社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

決定グラフに基づく論理関数の評価のメモリパッキングを用いた高速化について

田中浩之† 中原啓貴† 松浦宗寛†† 笹尾勤††

† 九州工業大学大学院情報工学研究科情報創成工学専攻
†† 九州工業大学情報工学部電子情報工学科

あらまし 決定グラフに基づく論理関数の評価法について検討する．特に, Quasi-Reduced Multi-valued Decision Diagrams(QRMDDs) に基づく論理関数の評価において, メモリパッキングを用いてメモリを削減し, 高速化する方法を提案する．メモリ量を削減することによって, キャッシュミスが減少し, その結果, 高速になったものと推測できる．
キーワード QRMDD, メモリパッキング

A Method to Evaluate Logic Functions Based On Decison Diagram Using Memory Packing

Hiroyuki TANAKA†, Hiroki NAKAHARA†, Munehiro MATSUURA††, and Tsutomu SASAO††

† Program of Creation Informatics, Kyushu Institute of Technology
†† Department of Computer Sience and Electronics, Kyushu Institute of Technology

Abstract This paper proposes a method to evaluate logic functions using decision diagrams. Quasi-Reduced Multi-valued Decision Diagrams(QRMDDs) are used for fast logic simulation. This paper also shows a method to reduce memory requirement and computation time by memory packing. The speedup is due to the reduction of cache miss.
Key words QRMDD, memory packing

1. はじめに

LSI の集積度の向上に伴い, 開発期間, 開発費用の増大が問題になっている．特に, 論理シミュレーションは論理回路の機能を検証する際の基本作業であるが, 多大な時間を必要とすることから高速化が求められている．Binary Decision Diagrams(BDDs) や Multi-valued Decision Diagrams(MDDs) は, 論理シミュレーションなどで用いられるデータ構造である．現在のコンピュータの記憶装置は階層構造を有する．Quasi-Reduced Binary Decision Diagrams(QRBDDs) や Quasi-Reduced Multi-valued Decision Diagrams(QRMDDs) では, メモリを一定の順序で読み出すため, そのような階層構造に適している．そこで, 本論文では QRMDD に基づく論理シミュレータを PC 上に実現し, メモリパッキングという手法を用いて必要メモリ量を削減し, 論理関数を高速に評価する手法を提案する．

2. BDD(二分決定グラフ)

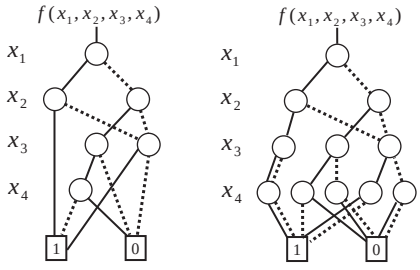
BDD は, 論理関数をグラフ的に表現したもので, 多くの実用的な関数に対しては, 他の表現法に比べ, コンパクトに表現できる．論理式で表現された論理関数を BDD に変換するには, 次に示すシャノン展開を繰り返し適用する．

［定理 2.1］ 任意の論理関数 $f(x_1, x_2, \cdots, x_n)$ は, 次のように展開できる．

$$f(x_1, x_2, \cdots, x_n) = \bar{x}_1 f(0, x_2, \cdots, x_n) \vee x_1 f(1, x_2, \cdots, x_n)$$

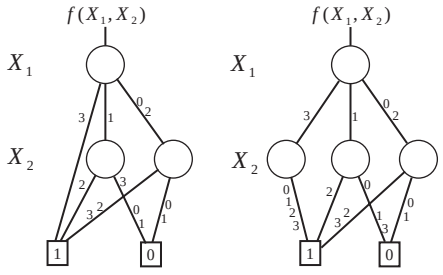
このような展開を, **シャノン展開**という．また, BDD において, シャノン展開を適用する変数の順序を**変数展開順序**と呼ぶ．

［定義 2.1］ BDD は, 節点と枝からなる有向グラフである．節点と節点集合をそれぞれ v, V と表記する．節点 v は**インデックス**, $\mathrm{index}(v) \in \{1, \cdots, n\}$ を持つ．ここで, $\mathrm{index}(v)$ は v に対応する変数のインデックスである．節点 v には 2 種類あり, $\mathrm{index}(v) = 0$ のとき, v は**終端節点**, $\mathrm{index}(v) > 0$ のとき, v は**非終端節点**である．終端節点は, 葉ともいう．終端節点には 0 と 1 の 2 種類があり, それぞれ論理関数の関数値の 0,1 に対応している．非終端節点 v は, 2 つの子節点, $\mathrm{low}(v)$, $\mathrm{high}(v) \in V$ を持ち, $\mathrm{low}(v)$ を指す枝を **0 枝**, $\mathrm{high}(v)$ を指す枝を **1 枝**とする．BDD において, 変数展開順序を固定し, 冗長な節点と同形な部分グラフの共有を可能な限り行ったものを**既約順序付き二**



(a)ROBDD (b)QRBDD

図 1 $f = x_1x_2 \vee \bar{x_2}x_3 \vee x_3\bar{x_4}$



(a)MDD(2) (b)QRMDD(2)

図 2 $f = x_1x_2 \vee \bar{x_2}x_3 \vee x_3\bar{x_4}$

分決定グラフ (Reduced Ordered Binary Decision Diagram : **ROBDD**) という.

