Efficient Implementation of Hashing in BDD Package

Miloš M. Radmanović[1]

Abstract – **The efficient manipulation of Boolean functions is an important component of CAD tasks. Binary Decision Diagram (BDD) packages have always been sensitive to hash design. This paper describes the use of various hash keys in implementation of basic BDD algorithms. The various hash key strategies have been implemented within a BDD package. In this paper, I experimentally performed a detailed analysis of hashing in BDD package using BDD algorithms computation and direct performance monitoring. The ultimate goal is to exceed the computation performance for various BDD algorithms of BDD packages.**

Keywords – **Boolean functions, BDD package algorithms,hash key strategies.**

I. INTRODUCTION

Binary Decision Diagrams (BDDs) have become the dominant data structure for representing Boolean functions in computer-aided-design (CAD)applications [1]. They are widely used in various areas of CAD: logic synthesis, testing, simulation, design and simulation verification [2]. [3], [4].

In practice, the memory required by large BDDs is typically the limiting factor for CAD tools. In same cases, especially with various BDD algorithms, run time is also a limiting factor. Therefore, considerable research has been intended for more efficient BDD algorithms implementation [4], [5], [6], [7], [8].

Various BDD algorithms are usually built on the top of a BDD package. Many BDD package implementations have been built in a variety of programming languages (C, C++, Lisp, Java) [9], [10], [11], [12]. The most of BDD packages are freely available in public domains (on the Web). The packages CAL (UC Berkeley, USA), CMU (CMU/ATT, USA), and CUDD (Boulder, CO, USA)are famous.The packages IBM (IBM Watson, USA), Tiger (Bull/DEC/Xorix, USA) and TUD (Darmstadt, Germany) are popular [13]. The choice of the BDD package might be guided by the following aspects: functionality, software interface, robustness, reliability, portability, support and performance. Naturally, performance is of concern [14], [15], [16], [17], [18], [19]. But, from performance comparative studies, BDD packages behave similarly as long as they are not put to theextreme [13]. They are many parameters that influence the run-time of the BDD package, for example: programming language, software and hardware platform, BDD node structure, type of garbage collection, unique and operation hash table size, and hash keystrategiesof tables [1], [20], [21].

The concept of hash miss complexity of the BDD package has been introduced by paper [21]. It also has been shown that run-time of BDD algorithms depends on BDD node and operations hashing strategies of the BDD package.

This paper describes the use of various hash keys strategies in implementation of a BDD package and their influence on the run-time of the basic BDD algorithms on the top of a BDD package. The various hash key strategies have been implemented within a BDD package.I build a simple software tool for evaluating algorithms depending on hash key of the BDD package. Using this software tool, I experimentally performed a detailed analysis of hashing in BDD package. The analysis of hashing usesbasic BDD algorithms computation and direct performance monitoring of the BDD package.

This paper is organized as follows: Section 2 shortly introduces the BDD representation of Boolean function.Section 3 describes the BDD package technique includingbasic BDD algorithms. Section 4presents the hash key strategies of the BDD packages. Section 5shows experimental analysis of hashing in BDD package and gives some examples of using various hash keys. Some concluding remarks end the paper.

II. BINARY DECISION DIAGRAM

The Binary Decision Diagram (BDD) is a graphic representation of the Shannon's expansion of a Boolean function. The concept of BDD was first proposed by Lee in 1959 [22]. It has been developed into a useful data structure by Akers [23] and later by Bryant [2], who introduced a concept of reducedordered BDD (ROBDD), and a set of efficientoperators for their manipulation.

The BDD is directed acyclic graph that contain non-terminal nodes, two terminal nodes, and edges. Non-terminal nodes are labeled with variables x_i and have two outgoing edges. Outgoing edges are labeled '0' and '1' values of variable x_i. Terminal nodes contain values '0' and '1'. The truth table entry of Boolean function labels edges from the root node to the corresponding terminal node. An example of the BDD representation for the function $f(x_1, x_2, x_3) = x_1 + x_2\overline{x}_3$ is shown in Figure 1.

The strength of BDDs is that they can represent Boolean function data with high level of redundancy in a compact form, as long as the data is encoded in such way that the redundancy is exposed.

It is well known that the size of the BDD for a given Boolean function depends on the variable order chosen for the function.

