \odot g_1(X_2)$ where $X_1 \cup X_2 = X$ and $\odot$ stands for any binary Boolean operation. The functional decomposition we are interested in can be seen as an extension of a conjunctive bi-decomposition which has the form of $f(X) = g_1(X_1) \wedge g_2(X_2) \wedge \cdots \wedge g_n(X_n)$, where each $g_i(X_i)$ is a product term and $X_1 \cup X_2 ... \cup X_n = X$.

Decomposition of Characteristic Functions: For a design that has a large number of outputs and/or inputs, it is impractical to build a monolithic characteristic function BDD for the entire network. However, clustering allows the characteristic function to be written as:

$$C(X, Z, \lambda) = \prod_{i=1}^{k} \hat{C}_i(z_i, X, \lambda_i) \qquad (2)$$

where $k \le m$, and each cluster $\hat{C}_i$ is the conjunction of some proper subset of C_i's, called a *clustered function*.

The conjunctive decomposition problem consists of deciding the value of *k* and determining which bit functions are to be clustered together.

2.2 Related Work

Previous work in conjunctive decomposition mostly focused on sequential logic, especially for image computation [1][8][9]. Moon et al. [7] proposed a generalized strategy, known as FMCAD. The ordering algorithm of FMCAD is based on computing the Bordered Block Triangular form of the dependence matrix. The bit functions are clustered according to the affinity between them.

3. CONJUNCTIVE DECOMPOSITION

The two-step conjunctive decomposition presented in the paper consists of a GA-based ordering algorithm and an affinity based clustering algorithm. The overall objective is to reduce the shared size, the individual size as well as the number of product terms.

3.1 GA Based Ordering Algorithm

A genetic algorithm emulates the metaphor of natural biological evolution to solve optimization problems. Genetic algorithms generally utilize the following steps. a) Initialize population: find a collection of potential solutions to the problem, also called current population. b) Create offspring: produce a new population through the application of genetic operations on selected members of the current generation. c) Evaluate fitness: evaluate the quality of the solution in the new generation. d) Apply selection: select solutions that will survive to become parents of the next generation based on their quality of solution to the problem. In this way, it is more likely that desirable characteristics are inherited by the offspring solutions. e) This cycle repeats until some threshold or stopping criterion is met.

3.1.1 Problem Representation and Initial Population

The GA starts with mapping a problem into a set of chromosome representations used within GA. Since we are interested in the order of functions and their support set, a preprocessing step converts the order information into a chromosome. Each gene (denoted as π) in chromosome is an index of a function. Any ordered set of functions could be a solution, so an initial population is generated by randomly mutating the order of the genes in the chromosome

3.1.2 Fitness function and Selection

The fitness function discussed here is based on the dependency matrix of chromosome. The *dependence matrix* defined in [7] is used for an ordered set of functions. The *dependence matrix* of a set of *m* single-output functions $(f_1, ..., f_m)$ depending on *n* variables $x_1, ..., x_n$ is a matrix *D* with *m* rows (corresponding to *m* functions) and *n* columns (corresponding to *n* variables) such that $d_{i,j} = 1$ if function f_i depends on variable x_j, and $d_{i,j} = 0$ otherwise. We can define the dependency matrix of a chromosome in the same way. The size of a BDD depends on the number of variables and the functions it represents. Smaller BDDs usually can be produced by conjoining two product terms that have a similar support set because few new variables are introduced. Based on the above observation, the *normalized active lifetime* of the variables in matrix *D* is given in [7] by

$$\alpha = \frac{\sum_{i=1}^{n} (h_j - l_j + 1)}{n \cdot m}$$

where $l_j (h_j)$ be the smallest (largest) index i in column j such that $d_{i,j} = 1$ respectively.

$h_j - l_j$ gives out a quantity measure on sharing the variable in column j stays. The normalized active lifetime measure how closely that the product terms stays based on their support variables. The objective of ordering becomes to lower the normalized average active lifetime for a given matrix by manipulating the order of columns.

Because the objective of ordering is to minimize α, we use α as fitness function.