任意の論理関数 f は ROBDD を用いて表現でき, また, 変数展開順序を固定すれば, ROBDD は一意に定まる.

[定義 2.2] 決定グラフの根から終端節点までの経路のことを**パス**といい, パス上に現れる枝の数を**パス長**という.

論理関数 $f = x_1x_2 \vee \bar{x_2}x_3 \vee x_3\bar{x_4}$ を表す ROBDD を図 1(a) に示す. 図 1 中の実線は 1 枝, 点線は 0 枝を表す. 通常, BDD といえば ROBDD のことを指し, 本論文でも同様に取り扱う.

3. QRBDD(Quasi-Reduced BDD)

QRBDD は, 根節点から終端節点に至るいずれのパス上においても論理関数の全ての変数が, 丁度一度ずつ現れるような BDD である. 図 1(a) の BDD から得られる QRBDD を図 1(b) に示す. QRBDD においては, 根節点から終端節点に至る全てのパスのパス長は等しい.

4. MDD(Multi-valued Decision Diagram)

[定義 4.1] 二値論理関数を $f(X)$, 入力変数を $X = (x_1, x_2, \ldots, x_n)$ とする. ここで, $x_i (i = 1, 2, \ldots, n)$ は 0 又は 1 をとる変数, $\{X\}$ は X の変数の集合, $\mid X \mid$ は X の要素数とする.

$$\{X\} = \{X_1\} \cup \{X_2\} \cup \cdots \cup \{X_u\}$$
$$\{X_i\} \cap \{X_j\} = \phi \quad (i \neq j)$$

であるとき, $(X_1, X_2, \ldots, X_u)$ を X **の分割**という.

また, $\mid X_i \mid = k$ $(i = 1, 2, \ldots, u)$ であるとき, 二値論理関数 $f(X)$ は写像

$$f(X_1, X_2, \ldots, X_u) : \{0, 1, 2, \ldots, 2^k - 1\}^u \rightarrow \{0, 1\}$$

を表現する.

[定義 4.2] $X = (x_1, x_2, \ldots, x_n)$ の分割を $(X_1, X_2, \ldots, X_u)$, $\mid X_i \mid = k$ $(i = 1, 2, \ldots, u)$ とするとき, $f(X_1, X_2, \ldots, X_u)$ を表す決定グラフを MDD(k) と表記する. このとき, 各非終端節点は 2^k 本の枝を持つ. また, MDD(1) は BDD と同じものである.

MDD において, 変数展開順序を固定し, 冗長な節点と同形な部分グラフの共有を可能な限り行ったものを Reduced Ordered Multi-valued Decision Diagram(ROMDD) という. 通常, MDD といえば ROMDD のことを指し, 本論文でも同様に取り扱う.

[例 4.1] 関数 $f(X) = x_1x_2 \vee \bar{x_2}x_3 \vee x_3\bar{x_4}$ の変数 $X = (x_1, x_2, x_3, x_4)$ の分割を, $X_1 = (x_1, x_2)$, $X_2 = (x_3, x_4)$ とすると, MDD(2) は図 2(a) のように表される. このとき, X_i の取り得る値は {00, 01, 10, 11} の 4 値であり, X_i がそれぞれ, 00 のときを 0 枝, 01 のときを 1 枝, 10 のときを 2 枝, 11 のときを 3 枝とする. (例終り)

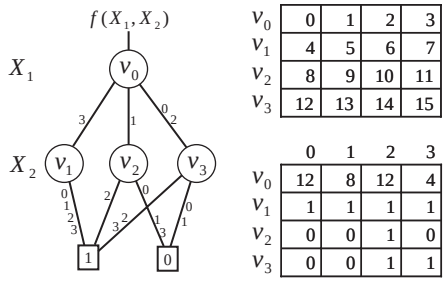
一般に, MDD(k)($k \geqq 2$) のパス長は BDD のパス長の $1/k$ となり, BDD に比べ, 高速に関数を評価できる. また, 図 1(b) に示した QRBDD の変数 $X = (x_1, x_2, x_3, x_4)$ の分割を, $X_1 = (x_1, x_2)$, $X_2 = (x_3, x_4)$ とすると, 図 2(b) に示す QRMDD(2) が得られる. QRMDD においては, 根節点から終端節点に至る全てのパスのパス長は等しい.

