A Comparison of Architectures for Various Decision Diagram Machines

Hiroki Nakahara*, Tsutomu Sasao*, and Munehiro Matsuura*
*Kyushu Institute of Technology, Iizuka, Japan

***Abstract*—This paper compares 6 decision diagram machines (DDMs) with respect to area-time complexity, throughput, and compatibility to the existing memory. First, 6 types of decision diagrams (DDs): BDD, MDD, QRBDD, QRMDD, heterogeneous MDD (HMDD), and QRHMDD are introduced. Second, corresponding DDMs are developed. Third, memory sizes and average path length (APL) for these DDs are compared. As for area-time complexity, the QDDM is the best; as for throughput, the QRQDDM is the best; and as for compatibility to the existing memory, the HMDDM is the best.**

I. Introduction

Various decision diagrams (DDs), e.g., BDD[2], MDD[6], QRBDD[12], QRMDD[4], heterogeneous MDD (HMDD)[8], have been proposed. DD machines (DDMs) are special purpose processors that evaluate DDs [1]. Various DDMs have been proposed [5], [1], [15], [7], [13]. Applications for DDMs include industrial process controllers [17], and logic simulators [5]. In [11], a parallelized DDM has been proposed. Compared with the Intel's Core2Duo microprocessor, it requires a quarter of the memory for the Core2Duo, while is about 100 times faster at its peak performance. As for the area-time complexity, [10] compares BDD, MDD, and HMDD, and concludes that, HMDD is the best for logic evaluation.

In this paper, we compare BDD, MDD, HMDD, QRBDD, QRMDD, and quasi-reduced ordered heterogeneous MDD (QRHMDD) with respect to the area-time complexity. Also, we present corresponding DDMs for these six types of DDs. Finally, we select the best types of DDs for specific applications and for an economical implementation.

The rest of the paper is organized as follows: Chapter 2 defines important words; Chapter 3 introduces architectures for 6 types of DDMs; Chapter 4 compares these types of DDMs; and Chapter 5 concludes the paper.

II. Preliminary

Definition 2.1: Let $f(X) : B^n \rightarrow B$ be a two-valued logic function, where $B = \{0, 1\}$. Let $X = (x_1, x_2, \ldots, x_n), x_i \in B$ be an ordered set of binary variables. Let $\{X\}$ denote the unordered set of variables in X. If $\{X\} = \{X_1\} \cup \{X_2\} \cup \cdots \cup \{X_u\}$ and $\{X_i\} \cap \{X_j\} = \phi (i \neq j)$, then $(X_1, X_2, \ldots, X_u)$ is a **partition of** X, where X_i is **a super variable**. When $k_i = |X_i| (i = 1, 2, \ldots, u)$, $k_1 + k_2 + \cdots + k_u = n$.

Definition 2.2: A **BDD** is obtained by applying **Shannon expansions** repeatedly to a logic function f. Each non-terminal node labeled with a variable x_i has two outgoing edges which indicate nodes representing cofactors of f with respect to x_i. When the Shannon expansions are performed with respect to k variables, all the non-terminal nodes have 2^k edges. In this case, we have a **Multi-valued Decision Diagram (MDD(k))**.

Definition 2.3: In a DD, a sequence of edges and non-terminal nodes leading from the root node to a terminal node is a **path**. **An ordered BDD (OBDD)** has the same variable order on any path. **A reduced ordered BDD (ROBDD)** is derived by applying the following two reduction rules to an OBDD:

1. Share equivalent sub-graphs.
2. If all the outgoing edges of a non-terminal node v indicate the same succeeding node u, then delete v and connect the incoming edges of v to u.

An **ROMDD(k)** can be similarly defined to the ROBDD. Note that, MDD(1) means BDD. In this paper, BDD and MDD(k) means ROBDD and ROMDD(k), respectively, unless stated otherwise.

For many benchmark functions, MDD(2)s are better than BDDs with respect to the area-time complexity [10]. Since each node of a MDD(2) has four edges, it is called a **Quaternary Decision Diagram (QDD)**. In this paper, we consider only QDDs among MDD(k)s, since MDD(2) has the best performance.

Definition 2.4: **A Quasi-Reduced ordered BDD (QRBDD)** is derived by applying only the reduction rule 1 in Definition 2.3.

In other words, the QRBDD has all variables on any path. **A Quasi-Reduced ordered QDD (QRQDD)** can be defined similarly.

Definition 2.5: In a QRBDD, a node with the outgoing edges indicating the same node is **redundant**.

Definition 2.6: Let $X = (X_1, X_2, \ldots, X_u)$ be a partition of the input variables, and $k_i = |X_i|$ be the number of inputs for node i. When $k = |X_1| = |X_2| = \cdots = |X_u|$, an ROMDD is **a homogeneous MDD (MDD(k))**. On the other hand, if there exists a pair (i, j) such that $|X_i| \neq |X_j|$, then, it is **a heterogeneous MDD (HMDD)**.