The selection is performed by linear ranking selection, i.e the probability that one element is chosen is proportional to its fitness. The size of the population is constant after each generation. Additionally, some of the best elements of the old population are inherited in the new generation. This strategy guarantees that the best element never gets lost and a fast convergence is obtained. Genetic algorithm practice has shown that this method is usually advantageous [10].

3.1.3 Genetic Operators

Two genetic operators are used in the algorithm: Partially Matched Crossover (PMX) and a random Mutation (MUT).

PMX generates two children from two parents. The parents are selected by method described above. The operator chooses two cut positions at random. Notice that a simple exchange of the parts between the cut positions would often produce invalid solutions. A validation procedure has to be executed after exchange. The detailed procedure for PMX is given below.

Construct the children by choosing the part between the cut positions from one parent and preserve the position and order of as many variables as possible from the second parent. For example, $p_1 = \pi(1,2,3,4)$ and $p_2 = \pi(2,4,3,1)$ are the parents while $i_1 = 1$ and $i_2 = 3$ are the two cut positions. The resulting children before the application of the validation procedure are $c_1' = \pi(1,4,3,4)$ and $c_2' = \pi(2,2,3,1)$. The validation procedure goes through the elements between the cut positions and restores the ordering. This results in the two valid children $c_1 = \pi(1,4,3,2)$ and $c_2 = \pi(4,2,3,1)$.

MUT selects a parent by the method described above and randomly choose two positions. Two values at these two positions are exchanged.

3.1.4 Algorithm

The genetic algorithm is outlined as follows:

a) The initial population is generated using an arbitrary order as the first individual in current generation and by applying MUT to create other populations in current generation. b) Genetic operators are selected randomly according to a given probability. The selected operator is applied to the selected parent (MUT) or parents (PMX). The better half of the population is inherited in each iteration without modification. c) The new generation is updated according to their fitness. d) The algorithm stops if no improvement is obtained for certain number of iterations.

```
Genetic algorithm(){
  Generate_initial_population();
  do{
    for( each child  i ){
       j =linear_ranking_selection();
      randomly_select_method;
        case MUT: child( i ) = MUT(parent  j );
        case PMX:  k =linear_ranking_selection();
           child( i , i+1 ) = PMX(parent j , k );
    }
    update_population();
  }until(no improvement iterations > threshold)
}
```

3.2 Affinity Based Clustering Algorithm

The ordering algorithm described above rearranges product terms so that product terms sharing more variables stay as closely as possible. The next step is clustering some of the small product terms to a big one while the BBD size of the clustered product terms is within a reasonable threshold. The motivation for clustering is to reduce iterations and improves efficiency in computation.

Affinity defines the similarity of support sets of two functions [7].

Let d_i be the *i*-th row of the dependency matrix. Let $d_i \times d_j$ designate the inner product of d_i, d_j and $|d|$ be the width of matrix. The *affinity*, β_{ij} of vector d_i and d_j is defined as:

$$\beta_{ij} = \frac{d_i \times d_j}{|d|}$$

The affinity based clustering algorithm is now discussed. The affinities for pairs of adjacent product terms are computed as above, and then the pair with the highest affinity is merged. As in the sequential approach, merging is accepted only if the resulting BDD size does not exceed the cluster threshold size. If the threshold is exceeded, a barrier is introduced between the two terms. The process is then recursively applied to the two subsets of the rows above and below the barrier. If the size of the conjunction BDD is below the threshold, the algorithm computes the affinity for the new function and its neighbors and then selects a new pair with the highest affinity. The terminal case of the recursion occurs when only one function is left.

Affinity based approach sacrifices complexity to achieve more compact BDDs.

4. EXPERIMENTAL RESULTS

In order to evaluate the GA approach, we ran the conjunctive decomposition algorithm. The benchmarks we used are from the ISCAS'89 and LGSYNTH'91 suites. The algorithm is implemented using the CUDD BDD package [11]. All experiments are carried out on a 733MHz HP PC running Linux.

We compared the GA approach with FMCAD. FMCAD is implemented in VIS 2.0 [12]. In ISCAS'89 set, we ignore the benchmarks larger than s3271 since both methods can not handle these benchmarks. Dynamic BDD variable reordering is enabled in both approaches. A time limit of 7200 seconds is used. A threshold, 5000, is set to limit the number of nodes for each partitioned BDD. The two parameters we measured for decomposition are the number of clusters (in the column labeled "clusters") and the total number of BDD nodes (in the column labeled nodes). Shared nodes among various clusters only count once. Table 1 shows the result. Compared with FMCAD, our GA approach has a better result in term of memory requirements in most cases. We don't compare time in the table 1 since the clustering time information for FMCAD is not available in VIS.