5. QRMDD に基づく論理シミュレーション

決定グラフの評価法には, 決定グラフのデータ構造を率直に構築する方法や, 決定グラフの各非終端節点を If then else 文で書き換えるブランチング・プログラム法などがある. しかし, QRMDD のデータ構造を率直に構築する方法は, 必要メモリ量が大きくなる. また, ブランチング・プログラム法は, QRMDD の各非終端節点数に比例して命令コードのサイズが大きくなる. そこで, 必要メモリ量を削減するために, QRMDD の評価にテーブルと呼ばれる整数配列を用いる.

[定義 5.1] **テーブル**は, QRMDD(k) の各非終端節点を表す部分配列からなる. 節点 v_n を表す部分配列が配列の第 s 番目から始まり, 節点 v_n が $\{0, \ldots, 2^k - 1\}$ の枝を持つとき, 節点 v_n を表す部分配列を Table[s]〜Table[$s + 2^k - 1$] とする. Table[$s + j$] には節点 v_n の j 枝が指す節点 v_m の部分配列の先頭の位置を格納する. 節点 v_m が終端節点であれば, 終端節点の値を格納する.

[例 5.1] 図 3(a) の QRMDD(2) の各節点をそれぞれ v_0, v_1, v_2, v_3 とすると, 部分配列の要素数は節点の枝数と等しい. v_0 を Table[0]〜Table[3], v_1 を Table[4]〜Table[7], v_2 を Table[8]〜Table[11], v_3 を Table[12]〜Table[15] で表す. 節点 v_0 の 0 枝は節点 v_3 を指しているので, Table[0 + 0] には節点 v_3 を表す部分配列の先頭の値である 12 を格納する. 同様に, 節点 v_0 の 1 枝は節点 v_2 を指しているので, Table[0 + 1] には節点 v_2 を表す部分配列の先頭の値である 8 を格納する. また, 節点 v_1 の 0 枝は終端節点 1 を指しているので, Table[4 + 0]



(a)QRMDD(2)

	0	1	2	3
v_0	0	1	2	3
v_1	4	5	6	7
v_2	8	9	10	11
v_3	12	13	14	15

	0	1	2	3
v_0	12	8	12	4
v_1	1	1	1	1
v_2	0	0	1	0
v_3	0	0	1	1

(b) テーブル

図 3　QRMDD(2) をデータ構造とするテーブル

には 1 を格納する．このような操作を繰り返し行うと, 図 3(a) の QRMDD(2) は図 3(b) 下部のようなテーブルで表現される．
(例終り)

以下に, テーブルを用いて論理関数を評価するための疑似コードを示す．

［アルゴリズム 5.1］（テーブルを用いた論理関数の評価） インデックス i 段目 $(i = 1, 2, \ldots, u)$ の入力値を $input_i$ とする．

（1） $ptr \leftarrow 0$, $i \leftarrow 1$.

（2） $ptr \leftarrow \mathrm{Table}[ptr + input_i]$.

（3） $i \neq u$ ならば, $i \leftarrow i + 1$ とし 2 へ戻る．

（4） $f \leftarrow ptr$.

MDD ではインデックスの値を参照する必要があるが, QRMDD の全てのパス長は等しいので, インデックスの値を参照する必要はなく, u 回のテーブル参照により関数値が求まる．

6. 多出力論理関数の表現

多出力論理関数の表現には, 特性関数を用いる．特性関数を用いることで, n 入力 m 出力の多出力関数を, $(n + m)$ 入力 1 出力の論理関数として扱うことができる．

［定義 6.1］ 入力変数を $X = (x_1, x_2, \ldots, x_n)$, 多出力関数を $F = (f_1(X), f_2(X), \ldots, f_m(X))$ とするとき,

$$\chi(X, Y) = \bigwedge_{i=1}^{m} (y_i \equiv f_i(X))$$

を満たす関数 $\chi(X, Y)$ を**多出力関数の特性関数**という．ここで y_i は出力を表す変数である．

特性関数を表現する BDD を, BDD_for_characteristic function(**BDD_for_CF**) という．BDD_for_CF の変数は入力を表す入力変数 $x_j (j = 1, 2, \ldots, n)$ と, 出力 $f_i (i = 1, 2, \ldots, m)$ の関数値を表す出力変数 y_i からなる．出力変数 y_i を表す節点の 2 本の枝のうち, 必ず 1 本は終端節点 0 を指し, もう一方の枝の値が f_i の関数値を表す．また, 出力変数を表す節点から終端節点 0 への枝は無視できる．