Definition 2.7: **A Quasi-Reduced ordered heterogeneous MDD (QRHMDD)** is derived by applying only the reduction rule 1 in Definition 2.3 to a heterogeneous MDD.

Example 2.1: Fig. 1 illustrates 6 types of DDs for MCNC benchmark function C17 [16]. The gray nodes are redundant. (End of Example)

Suppose that the evaluation time for all the DD nodes are the same, then the evaluation time for a DD is proportional to the **average path length (APL)** [3]. We assume that a DD machine evaluates each node in a fixed time. In this case, we can use APL to estimate the computation time.

Definition 2.8: Let $(X_1, X_2, \ldots, X_u)$ be a partition of the input variables X. Suppose that X_i can take any value c, where $c \in \{0, 1, \ldots, r-1\}$. Then, $P(X_i = c)$ denotes the probability that X_i has value c. **The Path Probability (PP)** of a path

0195-623X/10 $26.00 © 2010 IEEE
DOI 10.1109/ISMVL.2010.50

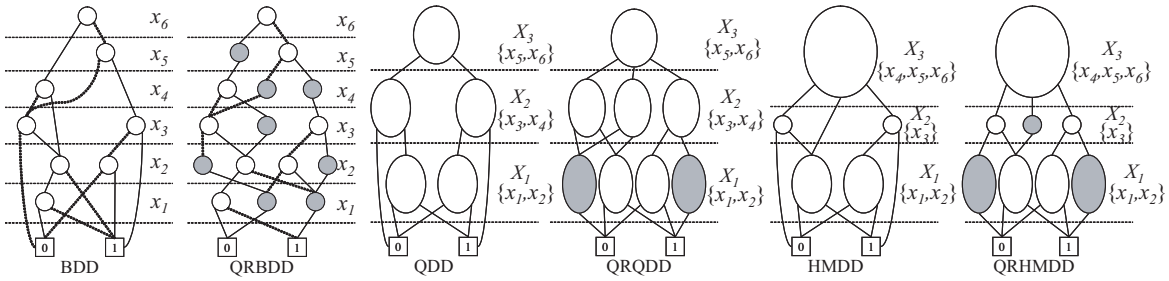

Fig. 1. Various Decision Diagrams (DDs).

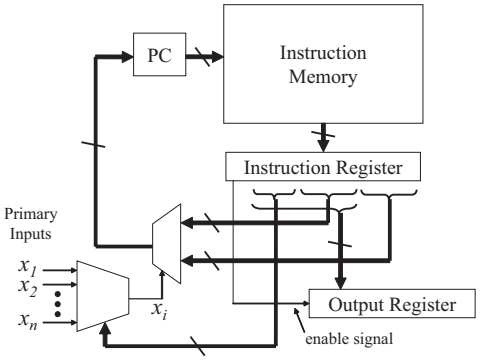

Fig. 2. BDDM.

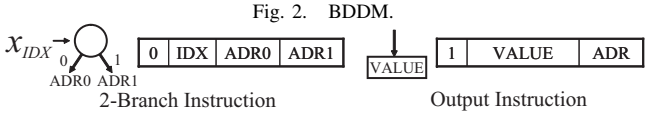

Fig. 3. Instruction set for the BDDM.

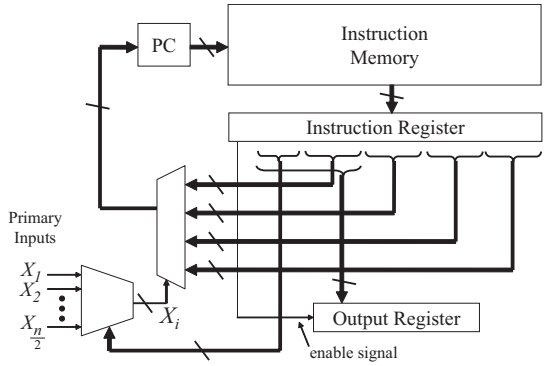

Fig. 4. QDDM.

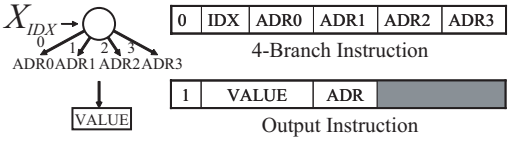

Fig. 5. Instruction for the QDDM.

p_i, denoted by $PP(p_i)$, is the probability that the path p_i is selected in all assignments of values to the r-valued variables. Then, we have $PP(p_i) = \sum_{\vec{c} \in C_i} P(X_1 = c_1) \cdot P(X_2 = c_2) \cdot \ldots \cdot P(X_u = c_u)$, where C_i denotes a set of assignments of values to the variables X selecting the path p_i, and $\vec{c} = (c_1, c_2, \ldots, c_u)$. **The average path length (APL)** of a DD is $APL = \sum_{i=1}^{N} PP(p_i) \cdot l_i$, where N denotes the number of paths, and l_i denotes the path length of path p_i.