Other variants of BDD have been developed to address some of the problems with BDDs. Boolean expression diagrams (BEDs) are a generalization of BDDs that can represent any Boolean circuit in linear space.Zero-suppressed binary decision diagrams (ZDDs) are similar to BDDs but use a different reduction rule. ZDDs are generally more efficient for representing sparse sets. Binary moment diagrams

[1]MilošM. Radmanović is with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Niš, Serbia, E-mail: milos.radmanovic@gmail.com

(BMDs) are a generalization of BDDs to linear functions over other domains than Booleans, such as integers or real numbers [24].

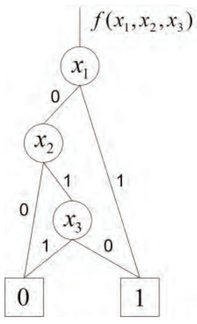

Fig. 1. BDD for the function $f(x_1, x_2, x_3) = x_1 + x_2\bar{x}_3$.

III. BDD Package Technique

In this section I describedthree data structures that are fundamental to most BDD packages.

The main principles of BDD package implementation are [2], [3], [13]:

- BDD nodes are data structures that contain a variable identifier, “1” and “0” children pointers and “next” pointer that links nodes together belonging to the same collision chain in the unique table.
- BDD nodes are kept in a unique table.
- BDD operations are sped up by using an operation table.
- The order of variables may be changed to reduce the total number of nodes.
- Recycling of nodes is easily implemented by keeping a reference count for each node.
- BDD packages use breadth-first traversal of the directed acyclic graphs.

It is obvious what needs to be stored for each node in a BDD package data structure: the variable field that labels the node and two edges field to point children nodes. There should obviously also be a field to identifier non terminal nodes from terminals. BDD traversal uses a “visited” field.Garbage collection might use a reference counter field.Chaining of nodes in a hash table needs a “next” field. The decisions made in defining the basic node data structure have an immediate impact on several related aspects. All this data requires certain memory space and needs to be packed into a node structure or object [14].

The unique table maps a triple of (*v*, *G*, *H*) of BDD node, where *v* is the variable identifier, *G* is the node connected to the "1" edge, and *H* is the node connected to the "0" edge. Each node in the BDD has an entry in the unique table. Before a new node is added to the BDD, a lookup in the unique table determines if the node for that function already exists [3], [14].

The If-Then-Else or *ITE* operator forms the core of the BDD package. *ITE* is a Boolean function defined for inputs *F*, *G* and *H*which computes: “If*F* Then *G* Else *H*”. ITE operations can be used to implement all two-variable Boolean operations.The operation table records results of operations on BDDs. Typically it stores the fact that $R = op(F,G,H)$, i.e., the BDD *R*is the result of the operation *op* applied to three BDD arguments *F*, *G*, and *H*, in a operation table [3], [14].

Garbage collection invalidates entries that refer to dead nodes. Most BDD packages use a reference counting garbage collector.

Originally designed to support the standard operations on Boolean functions, BDD packages have grow to build in various algorithms. Basic BDD algorithms are based on a recursive formulation that leads to a depth-first traversal of the directed acyclic graphs representing the BDD. The depth-first traversal visits the nodes of the BDD on a path-by-path basis. The large in-degree of a typical BDD node makes it is impossible to assign contiguous memory locations for the BDD nodes along a path. Therefore, the recursive depth-first traversal leads to an extremely disorderly memory access pattern [5].

The most of BDD algorithms are the result ofsome other basic BDD algorithms yet to be completed. A comprehensiveset of BDD manipulation algorithms are implemented usingthe above techniques. The most common BDD algorithms are, for example,BDD construction,BDDsaddition,and BDD printing [2], [3], [13].

IV. Hash Key Strategies of the BDD Packages

The unique table is usually implemented as a hash table [25]. For greater flexibility, open hashing with collision chains is normally used. The collision lists can be kept sorted [1] to reduce the number of memory accesses required for a lookup on average. The unique table might be divided in subtables, one for each variable [14].

The size of the hash table is either a prime number or a power of two. For hashed insertion, the hash table is used in a two-way associative manner: the hash function calculates an index *k* (hash key) for a node (v, G, H) and the node is found either at *k* hash entry or in the collision list. The hash key should obviously be composed of the memory position of the node and its successors or by defining a signature for each node which consists of the variable associated with that node and a pseudo random number [1].

It should be noticed that the hash key calculation of the unique table can be defined as follows:

$$UTH(v,G,H) = (m(G)\ op\ m(H)) \bmod hs \quad (1)$$

where *m*(*G*) and *m*(*H*) denote memory addresses of the nodes *G* and *H*, *hs* denotes hash table size, and *op* represents some logic or arithmetic operation.