7. QRMDD_for_CF をデータ構造とする多出力論理関数の論理シミュレーション

テーブルに格納する次の節点へのアドレスの値は, 各インデックス i 段目 の各節点の間では, 互いに異なる番号を割り当て[(注1)], その節点に割り当てられた番号をアドレスとしてテーブルに格納する．

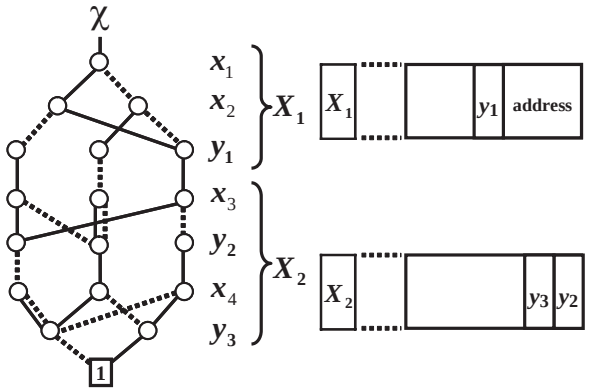


図 4　QRMDD(2)_for_CF を表すテーブル

QRBDD_for_CF の根節点から k 個ずつ終端節点へ向けて変数をグループ化する．このとき, QRMDD(k)_for_CF のインデックス i 段目に対応する変数をグループ化する際に, 出力変数 $y_j (j = 1, 2, \ldots, m)$ が現れた場合, y_j はインデックス i 段目に対応する変数とするが, グループ化する変数の数 k には含まない．

［例 7.1］ 図 4 に示す QRBDD_for_CF を, 根節点から 2 個ずつ終端節点へ向けて変数をグループ化し, QRMDD(2)_for_CF を表現するテーブルに格納する．出力変数 $y_j (j = 1, 2, 3)$ を表す節点は, 終端節点 0 への枝を無視すると, 出力変数 y_j を表す節点からの分岐はないとみなすことができる．出力変数を表す節点の関数値を表す枝の値をテーブルの未使用領域 (上位ビット) に格納することで, グループ化の際に, 出力変数を無視できる．つまり, インデックス 1 段目に対応する変数を (x_1, x_2, y_1), インデックス 2 段目に対応する変数を (x_3, y_2, x_4, y_3) とできる．このとき, インデックス 1 段目を表すテーブルには, インデックス 2 段目を表す節点へのアドレスを格納し, テーブルの未使用領域 (上位ビット) に出力変数 y_1 を表す節点の枝の値を格納する．インデックス 2 段目を表す節点は全て終端節点 1 を指すので, テーブルにアドレスを格納する必要は無く, 出力変数 y_2, y_3 を表す節点の枝の値を格納する．
(例終り)

インデックス i 段目 に対応する節点に出力変数が含まれる場合には, 出力変数の関数値を表す枝の値をアドレスの上位ビットに付加する．インデックス $i + 1$ 段目の節点の番号を格納するために必要なビット数は, インデックス $i + 1$ 段目の節点数を $node(i + 1)$ とすると, $\lceil \log_2 node(i + 1) \rceil$ ビットである．

［例 7.2］ QRMDD(2)_for_CF において, $node(i + 1) = 76$ であるとき, インデックス i 段目を表すテーブルのアドレス部に必要なビット数は, $\lceil \log_2 76 \rceil = 7$ ビットである．整数 int のうち第 0 ビット目から第 6 ビット目をアドレス部とする．インデックス i 段目に出力変数が含まれる場合は, 第 7 ビット目に出力

(注1)：通常の BDD では, 各節点に通し番号をつける．このため, 番号を表現するデータのビット数は多くなる．本手法では, 各インデックス毎に節点を区別する番号を付加するため, データのビット数は少なくてよい．

変数の関数値を表す枝の値を格納する．出力変数が複数含まれる場合には，第 8 ビット目，第 9 ビット目，...，と関数値を格納する． (例終り)

インデックス i 段目に出力変数が含まれる場合，インデックス i 段目のテーブル参照を行う際に，インデックス $i+1$ 段目の節点の番号と同時に関数値を読み出すことができ，出力変数を読み出す際のテーブル参照回数を削減できる．

以下に，論理関数の評価のための疑似コードを示す．

［アルゴリズム 7.1］（QRMDD_for_CF を用いた論理関数の評価）$page\_adr_i$ は，テーブル中のインデックス i 段目を格納している先頭のアドレス，$input_i$ はインデックス i 段目の入力値，adr_mask_i はインデックス i 段目のアドレス部のマスク，$output\_mask_i$ はインデックス i 段目の関数値のマスクとする．

（1） $ptr \leftarrow 0, i \leftarrow 1$.