III. Architectures for Various DD Machines

A. BDD Machine (BDDM)

Fig. 2 shows a BDD Machine (BDDM), where **the instruction memory** stores instructions that evaluate nodes for a BDD; **the instruction register** stores an instruction from the instruction memory; **the output register** stores primary outputs; **the program counter (PC)** retains an address of a node currently evaluated. Fig. 3 shows the instruction set for the BDDM. The 2-branch instruction evaluates a non-terminal node, while the output instruction evaluates a terminal node.

Let M_{BDDM} be the size of the instruction memory, n be the number of inputs, and N_{BDD} be the number of nodes. Then, we have the relation:

$$M_{BDDM} = N_{BDD}(1 + \lceil log_2 n \rceil + 2\lceil log_2 N_{BDD} \rceil). \quad (1)$$

B. QDD Machine (QDDM)

Fig. 4 shows a QDD Machine (QDDM). The differences from the BDDM are the number of branches (four in the QDDM) and the number of inputs for each node (two in the QDDM). Fig. 5 shows the instruction set for the QDDM. The QDDM uses the 4-branch instruction that evaluates a non-terminal node, and the output instruction.

Let M_{QDDM} be the size of the instruction memory, $(X_1, X_2, \ldots, X_u)$ be a partition of the inputs X, and N_{QDD} be the number of the nodes. Then, we have the relation:

$$M_{QDDM} = N_{QDD}(1 + \lceil log_2 u \rceil + 4\lceil log_2 N_{QDD} \rceil), \quad (2)$$

where the first term corresponds to the opecode; the second term corresponds to the index; and the last term corresponds to the four pointers.

C. QRBDD Machine (QRBDDM)

Fig. 6 shows a QRBDD Machine (QRBDDM), where the instruction memory, the instruction register, the output register, and the PC are the same as those of the BDDM shown in Fig. 2. In the QRBDD, all the variables appear on any path. The QRBDDM uses a counter and a shift register. The counter keeps the current index, while the shift register keeps the input variables. **The controller** generates signals to change between **the branch mode** and **the output mode**. Fig. 7 shows the instruction set for the QRBDDM. Since the indices of nodes to be evaluated are known in advance, the opcode and the index fields can be omitted. Thus, the QRBDDM has shorter instruction words than the BDDM.

The input variable and the jump addresses can be read concurrently. Thus, the QRBDDM can perform the branch

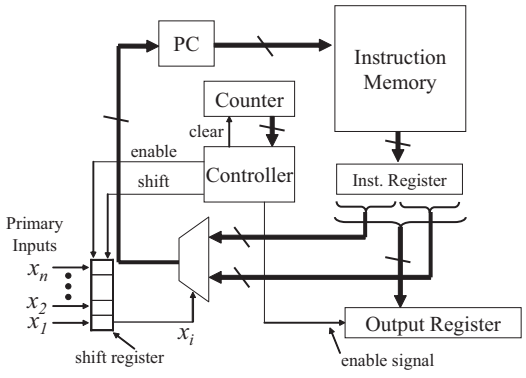

Fig. 6. QRBDDM.

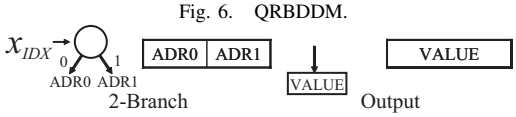

Fig. 7. Instruction set for the QRBDDM.

operation in one clock cycle. In contrast, since in a BDDM, a branches to a node with an arbitrary index is permitted, the BDDM must read the jump addresses after reading the index. The QRBDDM can be pipelined, while the BDDM cannot be.

Let M_{QRBDDM} be the size of the instruction memory, and N_{QRBDD} be the number of nodes. Then, we have the relation:

$$M_{QRBDDM} = 2 \cdot N_{QRBDD} \cdot \lceil log_2 N_{QRBDD} \rceil. \quad (3)$$

D. QRQDD Machine (QRQDDM)

In a QRBDDM, by extending the number of branches to four, and by using two shift registers, we have the QRQDD Machine (QRQDDM) shown in Fig. 8. The QRQDDM has the branch mode and the output mode, similarly to the QRBDDM. Fig. 9 shows the instruction set for the QRQDDM. The QRQDDM has four jump addresses in the branch instruction. In the QRQDDM, the opcode and the index field can be omitted, since the indices are known in advance.