The operation table is usually implemented as a unique table [25]. The hash function calculates an index *k* (hash key) for anoperation (*F*, *G*, *H*) and the operation is found either at *k* hash entry or in the collision list.

It should be noticed that the hash key calculation of the operation table can be defined as follows:

$$OTH(F,G,H) = (m(F)\ op\ m(G)\ op\ m(H)) \bmod hs \quad (2)$$

where *m*(*F*), *m*(*G*) and *m*(*H*) denote memory addresses of the nodes *F*, *G* and *H*, *hs* denotes hash table size, and symbol *op* represents some logic or arithmetic operation.

Experimentally, it is proved that efficient operation for symbol *op* can be + or $\oplus$ [1], [8], [9], [13], [14].

From previous consideration, it is evident that various hash key strategies can be defined:

(1) $UTH(v,G,H) = (m(G) + m(H)) \bmod hs$

$OTH(F,G,H) = (m(F) + m(G) + m(H)) \bmod hs$

(2) $UTH(v,G,H) = (m(G) + m(H)) \bmod hs$

$OTH(F,G,H) = (m(F) \oplus m(G) \oplus m(H)) \bmod hs$

(3) $UTH(v,G,H) = (m(G) \oplus m(H)) \bmod hs$

$OTH(F,G,H) = (m(F) + m(G) + m(H)) \bmod hs$

(4) $UTH(v,G,H) = (m(G) \oplus m(H)) \bmod hs$

$OTH(F,G,H) = (m(F) \oplus m(G) \oplus m(H)) \bmod hs$

V. Experimental Results

It is interesting to consider proposed hashing strategies that could be applied to BDD package and tested on basic BDD algorithms.

Below I give tables of BDD package performanceusing different basic BDD algorithms. I performed the testing on a PC Pentium IV on 2,66 GHz with 4GB of RAM (MS Windows 7). The memory usage for all tests was limited to 2 GB.The size of unique and operation table was limited to 65536 entries. The garbage collection was activated if the total number of BDD nodes and operations in memory became greater than 524288. All benchmarks were used in the Espresso-mv or pla format [31].

Table 1, Table 2, and Table 3 gives the complete list of experimental results of BDD package time statistics for hash key strategies proposed in the previous section tested on BDD construction, BDD addition and BDD printing algorithms, respectively.

All times in all tables are given in seconds. It should be noticed that hash key strategies in previous section and in headers of all tables have same denotation with symbols (1), (2), (3), and (4). Last four columnsfor each table presents comparable time performance of BDD package hash key strategies.

There are 15 of 19 benchmarks in table 1 for which algorithm has minimum computation time for strategy (4). According to that fact, the BDD construction algorithm is in most cases efficient for strategy (4).

There are 14 of 19 benchmarks in table 2for which algorithm hasminimum computationtime for strategy (1). According to that fact, the BDD addition algorithm is in most cases efficient for strategy (1).

There are only 17 of 19 benchmarks in table 3for which algorithm has minimum computationtime for strategy (2). According to that fact, the BDD printing algorithm is in most cases efficient for strategy (2).

For comparison, the hash key strategy (3) was least efficient for all BDD basic algorithms.

This BDD basic algorithms dependency of proposed hash key strategies is a major advantage of this paper.

TABLE I
BDD PACKAGE TIME STATISTIC FOR VARIOUS HASH KEY STRATEGIES TESTED ON BDD CONSTRUCTION ALGORITHM

Fun. name	inp/out/cubes	algorithm performance [s]			
		(1)	(2)	(3)	(4)
alu4	14 / 8 / 1028	0.20	0.15	0.18	0.15
apex1	45 / 45 / 206	5.78	5.18	5.04	4.80
apex2	39 / 3 / 1035	3.38	3.31	3.32	3.40
apex4	9/19/438	0.08	0.05	0.04	0.04
apex5	117/88/1227	0.34	0.30	0.29	0.27
b12	15/9/431	0.01	0.01	0.01	0.01
clip	9/5/167	0.01	0.01	0.01	0.01
con1	7/2/9	0.02	0.01	0.01	0.01
cordic	23/2/1206	0.08	0.06	0.05	0.03
cps	24/109/654	0.16	0.15	0.15	0.14
duke2	22/29/87	0.03	0.03	0.02	0.03
e64	65/65/65	0.02	0.01	0.01	0.01
ex4p	128/28/620	0.01	0.01	0.01	0.01
ex1010	10/10/1024	0.02	0.02	0.02	0.02
misex2	25/18/29	0.05	0.05	0.05	0.05
misex3	14/14/1848	0.24	0.20	0.20	0.21
misex3c	14/14/305	0.05	0.03	0.02	0.04
table3	14/14/135	0.03	0.02	0.01	0.01
table5	17/15/158	0.02	0.01	0.01	0.01