（2） $Read\_data \leftarrow \mathrm{Table}[ptr + input_i + page\_adr_i]$.

（3） $ptr \leftarrow Read\_data \ \&\& \ adr\_mask_i$.

（4） インデックス i 段目に出力変数が含まれる場合，$output \leftarrow (Read\_data \ \&\& \ output\_mask_i) \gg output\_shift_i$.

（5） $i \neq u$ ならば，$i \leftarrow i+1$ とし 2 へ戻る．

8. 出力の分割

多出力論理関数を単一の BDD_for_CF で表現した場合，節点数が増大し，コンピュータのメモリに格納できない場合がある．そこで出力関数をいくつかのグループに分割し，各グループを別々の BDD_for_CF で表現する．出力関数を分割する際，それぞれの依存変数は少ない方が望ましい．そこで，依存変数が増加しないような順序に出力関数を並べ，次にその順序に従い，出力関数を分割する．また，各 BDD_for_CF に対し，BDD_for_CF の幅が最小となる変数順序最適化を行う．得られた BDD_for_CF を表現する QRMDD(k)_for_CF を構築し，データ構造を 1 つのテーブルに格納する．

［アルゴリズム 8.1］（出力の分割） m 出力論理関数の出力関数を依存変数が増加しないように並べた出力順序を $F = (f_1, f_2, \ldots, f_m)$，分割後の出力関数の部分集合の集合を $Z = \{G_1, G_2, \ldots, G_r\}$，BDD_for_CF の変数順序最適化前の節点数の閾値を $node_{TH}$，G を表現する BDD_for_CF の節点数を $node(G)$ とする．

（1） $Z \leftarrow \phi, i \leftarrow 1, j \leftarrow 1$.

（2） $G_j \leftarrow \phi$.

（3） $F \leftarrow F - \{f_i\}, G_j \leftarrow G_j \cup \{f_i\}, i \leftarrow i+1$.

（4） $F = \phi$ ならば 7 へ．

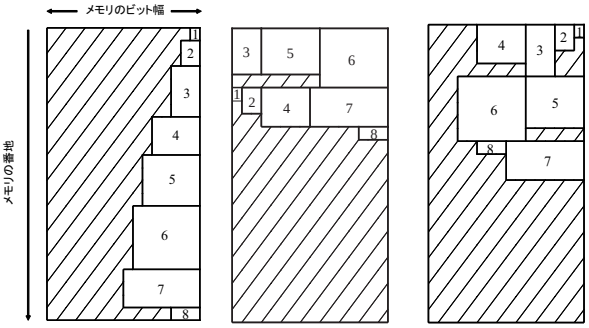
（5） G_j を表現する BDD_for_CF を構成する．$node(G_j) < node_{TH}$ ならば 3 へ戻る．

（6） $Z \leftarrow Z \cup G_j, j \leftarrow j+1$ とし 2 へ戻る．

（7） $G_j (j = 1, 2, \ldots, r)$ を表現する BDD_for_CF の変数順序最適化を行う．

9. メモリパッキング

図 5(a) に示すように QRMDD_for_CF のデータが各インデックスごとにテーブルに格納されているとする．図の斜線部分は未使用領域 (アドレスの上位ビットで値が全て 0 の領域) を表す．格納されているデータのビット数が整数 int のビット幅より小さい場合，未使用の上位ビット領域に異なるインデックスのデータを格納できる．この手法を**メモリパッキング**という．



(a)Dct パッキング前　(b)MP1 メモリ量最小　(c)MP2 インデックス順

図 5　メモリパッキング

図 5(a) に表される QRMDD_for_CF のデータを，必要メモリ量が最小となるようにテーブルに対しメモリパッキングを行うと，図 5(b) のようになる．このようなメモリパッキングを MP1 と略記する．また，インデックスの順にメモリパッキングを行うと，図 5(c) のようになる．このようにデータをパッキングすることでデータを順に読み出すことができ，キャッシュミスを削減できる．このようなメモリパッキングを MP2 と略記する．

メモリパッキングを行う場合，各インデックスごとの参照される確率の高い節点のデータをメモリブロックの上部に配置する．このとき，参照される確率の高いデータが読み出されると，他のインデックスの参照される確率の高いデータを同時に読み出すため，キャッシュミスを削減し，評価速度が高くなると考えられる．論理関数の評価時にはデータのシフト操作等が必要となる．このため，評価速度は低くなる可能性もあるが，実際はメモリ量が減るため，キャッシュミスを削減し，評価速度が高くなる場合が多い．

10. 実験結果

MCNC ベンチマーク関数を複数の QRMDD(k)_for_CF で表現し，そのデータ構造から C コードおよびテーブルを生成し評価した．多出力論理関数の出力を分割する際の節点数の閾値 $node_{TH}$ を 10 万個とした．k の値を変化させた場合のテーブル参照回数 $L(k)$[回] を表 1 に，テーブルを MP1 および MP2 で表現した場合の，テーブルの必要メモリ量 $M(k)$[KByte] を表 2 に，100 万回テストベクトルを与えたときの評価時間 $T(k)$[sec] の比較を表 3 に示す．実験環境は，IBM PC/AT 互換機 (CPU:Pentium4 Xeon 2.8GHz L1 instruction cache:12Kμops L1 data cache:8KB L2 cache:512KB メモリ:4GB)，OS は Redhat Linux 7.3，コンパイラは gcc version 3.2，最適化オプション O3 である．

表中の $Name$ は関数名を，表 1 中の In は入力数を，Out は出力数を，r は分割後の QRMDD(k)_for_CF の個数を表す．表 2, 3 中の Dct はメモリパッキングを適用しない場合を，MP1 は

表 1　QRMDD(k)_for_CF のテーブル参照回数 $L(k)$[回]

$Name$	In	Out	r	k				
				1	2	3	4	5
C432	36	7	1	36	18	12	9	8
C499	41	32	6	246	126	84	66	54
C880	60	26	4	178	90	61	46	36
C1908	33	25	5	157	81	53	43	33
C2670	233	140	9	427	214	147	108	91
C3540	50	22	6	265	134	90	70	54
C5315	178	123	12	826	417	280	212	172
C7552	207	108	9	856	430	289	216	173
rot	135	107	10	563	283	192	143	116
seq	41	35	3	116	59	41	31	25
apex6	135	99	8	311	159	107	82	66
des	256	245	49	1445	737	495	383	308
pair	173	137	15	704	356	239	182	146
frg2	143	139	10	368	186	128	95	76

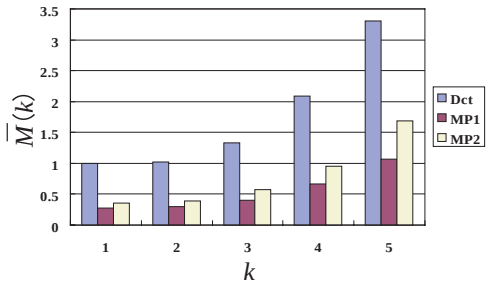


図 6　必要メモリ量の比較

QRMDD(k)_for_CF を MP1(メモリ量最小) で表現した場合を, MP2 は QRMDD(k)_for_CF を MP2(インデックス順) で表現した場合を示す. ave は各関数において, Dct(1) の場合を 1 として他の値を出し, その値を相加平均した値を表す.

表 1 において, $k = 1$ の場合のテーブル参照回数 $L(1)$ は, 高々 $In \times r$ となっている. $In \times r$ より小さくなるのは, BDD を分割した際, 依存変数の個数が減ったことによる. また, $k \geq 2$ の場合のテーブル参照回数 $L(k)$ は, ほぼ関係 $L(k) = \frac{L(1)}{k}$ を満たしている. 表 2 において, 必要メモリ量 $M(k)$ は $k = 1$ の場合と, $k = 2$ はほぼ等しいが, $k \geqq 3$ の場合は k とともに増加する. 表 3 において評価時間 $T(k)$ は k の値の単調減少関数となっている. メモリ参照時間が一定であるとすると, Dct の場合, $T(k) = \alpha L(k)$ の関係が成立しているはずであるが, 実際の評価時間はその関係を満たしていない. 比例関係が成立しないのは, キャッシュミス等による影響と思われる. また, Dct で必要なメモリ量 $M(1)$ を 1.00 としたときの, MP1 および MP2 の必要メモリ量の割合の平均 $\bar{M}(k)$ を図 6 に, Dct の評価時間 $T(1)$ を 1.00 としたときの, MP1 および MP2 の評価時間の割合の平均 $\bar{T}(k)$ を図 7 に示す.

図 6, 7 より明らかなように, データ構造として MP1 または MP2 を使用することにより, Dct に比べ必要メモリ量を 30%から 50%に削減でき, 評価時間も 60%から 70%に短縮できた. これは, メモリパッキングにより必要メモリ量が減少し, この結果

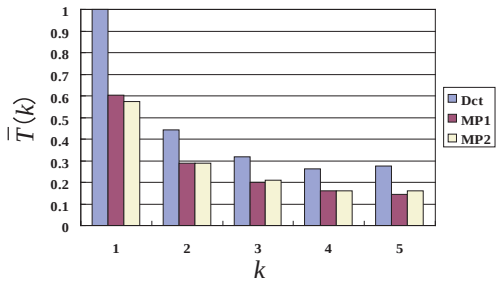


図 7　評価時間の比較

CPU のキャッシュミスも減少し, 評価時間を短縮できたと考えられる. また, MP2 は MP1 よりも余分にメモリ量を必要とした. これは, MP2 では QRMDD_for_CF のデータを順に並べたのに対し, MP1 は必要メモリ量を最小にするように詰め込んだためである. 平均すると, 必要メモリ量を最小とする MP1 が, MP2 に比べ評価時間を短縮できた. ただし, MP1, MP2 の必要メモリ量がほぼ等しい場合は, MP2 が MP1 よりも高速な場合がある. これは, MP2 は QRMDD_for_CF のデータを順に読み出すことが可能であるため, MP1 よりも評価時間を短縮できたと考えられる.

論理関数の評価の際, Dct の場合には, データをシフトする必要はないが, MP1 や MP2 の場合にはデータをシフトする余分の操作が必要となる. それにもかかわらず, MP1 や MP2 の方が高速となっている. また, 平均すると QRMDD_for_CF のデータを読み出す順序を考慮しメモリパッキングを行うよりも, 単にメモリ量を最小とするようにメモリパッキングを行う方が評価時間を短縮できるとわかった.

11. 結　　論

QRMDD_for_CF をデータ構造とする論理シミュレータを PC 上に実現し, メモリパッキングを適用した. メモリパッキングを適用することで, 必要メモリ量をもとの 30%から 50%に削減でき, その結果 CPU のキャッシュミスが減少し, 評価時間をもとの 60%から 70%に短縮できた. また, 平均すると, 必要メモリ量を最小とするメモリパッキングが, QRMDD_for_CF のデータを読み出す順序を考慮し, メモリパッキングを行う方法よりも評価時間を短縮できた.

12. 謝　　辞

本研究は一部, 文部科学省, 北九州地域知的クラスタ創成事業の補助金による.

文　　献

[1] P. Ashar and S. Malik, "Fast functional simulation using branching programs," *ICCAD*'95, pp.408-412, Nov, 1995.

[2] Y. Iguchi, T. Sasao, and M. Matsuura, "Evaluation of multiple-output logic functions using decision diagrams," *ASP-DAC* 2003, Kitakyusu, Jan 21-24, 2003, pp.312-315.

[3] P. C. McGeer, K. L. McMillan, A. Saldanha, A. L. Sangiovanni-Vincentelli, and P. Scaglia, "Fast discrete function evaluation using decision diagrams," *ICCAD*'95, pp.402-407, Nov 1995.

[4] S. Nagayama and T. Sasao, "Code generation for embed-

表 2　必要メモリ量 $M(k)$ の比較 [KByte]

Name	k														
	1			2			3			4			5		
	Dct	MP1	MP2	Dct	MP1	MP2	Dct	MP1	MP2	Dct	MP1	MP2	Dct	MP1	MP2
C432	24	6	8	27	8	8	33	11	14	58	20	20	88	31	41
C499	206	51	66	208	52	78	278	71	121	433	111	196	694	182	318
C880	164	51	61	165	56	68	226	76	92	330	124	153	521	202	329
C1908	164	51	59	174	60	75	201	72	116	382	148	206	545	233	400
C2670	618	216	261	617	215	254	809	283	345	1202	425	571	1953	702	1031
C3540	1224	534	634	1212	518	653	1638	670	928	2432	970	1501	3795	1356	2188
C5315	1852	713	903	1841	707	944	2598	1003	1408	3625	1343	1930	5966	2181	3744
C7552	494	143	167	499	149	174	685	211	251	978	303	381	1652	525	679
rot	202	48	64	206	51	72	268	68	105	417	109	175	616	151	261
seq	20	4	5	20	4	6	27	6	9	41	10	16	68	18	27
apex6	29	5	7	30	5	8	38	7	12	58	12	21	91	19	39
des	172	32	52	175	33	59	228	45	91	348	68	150	572	118	265
pair	222	56	74	225	61	82	284	79	112	474	144	208	735	232	378
frg2	54	15	20	55	19	24	73	26	32	122	55	65	196	89	99
ave	1.00	0.28	0.35	1.02	0.30	0.39	1.34	0.40	0.57	2.09	0.67	0.95	3.30	1.07	1.69

表 3　評価時間 $T(k)$ の比較 [sec]

Name	k														
	1			2			3			4			5		
	Dct	MP1	MP2	Dct	MP1	MP2	Dct	MP1	MP2	Dct	MP1	MP2	Dct	MP1	MP2
C432	0.24	0.17	0.15	0.08	0.09	0.09	0.06	0.06	0.06	0.06	0.05	0.04	0.05	0.04	0.05
C499	2.83	2.21	2.28	1.18	0.91	1.04	0.82	0.56	0.70	0.87	0.39	0.45	1.07	0.30	0.47
C880	1.41	0.81	0.89	0.59	0.36	0.38	0.37	0.23	0.23	0.20	0.17	0.19	0.21	0.13	0.15
C1908	1.29	0.71	0.72	0.46	0.34	0.35	0.25	0.20	0.20	0.18	0.18	0.18	0.19	0.15	0.16
C2670	5.56	4.08	3.33	2.85	1.79	1.74	1.95	1.23	1.22	1.68	0.99	0.98	1.72	0.98	1.01
C3540	6.39	3.17	3.36	2.73	1.23	1.36	1.94	0.91	1.17	1.65	0.80	0.97	1.61	1.01	0.99
C5315	24.99	14.23	12.20	11.98	6.61	6.41	9.71	5.33	5.71	8.88	4.98	5.31	8.29	4.90	5.35
C7552	12.27	6.45	5.83	6.71	4.11	3.64	4.73	2.84	2.71	4.15	2.24	1.77	4.41	1.93	2.30
rot	6.08	4.54	4.14	2.77	1.90	1.79	2.57	1.24	1.23	2.02	1.04	1.06	1.97	0.84	0.90
seq	0.68	0.40	0.38	0.17	0.22	0.23	0.14	0.15	0.15	0.12	0.14	0.13	0.11	0.11	0.11
apex6	2.50	1.03	1.03	1.20	0.52	0.54	0.79	0.38	0.38	0.54	0.32	0.32	0.45	0.28	0.31
des	11.73	6.61	6.29	6.34	3.11	3.41	4.54	2.48	2.54	3.75	1.38	1.91	5.60	1.56	1.53
pair	6.97	4.94	4.38	3.35	2.56	2.03	2.71	1.86	1.96	2.21	1.19	1.21	2.56	1.04	1.14
frg2	2.84	1.41	1.43	1.45	0.68	0.75	0.84	0.53	0.54	0.66	0.46	0.44	0.58	0.44	0.46
ave	1.00	0.60	0.57	0.44	0.29	0.29	0.32	0.20	0.21	0.26	0.16	0.16	0.27	0.14	0.16

ded systems using heterogeneous MDDs," *SASIMI* 2003, Hiroshima, April 3-4, 2003, pp.258-264.

[5] S. Nagayama, T. Sasao, Y. Iguchi, and M. Matsuura, "Area-time complexities of multi-valued decision diagrams," *IEICE Transactions on Foundamentals of Electronics*, Vol.E87-A, No.5, pp.1020-1028, May, 2004.

[6] S. Nagayama, T. Sasao, Y. Iguchi, and M. Matsuura, "Representations of logic functions using QRMDDs," *Proc. International Symposium on Multiple-Valued Logic*, pp.261-267, Boston, Massachusetts, U.S.A, MAY 2002.

[7] H. Nakahara and T. Sasao, "A PC-based logic simulator using a look-up table cascade emulator," *IEICE Transactions on Foundamentals of Electronics, Communications and Computer Sciences*, Vol. E89-A, No.12, Dec. 2006, pp.3471-3481, Special Section on VLSI Design and CAD Algorithms.

[8] T. Sasao, H. Nakahara, M. Matsuura, and Y. Iguchi, "Realization of sequenial circuits by look-up table rings," *IEICE, Transactions on Foundamentals of Electronics*, vol.E87-A,No.12, pp.3141-3150.

[9] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-Output function for reconfigurable hardware," *IWLS-2001*, Lake Tahoe , CA, June 12-15,2001, pp.225-300.

[10] T. Sasao and M. Matsuura, "A method to decompose multiple-output logic functions," *41st Design Automation Conference*, San Diego, CA, USA, June 2-6, 2004, pp.428-433.

[11] T. Sasao, and J. T. Butler, "A method to represent multiple-output switching functions by using multi-valued decision diagrams," *Proc. International Symposium on Multiple-Valued Logic*, pp.248-254, Santiago de Compostela, Spain, May 1996.

[12] T. Sasao, M. Kusano, M. Matsuura, "Optimization methods in look-up table rings," *IWLS-2004*, pp.431-437, Temecula, California, U.S.A., June 2004.