Let M_{QRQDDM} be the size of the instruction memory, and N_{QRQDD} be the number of nodes. Then, we have the relation:

$$M_{QRQDDM} = 4 \cdot N_{QRQDD} \cdot \lceil log_2 N_{QRQDD} \rceil. \quad (4)$$

E. Direct Branch and Indirect Branch

Four machines (BDDM, QDDM, QRBDDM, QRQDDM) are homogeneous, that is, the numbers of branches are the same for each node. Thus, the word lengths of the branch

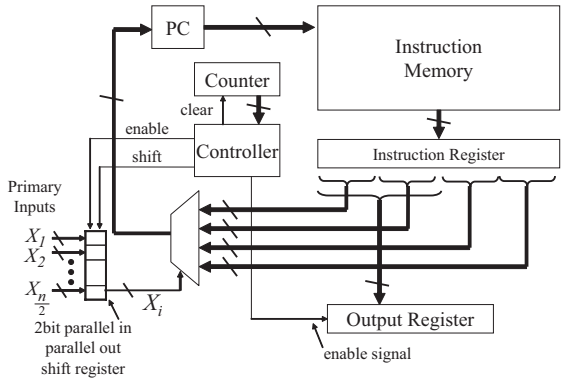

Fig. 8. QRQDDM.

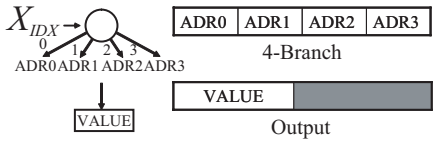

Fig. 9. Instruction set for the QRQDDM.

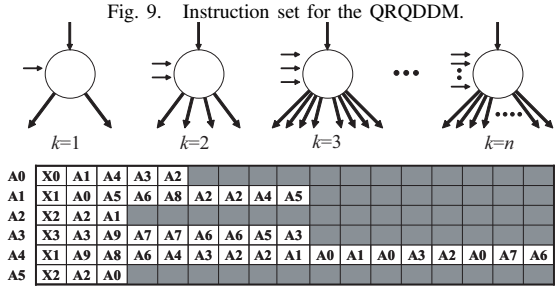

Fig. 10. Direct branch instructions for the HMDD.

instruction are also the same. These machines can directly get the jump address by reading input variables and the branch instruction. We call this **direct branch**. On the other hand, since the HMDD and the QRHMDD accept the arbitrary number of input variables for each node, the numbers of branch addresses can be different. Thus, the word lengths for the branch instruction can be different as follows:

Example 3.2: Fig. 10 shows the direct branch instructions for the HMDDM that evaluates non-terminal nodes. When the number of inputs for a node is k, the number of branches is 2^k. Thus, the word length for the branch instruction of nodes can be different. (End of Example)

To use the memory efficiently for the HMDDM and the QRHMDDM, we use **indirect branch** that reads the index and the jump address separately. First, the machine reads the current index. Then, it reads the jump address corresponding to the value of the current input variables. Although the indirect branch is slower than the direct branch, it uses the memory efficiently, since the words have the same length. Next example explains it.

Example 3.3: Fig. 11 shows the indirect branch instructions for the HMDDM that evaluates a non-terminal node. In Fig. 11, the *index* stores the index for the input variable. (End of Example)

The indirect branch is performed as follows:

Algorithm 3.1:
1. Read the *index*, then compute the indirect address for the jump address.
2. Read the jump address using the address obtained in Step 1.
3. Perform the jump operation.

Even the sizes of super variables are different, the word lengths for the indirect branch are the same. So, the indirect jump can use the memory efficiently for heterogeneous DDs. In this paper, for the HMDD and the QRHMDD, we use the indirect branch.

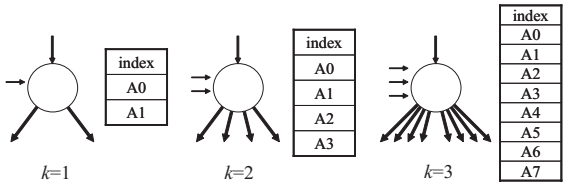

Fig. 11. Indirect branch instructions for the HMDDM.

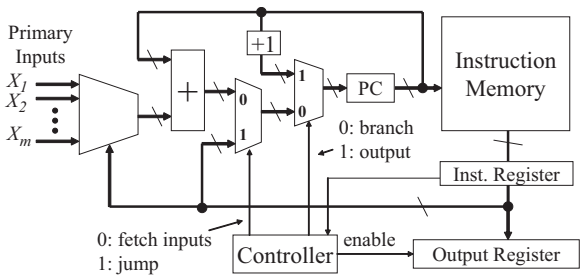

Fig. 12. HMDDM.

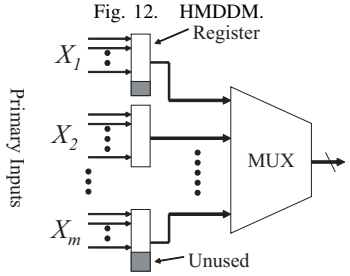

Fig. 13. Input selector for the HMDDM.

F. HMDD Machine (HMDDM)

Fig. 12 shows an HMDD Machine (HMDDM). The HMDDM consists of the instruction memory, the instruction register, the output register, and the PC. It uses indirect 2^k-branch instructions and output instructions. To execute the indirect 2^k-branch instruction, the HMDDM uses **the fetch mode** and **the jump mode**. In the fetch mode, input variables are selected. In the jump mode, the jump addresses are read and the branch operations are performed. To execute the output instruction, the HMDDM uses **the output mode**. To change the modes, the controller generates control signals, and two multiplexors select the mode. The HMDDM uses an adder to compute the address in the jump mode, and an increment circuit in the output mode.

Since the size of the super variables can be different, the HMDDM uses input registers with $\max_i\{k_i\}$ bits. Fig. 13 shows the input selector for the HMDDM.

Algorithm 3.2: (Indirect 2^k-branch instruction)

Step 1. Fetch mode.
1) 1.1 Read the instruction memory specified by the PC.
1.2 To add the input variable and the content of the PC, the controller generates signals. Then, the indirect address is sent to the PC.

Step 2. Jump mode.
2) 2.1 Read the jump address specified by the PC.
2.2 To perform the jump, the controller generates signals. Then, the jump address is sent to the PC.

Algorithm 3.3: (Output instruction)

Step 1. Output mode.
1) 1.1 Read the instruction memory specified by the PC.
1.2 The controller generates signals, and the output data is sent to the output register. Concurrently, it increments the PC.

Step 2. Perform the jump mode shown in Algorithm 3.2 Step 2.

Let M_{HMDDM} be the memory size for the instruction memory, k_i be the number of inputs for a node i, and N_{HMDD} be the number of nodes. Since in the HMDD, each node has an index and 2^{k_i} branch edges, the necessary number of addresses for each node is $2^{k_i}+1$. So, the total number of addresses for the HMDDM is $a = \sum_{i=1}^{N_{HMDD}}(2^{k_i}+1)$. Thus, we have the relation:

$$M_{HMDDM} \quad = \quad a \cdot \lceil log_2 a \rceil. \tag{5}$$

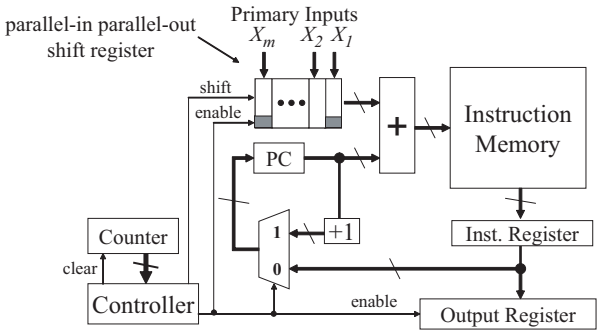

Fig. 14. QRHMDDM.

G. QRHMDD Machine (QRHMDDM)

Fig. 14 shows the QRHMDD Machine (QRHMDDM), where the instruction memory, the output register, and the PC are same as those for the HMDDM shown in Fig. 12. In the QRHMDD, all the super variables appear on any path. The QRHMDDM uses a counter and a shift register. The counter keeps the current index, while the shift register keeps the input variable. The controller generates the control signals. The QRHMDDM uses the branch mode and the output mode. Since the QRHMDDM uses fewer modes than the HMDDM, its controller for the QRHMDDM is simpler. Fig. 15 shows the indirect 2^k-branch instructions for the QRHMDDM. Since the QRHMDDM knows the index of the super variables in advance, the opcode and the index fields can be omitted.

Algorithm 3.4: (Branch mode for the QRHMDDM)

1. To obtain the indirect address, first, add the content of the PC and the value of the input variables. Next, read the jump address from the instruction memory, and send it to the instruction register.
2. Store the jump address to the PC. Concurrently, increment the counter for the index, and perform the shift operation.

Algorithm 3.5: (Output mode for the QRHMDDM)

1. To perform the indirect addressing, first, get the content of the PC. Then, read the output value from the instruction memory.
2. Store it to the output register. Concurrently, increment the PC.
3. Clear the counter for the index to zero, and store the input variables to the shift register.