TABLE 1. COMPARISON OF DECOMPOSITION

Circuit	FMCAD		GA		Improve on nodes
	clusters	nodes	clusters	nodes	
sbc	2	3163	2	1545	51%
clma	2	4336	2	3270	24%
clmb	2	4780	2	3270	31%
mm9a	2	3001	1	1526	49%
mm9b	2	4304	2	3130	27%
mm30a	4	6846	3	3369	50%
s1512	2	2097	3	1665	20%
s1269	4	9664	9	8069	16%
s4863	30	81660	29	70466	13%
s3271	10	10522	8	6668	36%

5. CONCLUSIONS

We presented a Genetic Algorithm for the conjunctive decomposition of large combinational functions and discussed its' application. This technique is a practical alternative as compared to existing methods and the experimental results show that the method performs well.

In sequential logic verification, FSM traversals based on transition relations is a key component. The transition relation is a type of characteristic function. Our conjunctive decomposition technique can be applied to image computation in FSM traversal.

Optimization of Timed Shannon Circuits [13] is another application for our conjunctive decomposition. Timed Shannon Circuits are motivated by the need to automatically synthesize circuits with low power dissipation characteristics. One difficulty of the Timed Shannon Circuit approach is that each output is a separate circuit. Thus, detecting the sharing of gates between circuits needs extra work. An improvement was proposed by using characteristic function in [14]. Our conjunctive decomposition could be used to improve the scalability of the method in [14].

6. REFERENCES

[1] R. Ranjan, A. Aziz, B. Plessier, C. Pixley, and R. Brayton, "Efficient BDD algorithms for FSM synthesis and verification.", *In Proc. of IWLS*, Lake Tahoe, 1995.

[2] Shi-Yu Huang, Kwang-Ting Chang, *Formal equivalence checking and design debugging*, Kluwer Academic Publishers, Boston, 1998

[3] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Computers*, vol. 35, pp. 677–691, Aug. 1986.

[4] R. Ashenhurst, "The decomposition of switching functions," *Technical report, Bell Laboratories*, BL-1(11), 1952, pp. 541-602.

[5] H. Curtis, "A New Approach to the Design of Switching Functions", *Van Nostrand*, Princeton, N.J. 1962.

[6] D. Bochman, F. Dresig, and B. Steinbach, "A new decomposition method for multilevel circuit design", in *In Proc. of Eur. DAC*, 1991, pp. 374-377.

[7] I. Moon and F. Somenzi, "Border-block triangular form and conjunction schedule in image computation", *In Proc. FMCAD,* vol. 1954 of LNCS, Nov. 2000, pp. 73–90.

[8] P. Chauhan, E. Clarke, S. Jha, J. Kukula, H. Veith, and D. Wang, "Using combinatorial optimization methods for quantification scheduling", *In Proc. CHARME,* Sep. 2001.

[9] R. Hojati, S. C. Krishnan, and R. K. Brayton. "Early Quantification and Partitioned Transition Relations", *In Proc. ICCD*, pp. 12-19, Austin, TX, Oct. 1996.

[10] R. Drechsler, *Evolutionary Algorithms for VLSI CAD*, Kluwer Academic Publishers, 1998.

[11] F. Somenzi et al. CUDD: University of Colorado Decision Diagram Package. http://vlsi.colorado.edu/~fabio/CUDD/.

[12] R. K. Brayton et al. VIS: A system for verification and synthesis. http://vlsi.colorado.edu/vis/.

[13] L. Lavagno, P. McGeer, A. Saldanha, and A.L. Sangiovanni-Vincentelli. "Timed shannon circuits: A power-efficient design style and synthesis tool", *In Proc. DAC*, pp 254–260, 1995.

[14] M.A. Thornton, R. Drechsler, D. M Miller, “Multi-output Timed Shannon Circuits”, *In Proc. ISVLSI*, pp. 47-52, 2002.