TABLE II
BDD PACKAGE TIME STATISTIC FOR VARIOUS HASH KEY STRATEGIES TESTED ON BDD ADDITION ALGORITHM

Fun. name	inp/out/cubes	algorithm performance [s]			
		(1)	(2)	(3)	(4)
alu4	14 / 8 / 1028	0.40	0.44	0.42	0.45
apex1	45 / 45 / 206	9.33	9.25	9.53	9.22
apex2	39 / 3 / 1035	5.57	5.75	5.74	5.63
apex4	9/19/438	0.31	0.26	0.29	0.31
apex5	117/88/1227	0.31	0.43	0.35	0.33
b12	15/9/431	0.02	0.03	0.03	0.04
clip	9/5/167	0.03	0.02	0.03	0.02
con1	7/2/9	0.03	0.04	0.04	0.04
cordic	23/2/1206	0.21	0.26	0.25	0.23
cps	24/109/654	0.42	0.51	0.57	0.59
duke2	22/29/87	0.07	0.07	0.07	0.07
e64	65/65/65	0.06	0.08	0.09	0.12
ex4p	128/28/620	0.11	0.16	0.19	0.21
ex1010	10/10/1024	0.16	0.18	0.25	0.24
misex2	25/18/29	0.11	0.12	0.13	0.12
misex3	14/14/1848	0.66	0.68	0.63	0.61
misex3c	14/14/305	0.14	0.12	0.11	0.10
table3	14/14/135	0.08	0.11	0.11	0.12
table5	17/15/158	0.04	0.05	0.05	0.06

TABLE III
BDD PACKAGE TIME STATISTIC FOR VARIOUS HASH KEY STRATEGIES TESTED ON BDD PRINTING ALGORITHM

Fun. name	inp/out/cubes	Algorithm performance [s]			
		(1)	(2)	(3)	(4)
alu4	14 / 8 / 1028	0.11	0.10	0.11	0.10
apex1	45 / 45 / 206	2.25	2.21	2.23	2.22
apex2	39 / 3 / 1035	1.81	1.80	1.83	1.84
apex4	9/19/438	0.03	0.02	0.02	0.02
apex5	117/88/1227	0.16	0.15	0.16	0.16
b12	15/9/431	0.01	0.01	0.01	0.01
clip	9/5/167	0.01	0.01	0.01	0.01
con1	7/2/9	0.01	0.01	0.01	0.01
cordic	23/2/1206	0.04	0.03	0.04	0.04
cps	24/109/654	0.07	0.06	0.07	0.06
duke2	22/29/87	0.02	0.02	0.02	0.01
e64	65/65/65	0.01	0.01	0.02	0.02
ex4p	128/28/620	0.01	0.01	0.01	0.01
ex1010	10/10/1024	0.02	0.01	0.02	0.02
misex2	25/18/29	0.03	0.04	0.04	0.03
misex3	14/14/1848	0.16	0.13	0.14	0.13
misex3c	14/14/305	0.02	0.01	0.02	0.02
table3	14/14/135	0.02	0.01	0.02	0.02
table5	17/15/158	0.02	0.01	0.02	0.02

VI. CONCLUSION

This paper describes the use of various hash keys in implementation of basic BDD algorithmson the top of a BDD package.I experimentally performed a detailed analysis of hashing in BDD package using BDD algorithms computation and direct performance monitoring. The ultimate goal is to exceed the computation performance for various BDD algorithms of BDD packages.

From experimental results, it is evident that different hash key strategies can be used in different points of BDD package computation. Especially, hash key strategies in basic BDD ccomputationsare promising. Further work will be devoted to deeper exploiting these possibilities as well as exploiting different hash key strategies of BDD packages.

REFERENCES

[1] D. Long, “The Design of Cache-friendly BDD Library“,Proceedings of the 1998 IEEE/ACM international Conference on CAD, pp. 639 - 645, 1998.