Let $M_{QRHMDDM}$ be the memory size, k_i be the number of inputs for a node i, and N_{QRHMDD} be the number of nodes. Since the number of branches for each node is 2^{k_i}, the total number of address for the QRHMDDM is $b = \sum_{i=1}^{N_{QRHMDD}}(2^{k_i})$. Thus, we have the relation:

$$M_{QRHMDDM} \quad = \quad b \cdot \lceil log_2 b \rceil \tag{6}$$

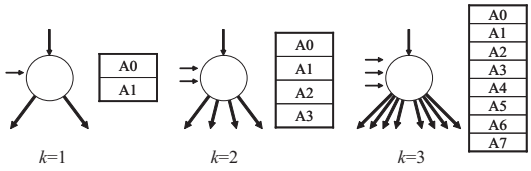

Fig. 15. Indirect 2^k-branch instructions for the QRHMDDM.

TABLE I
COMPARISON OF THE MEMORY SIZES [BYTES].

Name	BDD	QDD	QRBDD	QRQDD	HMDD	QRHMDD
C880	77760	82892	93724	86835	80557	74898
C1908	114136	106320	105323	101094	117515	84150
C432	4611	4400	7542	6380	4736	6543
apex2	2015	2274	3231	3130	2048	3234
too_large	2015	2274	3231	3130	2048	3525
apex1	9254	12208	31973	31499	9031	27846
apex3	6177	7248	30947	29926	6126	26334
apex7	1893	2241	3647	3895	2034	4104
chkn	987	931	1735	1593	1203	2079
duke2	2337	2473	4029	3840	2596	4344
frg1	223	302	736	696	260	978
misex3	2959	3139	3594	3335	3469	3891
pcle	313	340	610	560	337	804
ratio	1.00	1.11	2.10	2.01	1.07	2.17

IV. COMPARISON OF VARIOUS DD MACHINES

A. Construction of DDs

We constructed various DDs from selected MCNC benchmark functions [16]. Then, we obtained the memory size and APL. As for a multi-output function, we partition the function into single output functions. We used the variable order that minimizes the memory size of the BDD. For the HMDDM and the QRHMDDM, different partitions of the input X and variable orders produce different memory sizes and APL. In this experiment, we built the DD that minimizes APL with the memory size limitation [9]. To construct the HMDDMs and the QRHMDDMs, the memory size limitations are set to those of the BDDs and the QRBDDs, respectively.

B. Comparison of the Memory Size

Table I compares the memory sizes, where memory sizes for the DDs were obtained from Exprs. (1),(2),(3),(4),(5), and (6). From Table I, we have the following observations: The memory size of Quasi-Reduced DDs is 2.1 times of other DDs. The memory sizes of the BDDs are nearly equal to that of the QDDs. The number of edges for each node of the QDD is twice of the BDD. However, the number of the nodes for the QDD is about half of that for the BDD. Thus, BDDs and QDDs have nearly the same size of memory.

C. Comparison of APL

Table II compares APL. From Table II, we have the following observations: APL of Quasi-Reduced DDs are 3.7 times (BDD to QRBDD), 2.8 times (QDD to QRQDD), and 3.4 times (HMDD to QRHMDD), respectively. APL of QDDs is 33% smaller than BDDs. APL of HMDDs is 27% smaller than QDDs. APL of QRQDDs is 48% smaller than QRBDDs. APL of QRHMDDs is 14% smaller than QRQDDs.

D. Discussions

Table III compares various DDMs. We can find the best DDMs with respect to the area-time complexity, throughput, and memory compatibility.

TABLE II
COMPARISON OF APL.

Name	BDD	QDD	QRBDD	QRQDD	HMDD	QRHMDD
C880	145.5	98.2	419	218	64.9	155
C1908	260.8	151.9	753	390	85.6	289
C432	86.6	59.6	225	113	48.4	152
apex2	21.4	15.4	107	56	14.3	67
too_large	21.4	15.4	107	56	14.3	45
apex1	173.6	116.9	783	412	67.9	341
apex3	187.1	114.3	605	311	67.4	261
apex7	135.3	97.5	374	216	80.7	171
chkn	19.9	13.1	136	70	9.6	60
duke2	91.1	58.5	324	169	40.2	138
frg1	9.2	6.8	34	18	5.2	15
misex3	87.0	51.3	195	99	25.8	71
pcle	27.0	19.4	79	42	15.8	30
ratio	1.00	0.67	3.70	1.94	0.49	1.68

TABLE III
COMPARISON OF VARIOUS DDMS.

DDM	Branch Method	Architecture	Mem Size	APL	Pipeline	Clk Cycle
BDDM	Direct	Boute[1]	1.00	1.00	impossible	1
QRBDDM	Direct	Iguchi[4]	2.10	3.70	possible	1
QDDM	Direct	Thayse[15]	1.11	0.67	impossible	1
QRQDDM	Direct	Iguchi[4]	2.01	1.94	possible	1
HMDDM	Indirect	This work	1.07	0.49	impossible	2
QRHMDDM	Indirect	This work	2.17	1.68	possible	1

a) Area-time complexity: Area-time complexity is important for the embedded system, such as a sequencer, a controller, and so on. The power consumption can be divided into the static power and the dynamic power. The area for the DDM is related to the static power, while the APL (time) is related to the dynamic power. A processor with low area-time complexity dissipates low power. In a DDM, APL and architecture affect the performance (time). Since the instruction memory occupies the most area for the DDM, we assume that the area is proportional to the memory size. We consider the area-time complexity for each DDM.

The memory sizes for QRDDs (QRBDD, QRQDD, QRHMDD) are twice of other DDs (BDD, QDD, HMDD). APL for QRDDs is 1.68-3.70 times of other DDs. Since a QRDDM (QRBDDM, QRQDDM, QRHMDDM) uses the shift register instead of the input selector, the amount of hardware for the QRDDM is lower than that for other DDMs (BDDM, QDDM, HMDDM). Evaluation time for a node of the QRDDM is shorter than that of other DDMs. However, to match the area-time complexity for the QRDDM to that for the DDM, the evaluation for a node of the QRDDM must be 3.64-7.77 times faster than that of the DDM. Unfortunately, it is difficult in the current technology. Therefore, as for area-time complexity, the DDM outperforms the QRDDM.

APL for the BDD is 1.5 times of the QDD, and the memory size for the BDD is nearly equal to that for the QDD. Thus, as for the area-time complexity, the QDDM outperforms the BDDM.

APL for the QDD is 1.36 of the HMDD, and the memory size for the QDD is nearly equal to that for the HMDD. The HMDDM evaluates a node by the indirect branch that accesses the instruction memory twice. Thus, the HMDDM requires two clocks to evaluate a node[1]. On the other hand, the QDDM requires only one clock by the direct branch. In the HMDDM, an adder is used to compute an indirect address.

[1]By storing the jump address and its index, we can perform fetch and jump modes at a time. Thus, we can evaluate a node in each clock. However, it increases the word length and complicates the architecture.

So, the architecture for the HMDDM is more complex than that for the QDDM. Therefore, the QDDM evaluates a node faster than the HMDDM. Thus, as for area-time complexity, the QDDM outperforms the HMDDM.

From the above discussions, as for area-time complexity, the QDDM is the best.

b) Throughput: QRDDMs (QRBDDM, QRQDDM, and QRHMDDM) can use the pipeline architecture [5]. Let q be the number of units in the pipelined QRDDM. Although the hardware for the pipelined QRDDM is q times of the non-pipelined QRDDM, throughput for the pipelined QRDDM is at most q times of the non-pipelined ones. We consider a high-throughput machine for each pipelined QRDDM.

APL for the QRBDD is 1.90 times of the QRQDD. If the QRQDDM and the QRBDDM have the same number of pipeline stages, then the QRQDDM has higher throughput than the QRBDDM. By increasing the pipeline stage for the QRBDDM, throughput for the pipelined QRBDDM approaches to that for the pipelined QRQDDM. However, the hardware becomes large, and the controller for the pipeline operation also becomes complex. Thus, as for throughput, the pipelined QRQDDM outperforms the pipelined QRBDDM.

The sizes of the super variables for the QRHMDD can be different. Thus, the unification of the delay time for units to evaluate the super variables is difficult. On the other hand, in the QRQDD, since the size of the super variable is two, unification of the delay time for units is simple. Therefore, as for throughput, the pipelined QRQDDM outperforms the pipelined QRHMDDM.

From the above discussions, as for throughput, the QRQDDM is the best.

c) Compatibility to the Existing Memory: For the direct branch machines (BDDM, QDDM, QRBDDM, QRQDDM), the word length for the instruction depends on the number of nodes for the function. Thus, the word length for the direct branch machine may not match to the existing memory whose word length is multiple of 8. For example, in our experiment, the word lengths for the QDDM obtained by Expr. (2) are 34-63. Although we can use irregular sized memories, it is unrealistic. In FPGAs, we can configure the memory whose words have different lengths by combining embedded memories. However, the memory size for the FPGA is at most a few mega bits, so it cannot store the function requiring more nodes. On the other hand, in the indirect branch machines (HMDDM, QRHMDDM), the word lengths for the instruction are relatively short. Also, these machines can use all the given memory by selecting the optimal size of the super node. Therefore, as for the compatibility to the existing memory, the indirect branch machines are better.

APL for the QRHMDD is 3.42 times of the HMDD. The QRHMDDM uses one clock to evaluate a node, while the HMDDM uses two clocks. Thus, as for the evaluation of a path, the HMDDM is about 1.71 times faster than the QRHMDDM. On the other hand, the memory size for the QRHMDDM is about twice of the HMDDM. Thus, as for area-time complexity, the HMDDM outperforms the QRHMDDM.

From above discussion, for compatibility to the existing memory, the HMDDM is the best.

V. Conclusion and Comments

This paper compared DDMs with respect to area-time complexity, throughput, and compatibility to the existing memory. First, it considered 6 types of decision diagrams (DDs): BDD, MDD, QRBDD, QRMDD, HMDD, and QRHMDD. Second, it presented corresponding DDMs. Third, it compared the memory size and APL for these DDs by using benchmark functions. The QDDM is the best for area-time complexity; the QRQDDM is the best for throughput; and the HMDDM is the best for compatibility to the existing memory.

For the HMDD, the memory size limitation is set to that of the BDD. However, in many cases, the actual memory size is power-of-two. We can minimize the APL with an enough memory by increasing the size of the super variables.

VI. Acknowledgments

This research is supported in part by the Grants in Aid for Scientific Research of JSPS, and the grant of Innovative Cluster Project of MEXT (the second stage). Discussion with Dr. Shinobu Nagayama was quite useful.

References

[1] R. T. Boute, "The binary-decision machine as programmable controller," *Euromicro Newsletter,* Vol. 1, No. 2, pp. 16-22, 1976.

[2] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677-691, Aug. 1986.

[3] J. T. Butler, T. Sasao, and M. Matsuura, "Average path length of binary decision diagrams," *IEEE Trans. Comput.*, Vol. 54, No. 9, pp. 1041-1053, Sep. 2005.

[4] Y. Iguchi, T. Sasao, and M. Matsuura, "Implementation of multiple-output functions using PROMDDs," *30th Int'l Symp. on Multiple-Valued Logic (ISMVL2000)*, Portland, Oregon, U.S.A., Mya 23-25, 2000, pp.199-205.

[5] Y. Iguchi, T. Sasao, M. Matsuura, and A. Iseno, "A hardware simulation engine based on decision diagrams," *Asia and South Pacific Design Automation Conference (ASPDAC2000)*, Jan., 26-28, Yokohama, Japan, pp.73-76.

[6] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Multi-valued decision diagrams: Theory and applications," *Multiple-Valued Logic*, Vol.4, no.1-2, 1998, pp.9-62.

[7] D. Mange, "A high-level-language programmable controller: Part I-II," *IEEE Micro,* Vol. 6, No. 1, pp. 25-41 (Part I), Vol. 6, No. 2, pp. 47-63 (Part II), Feb/Mar, 1986.

[8] S. Nagayama and T. Sasao, "Compact representations of logic functions using heterogeneous MDDs," *33rd IEEE Int'l Symp. on Multiple-Valued Logic (ISMVL2003)*, May, 2003, pp.247-255.

[9] S. Nagayama and T. Sasao, "Code generation for embedded systems using heterogeneous MDDs," *the 12th workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI 2003)*, Hiroshima, Japan, April, 2003, pp.258-264.

[10] S. Nagayama and T. Sasao, "On the optimization of heterogeneous MDDs," *IEEE Transactions on CAD*, Vol.24, No.11, Nov., 2005, pp.1645-1659.

[11] H. Nakahara, T. Sasao, M. Matsuura, and Y. Kawamura, "Emulation of sequential circuits by a parallel branching program machine," *5th International Workshop on Applied Reconfigurable Computing (ARC2009)*, Karlsruhe, Germany, March 16-18, 2009. *Lecture Notes in Computer Science*, LNCS5443, March 2009, pp.261-267.

[12] T. Sasao and M. Fujita (ed.), *Representations of Discrete Functions*, Kluwer Academic Publishers, 1996.

[13] T. Sasao, H. Nakahara, M. Matsuura, Y. Kawamura, and J.T. Butler, "A quaternary decision diagram machine and the optimization of its code," *39th International Symposium on Multiple-Valued Logic (ISMVL 2009)*, May, 2009, pp.362-369.

[14] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Computing Curveys,* Vol. 37, Issue 3, Sep. 2005, pp.238-275.

[15] A. Thayse, M. Davio, and J. P. Deschamps, "Optimization of multi-valued decision algorithms," *Int'l Symp. (ISMVL79)*, Rosemont, IL., May, 1979, pp.171-177.

[16] S. Yang, "Logic synthesis and optimization benchmark user guide version 3.0," *MCNC*, Jan. 1991.

[17] P.J.A.Zsombor-Murray, L.J. Vroomen, R.D. Hudson, Le-Ngoc Tho, and P.H. Holck, "Binary-decision-based programmable controllers, Part I-III," *IEEE Micro* Vol. 3, No. 4, pp. 67-83 (Part I), No. 5, pp. 16-26 (Part II), No. 6, pp. 24-39 (Part III), 1983.