[2] R. Bryant, "Graph-based Algorithms for Boolean Function Manipulation", IEEE Trans. Computers, vol C-35, pp. 667-691, 1986.

[3] K. Brace, R. Rudell, R. Bryant, "Efficient implementation of a BDD package", Proc. Design Automation Conf., pp. 40-45, 1990.

[4] R. Rudell, "Dynamic Variable Ordering for Binary Decision Diagrams", Proc. Conf. on CAD, pp. 42-47, 1993.

[5] J. Sangavi, R. Ranjan, R. Bryton, A. Sangiovanni-Vincentelli, "High Performance BDD Package Based on Exploiting Memory Hierarchy", Proc. of the Design Automation Conf., 1996.

[6] H. Ochi, K. Yasuoka, S. Yajima, "Breadth-First Manipulation of Very Large Binary-Decision Diagrams", Proc. Int. Conf. on CAD, pp. 48-55, 1993.

[7] P. Ashar, M. Cheong, "Efficient Breadth-First Manipulation of Binary Decision Diagrams", Proc. Int. Conf. on CAD, pp. 622-627, 1994.

[8] G. Janssen, “Design of Pointerless BDD Package,“10th Int. Workshop on Logic & Synthesis Granlibakken, Lake Tahoe, CA, 2001.

[9] F. Somenzi, "CUDD: CU decision diagram package", Public Software, University of Colorado, Boulder, CO, 1997, http://vlsi.colorado.edu/~fabio/.

[10] T. Stornetta, F. Brewer, “Implementation of an Efficient Parallel BDD Package”, Proc. of 33rd Design Automation Conference, pp. 641–644, 1996.

[11] R. Sumners, "Correctness Proof of a BDD Manager in the Context of Satisfiability Checking", Proc. of ACL2 Workshop 2000, Technical Report TR-00-29, 2000.

[12] C. Krieger, "A Java 1 Implementation of a BDD Package", University of Utah, 1998, http://www.cs.colostate.edu/~krieger/.

[13] G. Janssen, “A Consumer Report on BDD Packages,“Proc. of the 16th Symposium on Integrated Circuits and Systems Design, pp.217-223, 2003.

[14] F. Somenzi, “Efficient Manipulation of Decision Diagrams,” Int. J. Software Tools for Technology Transfer (STTT), vol. 3, no.2, pp. 171-181, 2001.

[15] B. Yang, R. Bryant, D. O’Hallaron, A. Biere, O. Coudert, G. Janssen, R. Ranjan, F. Somenzi, “A Study of BDD Performance in Model Checking,” Int. Proc. FMCAD, Palo Alto, CA, pp. 255-289, 1998.

[16] S. Manne, D. Grunwald, F. Somenzi, "Remembrance of Things Past: Locality and memory in BDDs", Int. Proc. of the 34th ACM/IEEE Design Automation Conference, 1997, pp. 196-201.

[17] M. Lam, Context-Sensitive Pointers Analysis Using Binary Decision Diagrams, John Whaley, 2007.

[18] M. Sentovich, "A Brief Study of BDD Package Performance". Int. Proceedings of the Formal Methods on CAD, 1996, pp. 389-403.

[19] S. Minato, N. Ishiura, S. Jajima, "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation", Int. Proc. of the 27th ACM/IEEE Design Automation Conference, 1990, pp. 52-57.

[20] B. Yang, Y., Chen, R. Bryant, D. O'Hallaron, "Space and time efficient BDD construction via working set control". Int. 1998 Proceedings of Asia and South Pacific Design Automation Conference , 1998, pp. 423-432.

[21] N. Klarlund, T. Rauhe, "BDD algorithms and cache misses", BRICS Report Series RS-96-5, Department of Computer Science, University of Aarhus, 1996.

[22] C. Lee, “Representation of Switching Circuits by Binary Decision Programs,” Bell System Technical Journal, vol. 38, pp. 985-999, 1959.

[23] S. Ackers, “Binary Decision Diagrams”, IEEE Trans. on Computers, vol. C-27(6), pp. 509-516, 1978.

[24] T. Sasao, M. Fujita, “Representations of Discrete Functions”, Kluwer Academic Publishers, Boston, 1996.

[25] A. Aho, J. Hopcroft, J. Ullman,“Data Structures and Algorithms“,Addison-Wesley, Mass., USA, 1983.

[26] R. Rudell, Espresso Misc. Reference Manual Pages, 1993, http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm