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.

先読みヘテロジニアスMDDマシンについて

中原　啓貴† 　笹尾　勤† 　松浦　宗寛†

† 九州工業大学 情報工学部 〒820–8502 福岡県飯塚市大字川津 680–4

あらまし　先読みヘテロジニアス MDD マシンについて述べる．まず, ヘテロジニアス MDD について述べ, ヘテロジニアス MDD を模擬するマシン (標準ヘテロジニアス MDD マシン) について述べる．次に, インデックスを先読みする方法について述べる．そして, 与えられた論理関数に対して, メモリを効率よく使用し平均実行時間を最小にするコードを生成する手法を述べる．FPGA と外付けメモリを用いて標準ヘテロジニアス MDD マシンと先読みヘテロジニアス MDD マシンを実装した．インデックスを先読みすることで, 制御回路が簡単になり, 動作周波数を 18.2%増加できた．また, MCNC ベンチマーク関数を用いて比較を行った結果, 先読み HMDDM は QDDM と比較して 9.57-11.85 倍高速であり, Core2Duo と比較して 16.22-20.08 倍高速であった．

On a Prefetching Heterogeneous MDD Machine

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

† Department of Computer Science and Electronics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan

Abstract　This paper shows a heterogeneous multi-valued decision diagram machine (HMDDM). First, we introduce a standard heterogeneous multi-valued decision diagram (HMDD). Then, we show a method to prefetch index for the HMDDM. Next, we introduce a code generation method for the HMDDM utilizing given memory size efficiently. We implemented the standard HMDDM and the prefetching HMDDM on an FPGA. The implementation results show that the prefetching HMDDM is 18.2% faster than the standard HMDDM. Also, we compared with Intel's Core2Duo (1.2 GHz) and a quartary decision diagram machine (QDDM). As for the execusion time, the prefetching HMDDM is 9.57-11.85 times faster than the QDDM, and 16.22-20.08 times faster than the Core2Duo.

1. はじめに

決定グラフマシン [1], [7] とは決定グラフ (DD: Decision Diagram) を模擬する特定用途プロセッサである．様々な決定グラフ (BDD [2], MDD (k) [6], QRBDD [15], QRMDD [4], ヘテロジニアス MDD (HMDD) [8]) を模擬するマシンが提案されている [1], [5], [11], [14], [16]．決定グラフマシンの応用として, BDD マシンを用いた産業用シーケンサ [18], 多値決定グラフ (MDD) に基づくパイプライン化した論理シミュレーションアクセラレータ [5], 並列 QDD マシンを用いたパケット分類器が報告されている [10].

本論文では, 決定グラフマシンの高速化を考える．最も単純な高速化法は, 決定グラフマシンのマルチコア化である．しかしながら, 以下の問題点が挙げられる．

a） チップ面積増加によるコスト増加

マルチコア化によるチップ面積の増加に伴い, コストが増加する．例えば, 文献 [12] では 4 分岐決定グラフマシンを 128 台並列に用いたマシンを実装しているが, 実装に用いたハイエンド FPGA の価格は極めて高価(注1)である．ハイエンド FPGA にメニーコアを実装すれば, 高性能を容易に達成できるが, 価格がネックとなり現実的でない．

b） 並列化の難しさ

ストリーミング処理等では, データを並列処理できるため, マルチコアを用いて高速に処理できる．しかしながら, 処理を分散したときに負荷がばらつく応用では, マルチコアの性能は最も負荷が高い処理に拘束されてしまう．文献 [12] では 128 台並列に用いた決定グラフマシンを用いたが, 並列処理の困難なベンチマーク関数では, 約 20 倍しか高速化できていない．

本論文では, ヘテロジニアス多値決定グラフ (Heterogeneous Multi-valued DD: HMDD) マシンを用いてシンプルかつ高速な決定グラフマシンを実現する．HMDD は各節点のサイズを自由に設定できるため, メモリ量を増加することにより, 高速に評価できる．文献 [5] のように, 小さな FPGA と外付け SRAM で決定グラフマシンを用いれば, 低価格(注2)で実現できる．大規模 FPGA や ASIC と比較して, 外付けメモリは低価格なので, 大容量の外付けメモリを用いたとしても, 許容できる価格である．外付けメモリはワード数が 2 の冪乗個である, QDD マシンや BDD マシンでは全て使い切ることは難しいが, HMDD で

(注1)：Altera 社 Stratix III(EP3S340H1152C4) の場合．2010 年 9 月, Digikey での販売価格は 100 万円程度.

(注2)：2010 年 9 月, Digikey での販売価格は, 16MbitSRAM で数百円, ローエンド FPGA (Cyclone III) で 1000 円程度.

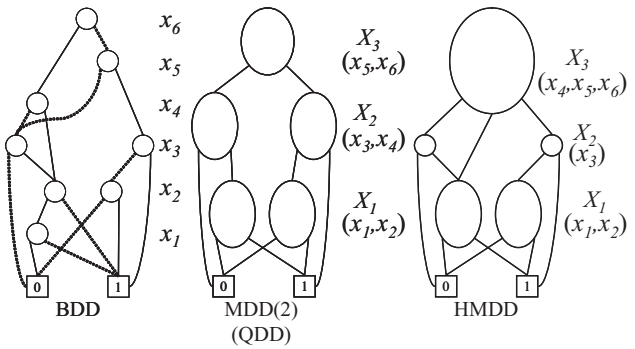

図 1　決定グラフ.

は, 変数の分割法を最適化することで, 与えられたメモリを殆ど全てを有効に活用できる. 本論文では, HMDD マシンを高速化するインデックスの先読み機能を提案する. そして, 提案 HMDD マシンと汎用の PC や 4 分岐決定グラフマシン (QDD マシン) との比較を行う.

第 2 章で本論文で用いる用語の定義を行い, 第 3 章でヘテロジニアス MDD マシンの構成について述べ, 第 4 章でヘテロジニアス MDD の生成アルゴリズムについて述べ, 第 5 章で実験結果について述べ, 第 6 章で本論文のまとめを行う.

2. 諸 定 義

2.1 論理変数の分割

[定義 2.1]　単一出力論理関数を $f(X) : B^n \to B, B = \{0,1\}$ とする. ここで, $X = (x_1, x_2, \ldots, x_n), x_i \in B$ は f の入力変数である. X の変数の集合を $\{X\}$ で表す. $\{X\} = \{X_1\} \cup \{X_2\} \cup \cdots \{X_u\}$ かつ $\{X_i\} \cap \{X_j\} = \phi (i \neq j)$ のとき, $(X_1, X_2, \ldots, X_u)$ を **X の分割**という. また, X_i を**超変数**という. $k_i = |X_i| (i = 1, 2, \ldots, u)$ とすると $k_1 + k_2 + \cdots + k_u = n$ である. k_i を**超変数 X_i のサイズ**という.

2.2 決定グラフ (DD: Decision Diagram)

論理関数 f に対して次に定義する**シャノン展開**を繰返し適用することで **2 分決定グラフ (BDD: Binary Decision Diagram)** を得る.

[定義 2.2]　任意の論理関数 $f(x_1, x_2, \ldots, x_n)$ は次のように展開できる.

$$f(x_1, x_2, \ldots, x_i, \ldots, x_n) = \bar{x}_i f(x_1, x_2, \ldots, 0, \ldots, x_n) \vee x_i f(x_1, x_2, \ldots, 1, \ldots, x_n).$$

シャノン展開を適用する変数の順序を**変数順序**という.

[定義 2.3]　BDD は節点と枝から構成され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 に対応する変数のインデックスを $index(v) \in \{0, 1, \ldots, n\}$ と表記する. $index(v) = 0$ のとき, 節点 v を**終端節点**といい, $index(v) \neq 0$ のとき, 節点 v を**非終端節点**という. 終端接点は 0 または 1 の 2 値を取り, 論理関数 f の関数値に対応する. 非終端節点 v は 2 つの子節点 $low(v), high(v) \in V$ に接続する枝を持つ. 変数順序を固定した BDD を**順序付き BDD (OBDD: Ordered BDD)** という.

[定義 2.4]　**多値決定グラフ (MDD(k): Multi-valued DD)** とは各非終端節点が 2^k 個の枝を持つ決定グラフである.

[定義 2.5]　**既約順序付き BDD (ROBDD: Reduced Ordered BDD)** とは OBDD に対して以下の 2 つの簡単化手法を適用して得られる決定グラフである.

1. 等価なサブグラフを共有する
2. 節点 u の出力枝が指す節点が, 節点 v の出力枝が指す節点と等しい場合, 節点 u を削除し, 節点 u の入力枝を節点 v に接続する.

ROBDD と同様に ROMDD(k) も定義できる. 本論文では断らない限り BDD, MDD(k) はそれぞれ ROBDD, ROMDD(k) のことを指すものとする. MDD(1) とは BDD のことを指す. 文献 [9] には, $MDD(k)$ に関して MDD(2) が面積遅延時間積において最も優れていることが示されている. 本論文では MDD(2) を実現するマシンを比較対象とする. $k = 2$ のとき, 各節点の分岐数は $2^2 = 4$ となるので, MDD(2) を特に **QDD: Quarternary DD** とよぶ [14]. 以降, 本論文では QDD マシンを MDD(k) マシンの代表とする.

[定義 2.6]　決定グラフの根節点から終端節点までの経路を**パス (path)** と呼ぶ. パス上に現れる枝の個数を**パス長**という.

[定義 2.7]　変数の分割を $X = (X_1, X_2, \ldots, X_u)$ とする. 各節点 i の入力数を $k_i = |X_i|$ とするとき, $k = |X_1| = |X_2| = \cdots = |X_u|$ となる MDD を**ホモジニアス MDD (homogeneous MDD)** と呼び, $MDD(k)$ と表記する. 一方, 各 k_i が必ずしも等しくない MDD を**ヘテロジニアス MDD (HMDD: Heterogeneous MDD)** と呼ぶ.

[例 2.1]　図 1 に MCNC ベンチマーク関数 [17]C17 の BDD, QDD, HMDD を示す. ■

2.3 平均パス長 (Average Path Length)

各節点の評価時間が一定であると仮定すると, 決定グラフの評価時間は**平均パス長** [3] に比例する. 決定グラフマシンでは各節点を一定時間で評価できる. 従って, 評価時間は平均パス長に比例すると仮定できる.

[定義 2.8]　決定グラフの変数 X の分割を $(X_1, X_2, \ldots, X_u)$ とする. X_i を r 値の論理変数とし $c \in \{0, 1, \ldots, r-1\}$ とする. ここで, $r = k^{k_i}$ である. $P(X_i = c)$ は X_i が c となる確率を表す. r 値の変数によってパス p_i を通過する確率を**パス確率 (PP: Path Probability)** とし, $PP(p_i)$ と表記する. このとき

$$PP(p_i) = \sum_{\vec{c} \in C_i} P(X_1 = c_1) \times P(X_2 = c_2) \times \cdots \times P(X_u = c_u)$$

である. ここで C_i はパス p_i を通過する変数 X の集合を現し, $\vec{c} = (c_1, c_2, \ldots, c_u)$ である.

[定義 2.9]　決定グラフの**平均パス長 (APL: Average Path Length)** とは以下の式で与えられる.

$$APL = \sum_{i=1}^{N} PP(p_i) \times l_i.$$

ここで N はパス数を表し, l_i はパス p_i のパス長を表す.

3. インデックスの先読みを行うヘテロジニアス MDD マシン

本論文ではインデックスの先読みを行うヘテロジニアス MDD マシン (HMDDM) を提案する. 提案手法の有効性を示すため, 他のブランチングマシンとの比較を行う. 文献 [11] では, QDD マシン (QDDM) が面積遅延時間積に関して, 最も優れていることを示した. 提案手法との比較を行うため, QDDM を説明する. 次に, HMDDM を説明し, 提案手法である超変数を先読みする HMDDM を説明する.

3.1 QDD Machine (QDDM)

図 2 に QDD を模擬するマシン (QDDM) を示す. QDDM が BDDM と異なるのは分岐数が 4, 及び分岐を決定するための変数が 2 ビット ($|X_i| = 2$) である. 図 3 に QDDM の命令セットを示す. QDDM は非終端節点を模擬する 4-Branch 命令と終端接点を模擬する Output 命令を用いる.

3.2 直接分岐方式と間接分岐方式

決定グラフマシンは 1 節点の評価に関して, **直接分岐方式**と**間接分岐方式**に大別される. ホモジニアス DD マシン (BDDM, QDDM) では各節点の入力数が一定であるから, 分岐数も一定である. 従って, 各節点の命令語長は均一である. 一方, HMDD では各節点の入力数を自由に設定できるため, 分岐数も不ぞろいである. よって, 各節点の命令語長は不ぞろいになってしまいメモリ使用効率が悪い.

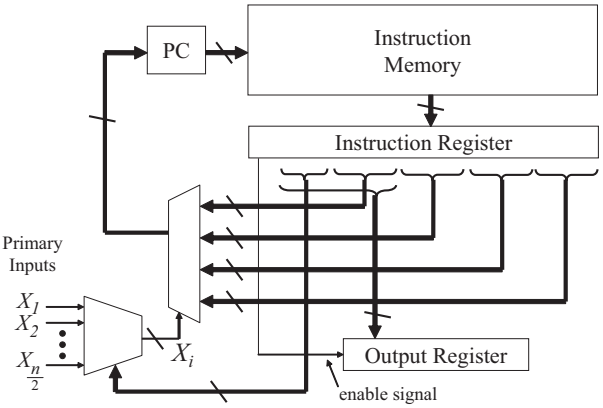

図 2　QDDM.

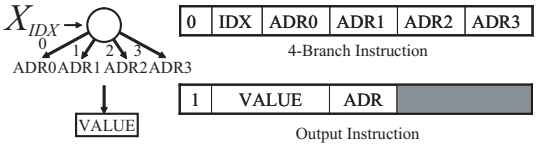

図 3　QDDM の命令セット.

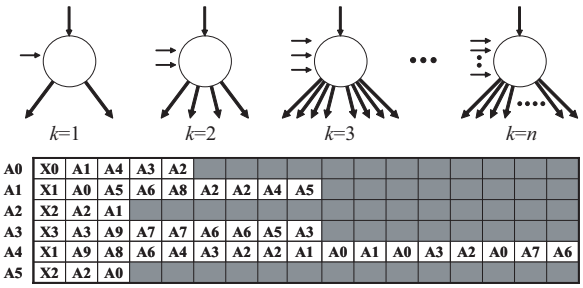

図 4　HMDD を直接分岐方式で実現した例.

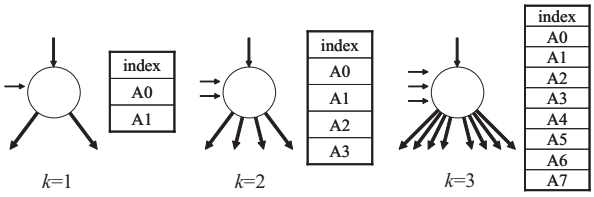

図 5　HMDD の非終端節点を模擬する標準間接分岐方式.

[例 3.2]　図 4 は HMDD を直接分岐方式で実現した例である. 節点の入力数 k が増えるに従って, 分岐数が 2 倍づつ増加する. よって, 節点によってはメモリに未使用部分が生じメモリ使用効率が悪い. ■

HMDD のメモリ使用効率を改善するため**間接分岐方式**を導入する [11]. 分岐を行うため, 現在の節点のインデックスに応じたアドレスを間接参照し, 分岐先のアドレスを読み出す. 直接分岐方式では 1 つの節点を評価するためにメモリ参照が 1 回でよいが, 標準間接分岐方式ではメモリ参照を 2 回行う. ただし, メモリを効率よく利用できる.

[例 3.3]　図 5 に HMDD の非終端節点を模擬するための標準間接分岐方式を示す. *index* は各節点に対応するインデックスを格納するフィールドである. 以下に, HMDD の非終端節点を模擬する手順を示す.

1.　*index* を読み出し, 現在のアドレスに加算して間接参照するアドレスを求める.

2.　1. で求めたアドレスを参照し, 分岐するアドレスを読み出す.

3.　2. で求めたアドレスにジャンプする. ■

図 5 からも明らかなように, 標準間接分岐方式では超変数 X_i のサイズ k_i が異なっていてもデータを密に格納でき, メモリを効率よく利用できる.

3.3　HMDD マシン (HMDDM)

図 6 に**標準 HMDD** を模擬するマシン (HMDDM) を示す.

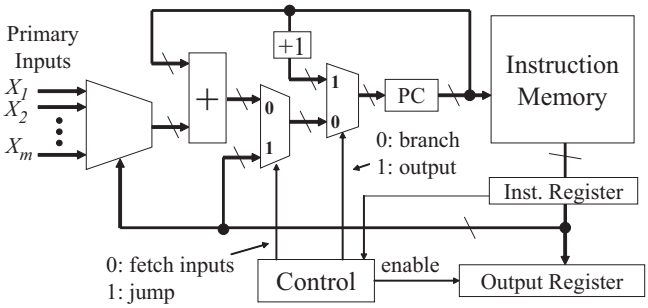

図 6　標準 HMDDM.

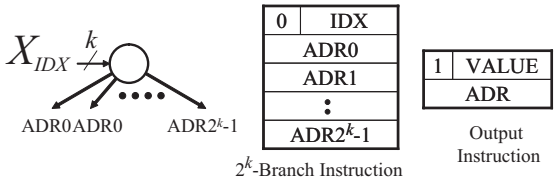

図 7　標準 HMDDM の命令セット.

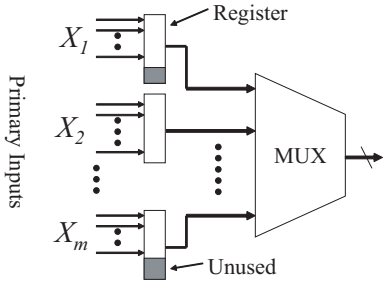

図 8　HMDDM の入力変数保持回路.

また, 図 7 に標準 HMDDM の命令セットを示す. 図 6 において, **命令メモリ (Instruction memory)** は命令を保持し; **命令レジスタ (Instruction register)** は命令メモリから読み出した命令を保持し; **出力レジスタ (Output register)** は出力値を保持し; **プログラムカウンタ (PC)** は現在評価を行っている節点のアドレスを保持する. 標準 HMDDM は分岐命令と出力命令を持つ. 分岐命令は間接分岐方式を実行するため, 入力を取り込む**フェッチモード (fetch mode)** と分岐を実行する**ジャンプモード (jump mode)** を実行する. 出力命令の場合は出力を実行する**出力モード (output mode)** を実行する. 制御回路を用いて 3 通りのモードを切り替える. 標準 HMDDM は入力を選択するマルチプレクサを持つ. また, 3 通りのモードを切り替えるためのマルチプレクサを用意する. フェッチモード時に間接アクセスを行うため加算器を用いる. また, 出力モード時は出力命令の次のアドレスを読込み, その値にジャンプするため, アドレスを 1 つ増加させる回路を用意する.

HMDDM では超変数のサイズが異なるため, 最大サイズの超変数を保持するレジスタとその値を選択するマルチプレクサを用意する. 図 8 に HMDDM の入力変数保持回路を示す.

[アルゴリズム 3.1]　図 9,10,11,12 に標準 HMDDM の分岐命令の実行を示す.

1.　フェッチモードの実行

1.1　PC を参照し命令メモリを読み出す (図 9).

1.2　入力変数と PC のカウンタを加算するため, 制御回路は制御信号を出力する. 間接参照するアドレスを PC に取り込む (図 10).

2.　ジャンプモードの実行

2.1　PC を参照しジャンプアドレスを読み出す (図 11).

2.2　ジャンプを行うため, 制御回路は制御信号を出力する. ジャンプアドレスを PC に取り込む (図 12).

[アルゴリズム 3.2]　図 13,14,11,12 に標準 HMDDM の出力命令の実行を示す.

1.　出力モードの実行

1.1　PC を参照し命令メモリを読み出す (図 13).

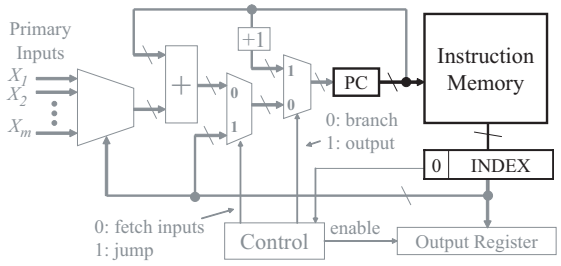

図 9　フェッチモードの実行 (標準 HMDDM).

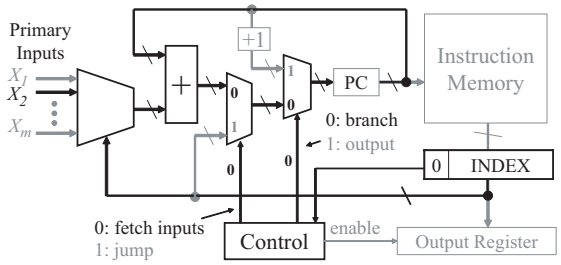

図 10　フェッチモードの実行 (標準 HMDDM).

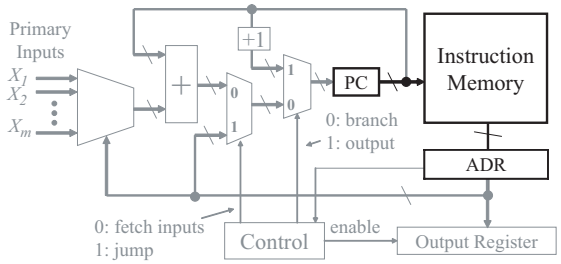

図 11　ジャンプモードの実行 (標準 HMDDM).

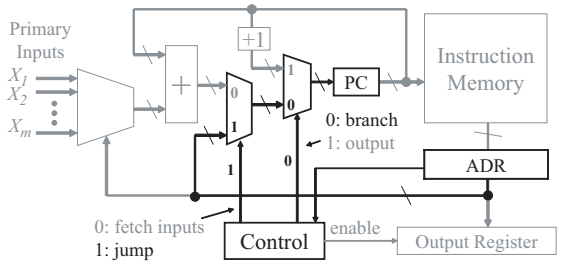

図 12　ジャンプモードの実行 (標準 HMDDM).

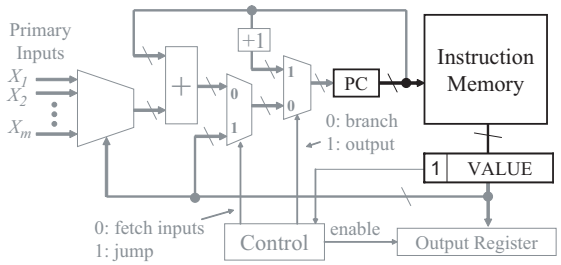

図 13　出力モードの実行 (標準 HMDDM).

1.2　制御回路は制御信号を出力し, 現在のアドレスを 1 つ増やし PC に取り込む (図 14).

2.　ジャンプモードの実行はアルゴリズム 3.1 で示した標準 HMDDM のジャンプモードと同じ.

3.4　先読みヘテロジニアス MDD マシン

図 6 に示した標準 HMDDM では, まず, 最初のステップでジャンプを行い, 次のステップでジャンプした節点のインデックスを読込む. そのため, 動作が遅くなる. **先読み HMDDM** では, ジャンプ先の節点のアドレスとそのインデックスを同時に読込む. ジャンプ先の節点のインデックスを先読みすることで, HMDD の 1 節点を 1 回のメモリ参照で評価できるため, 論理関数を高速に評価できる. ただし, 各ワードにインデックス

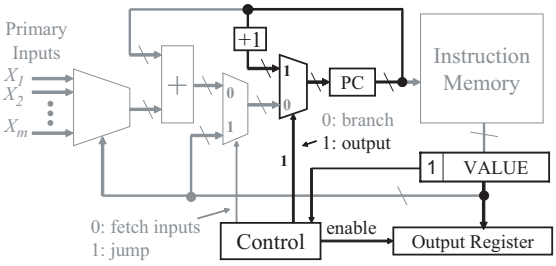

図 14　出力モードの実行 (標準 HMDDM).

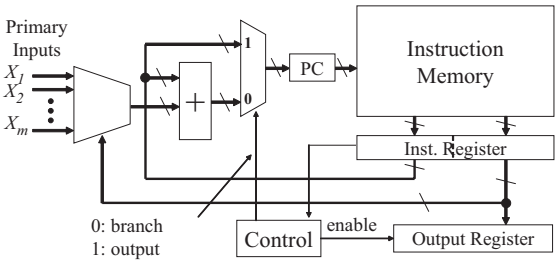

図 15　先読み HMDDM.

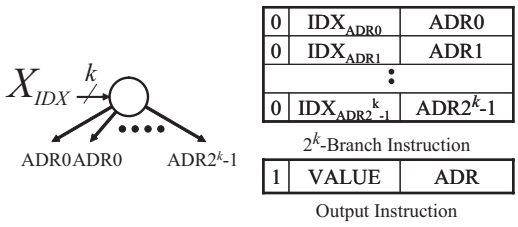

図 16　先読み HMDDM の命令セット.

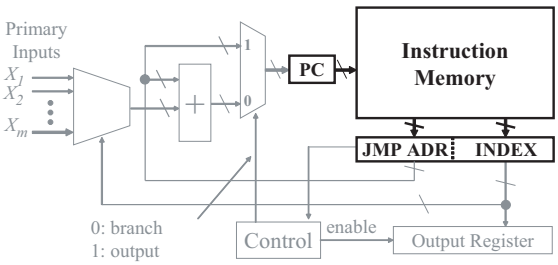

図 17　分岐モード (先読み HMDDM).

を重複して保持するため, 標準 HMDDM と比較して命令メモリの語長が増え, 総メモリ量も増える欠点がある.

図 15 に先読み HMDDM を示す.　また, 図 16 に先読み HMDDM の命令セットを示す. 図 15 において, 命令メモリ, 命令レジスタ, 出力レジスタ, 及びプログラムカウンタは図 6 に示した標準 HMDDM と同じである.

先読み HMDDM では, 分岐命令はジャンプアドレスとインデックスを同時に読込む**分岐モード**で実行される. また, 出力命令では, ジャンプと出力を同時に行う**出力モード**によって実行される. 以下に, 分岐命令と出力命令の実行を示す.

［アルゴリズム 3.3］　図 17,18,19 に先読み HMDDM の分岐命令の実行を示す.

1.　分岐モードの実行

1.1　PC を参照し命令メモリを読み出す (図 17).

1.2　分岐先の節点のインデックスを選択する (図 18).

1.3　分岐先のアドレスとインデックスで指定される変数の値を加算し, 次に分岐する節点のアドレスを PC に格納する (図 19).

［アルゴリズム 3.4］　図 20 に先読み HMDDM の出力命令の実行を示す.

1.　出力モードの実行

1.1　PC を参照し命令メモリを読み出す (図 17).

1.2　出力値を出力レジスタに格納する. 同時に, ジャンプ先アドレスを PC に格納する (図 20).

アルゴリズム 3.3 に示すように, インデックスの先読みを行

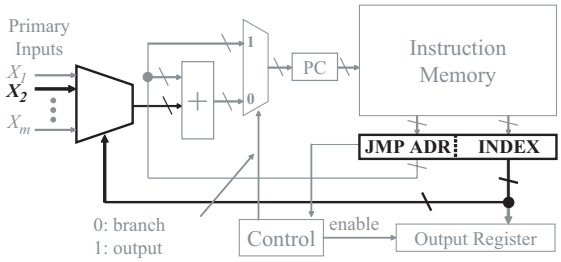

図 18　分岐モード (先読み HMDDM).

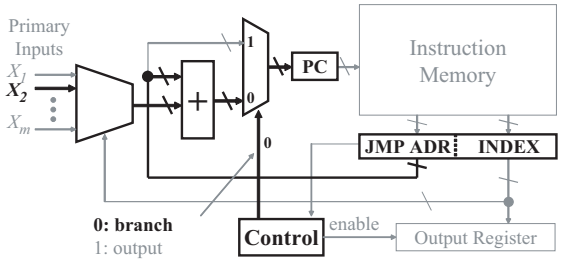

図 19　分岐モード (先読み HMDDM).

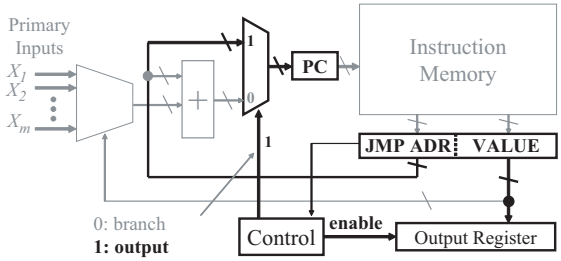

図 20　出力モード (先読み HMDDM).

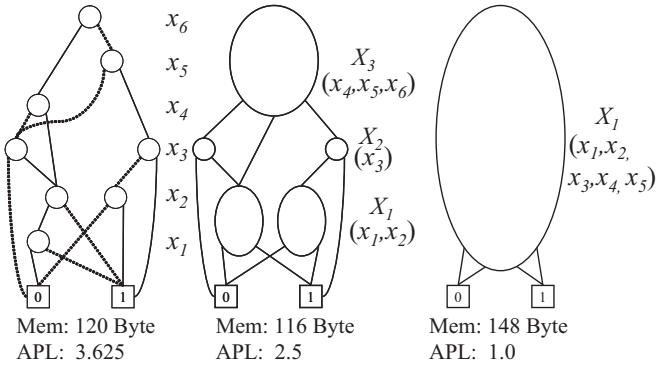

図 21　HMDD の例.

うことで, 1 ステップで HMDD の節点を評価できる. 図 6, 15 に示すように, 先読み HMDDM ではアーキテクチャが単純であるため, 高速にできる.

4. ヘテロジニアス MDD マシンのコード生成法

4.1 ヘテロジニアス MDD を用いた論理関数の表現

一般に, 多出力関数を単一の決定グラフで表現すると, 節点数が大幅に増大し表現できない. そこで, 本論文では出力関数を単一出力に分割後, 決定グラフで表現する.

ヘテロジニアス MDD (HMDD) では, 節点の超変数のサイズを任意に決定できるため, 様々な尺度を最適にする HMDD を構成できる.

[例 4.4]　図 21 に様々な超変数のサイズを持つ HMDD を示す. *Mem* はメモリ量を表し, *APL* は平均パス長を表す. 超変数のサイズを小さくすると, メモリ量は減少するが APL は増加する. しかし, 超変数のサイズを小さくしすぎると, メモリ量は増加する. 超変数 1 個に全ての変数を纏めた場合, APL は 1 になるがメモリ量は大きくなりすぎてしまう. 従って, 適切な超変数のサイズを求めなければならない. ■

1: **DynaHMDD**(F,M_{HMDDM},n,m)
2: **Input**
3: $F = (f_1, f_2, \ldots f_m)$: n 入力 m 出力論理関数
4: M_{HMDDM}: HMDDM のメモリ上界
5: **Output**
6: 総 APL
7: $APL_0 \leftarrow \{0\}$, $Mem_0 \leftarrow \{0\}$.
8: for $i \leftarrow 1$ to m do
9: (f_i を表現する BDD を作成).
10: (f_i のメモリ量を最小にする変数順序を求める).
11: $Apl_i(j) \leftarrow \phi$, $Mem_i(j) \leftarrow \phi$. ここで, $j \in \{1, 2, \ldots, n\}$.
12: $Apl_i(j) \leftarrow Apl_{i-1}(1)$, $Mem_i(j) \leftarrow Mem_{i-1}(1)$.
13: for $j \leftarrow 1$ to n do
14: $apl \in Apl_i(n-j)$, $Apl_i(n-j) \leftarrow Apl_i(n-j) - \{apl\}$.
15: $mem \in Mem_i(n-j)$, $Mem_i(n-j) \leftarrow Mem_i(n-j) - \{mem\}$.
16: for $k \leftarrow 1$ to $\lceil \log_2 M_{HMDDM} \rceil$ do
17: if(mem + (インデックス $n-j$ から k 入力の節点を作ったときのメモリ量)$\leqq M_{HMDDM}$
&& ($Apl(n-j+k)$ にインデックス $n-j$ から k までの APL が存在しない場合)
18: $Mem_i(n-k+k) \leftarrow Mem_i(n-j+k) \cup \{mem\}$.
19: $Apl_i(n-k+k) \leftarrow Apl_i(n-j+k) \cup \{apl\}$.
20: endif
21: ($Apl_i(n-j) \neq \phi$ ならば, Step.17. に戻る).
22: endfor
23: endfor
24: endfor
25: APL_m の中から, 最小の apl を出力し, 停止.

図 22　APL 最小の HMDD を求めるアルゴリズム.

4.2 HMDD を構成するアルゴリズム

本論文では, 与えられたメモリ上界の下で, 平均実行時間 (APL) を最小とする HMDD の構成法を考える. 入力変数の順序は固定されていると仮定するとき, n 変数論理関数を表現する HMDD は 2^{n-1} 個存在する. m 出力関数を単一出力関数に分割し, 各出力関数を HMDD で表現する場合, $2^{m(n-1)}$ 個の HMDD が存在する. 従って, n や m が大きい場合, 最適な HMDD を網羅的に求めるのは困難である.

本論文では, ダイナミック・プログラミングを用いて, 与えられたメモリ上界の下で総 APL が最小となる HMDD を構成する. まず, 問題の定式化を行う.

[問題 4.1]　m 出力論理関数 $F = (f_1, f_2, \ldots, f_m)$ が m 個の BDD で表現されている. APL_i を f_i を表現する HMDD の APL とし, MEM_i を f_i を表現する HMDD のメモリ量とする. 以下の条件を満たす HMDD を求めよ:

1. $\sum_{i=1}^{m} APL_i$ が最小.
2. $\sum_{i=1}^{m} MEM_i \leqq$ 与えられたメモリ上界.

[アルゴリズム 4.5]　図 22 に, 問題 4.1 を解くアルゴリズムを示す.

アルゴリズム 4.5 は与えられたメモリ量の制約の下で, 所与の論理関数を表現する BDD に対して, 総 APL が最小となる HMDD を構成する. 出力関数の分割の最適化については, 今後の課題である.

5. 実 装 結 果

5.1 ヘテロジニアス MDD マシンの実装

標準 HMDDM と先読み HMDDM を Altera 社 Cyclone III スタータキット (FPGA: Cyclone III, EP3C25) に実装し, LE 数と最大動作周波数を比較した. 合成ツールは Altera 社 Quartus II version.9.1 を用いた. 表 2 に実装結果を示す. 表 2 より, インデックスの先読みを行うことで制御部の LE 数を 31.3%削減でき, 動作周波数を 18.2%増加できた. 先読み HMDDM がコンパクトかつ高速で動作するのは, インデックスとジャンプ

表 1　他の手法との比較.

Name	In	Out	FF	APL	C2D@1.2GHz		QDDM		HMDDM+1MB SRAM			HMDDM+2MB SRAM			HMDDM+4MB SRAM		
					MEM [KB]	TIME [ns]	MEM [KB]	TIME [ns]	APL	MEM [KB]	TIME [ns]	APL	MEM [KB]	TIME [ns]	APL	MEM [KB]	TIME [ns]
s5378	35	49	164	703.9	75	12030	329	7039	176.1	943	1761	150.9	2048	1509	100.5	3979	1005
s9234	36	39	211	590.7	149	13450	368	5907	352.8	928	3528	283.1	2048	2831	256.7	3474	2567
dsip	229	197	224	649.3	112	17500	479	6493	204.1	942	2041	109.0	1981	1090	83.0	3950	830
bigkey	263	197	224	831.7	150	19170	524	8317	156.1	918	1561	125.3	1979	1253	110.3	4089	1103
apex6	135	99		297.1	23	3700	101	2971	28.1	1022	281	24.9	2035	249	24.8	4089	248
cps	24	102		242.5	34	3468	95	2425	8.8	999	88	8.8	1990	88	8.3	3797	83
frg2	143	139		529.9	40	6390	175	5299	36.1	1002	361	33.7	1919	337	31.2	4093	312
						0.59	1.00	1.00		13.52	9.57		27.31	10.68		54.37	11.85

表 2　HMDDM の実装結果.

方式	LE 数	ピン数	最大動作周波数 [MHz]
標準 HMDDM	348	202	93.1
先読み HMDDM	239	234	110.1

アドレスを同時に読むため, 制御回路とデータパスを単純にできたからである. ただし, インデックスの先読みを行うため命令長が長くなり, メモリ量が増える. また, メモリとの接続ピン数が増加する.

5.2　他の手法との比較

MCNC ベンチマーク関数 [17] を実装し, インデックスを先読みする HMDDM と他の手法との比較を行った. 比較に用いたのは, QDDM [12], Intel 社の汎用プロセッサ Core2Duo (1.2GHz) である. なお, ベンチマーク関数は全て多出力関数なので, 単一出力関数に分割後, 決定グラフを作成し, 変数順序最適化 [13] を行って節点数を削減してから各手法で実現した.

c)　QDDM

ベンチマーク関数を QDD に変換し, 最適なコードを生成した [14]. QDDM では HMDDM と同じ FPGA を用いて実装した. また, メモリは FPGA の組込みメモリを使用した. QDDM は分岐命令に 2 クロック要する. QDDM は 200MHz で動作させたため, 実行時間は $APL \times 2 \times 5$ nsec である.

d)　Core2Duo

ベンチマーク関数からアルゴリズム 4.5 を用いて HMDDM を作成した. 設定したメモリ上界は 2 MBytes である. これは, Core2Duo の 2 次キャッシュサイズ (2 MBytes) を越えると, キャッシュミスが頻繁に発生し, 性能が極端に低下するからである. HMDD を C コードで模擬し, GCC コンパイラ (最適化オプション-O3 を指定) を用いて実行コードを生成した. 使用した OS は Windows XP SP2 である.

e)　先読み HMDDM

先読み HMDDM では, 外付け SRAM のサイズを 1 MBytes, 2 MBytes, 4 MBytes と変化させメモリ量に対する性能向上率を求めた. 先読み HMDDM では分岐命令に 1 クロック要する. 100MHz で動作させたため, 実行時間は $APL \times 10$ nsec である.

表 1 に他の手法との比較結果を示す. 表 1 において, *NAME* はベンチマーク関数名を; *IN* と *OUT* はベンチマーク関数の入出力数を; *APL* は平均パス長を; *MEM* は必要メモリ量 (単位 KBytes) を; *TIME* はテストベクトル 1 個の平均評価時間 (単位 nsec) を表す. 表 1 より, 先読み HMDDM は QDDM と比較して 9.57-11.85 倍高速であり, Core2Duo と比較して 16.22-20.08 倍高速であった. 特に, 大きなベンチマーク関数 (s5378) では, HMDDM のメモリ量を 2 倍, 4 倍にすることで, 評価時間を 11.7%〜41%短縮できた. 逆に, 小さなベンチマーク関数 (apex6, cps, frg2) ではメモリ量を増加させても評価時間はほとんど短縮しなかったことから, 十分なメモリを与えていると考えられる. 先読み HMDDM は QDDM に比べ, メモリを 10〜50 倍余分に使用している. 実験結果より, HMDDM では, メモリ量を十分使用すれば PC やホモジニアス決定グラフマシンと比較して, 高速に評価できることがわかった.

6. ま と め

先読み HMDDM について述べた. また, HMDDM のメモリを効率よく使用し, 平均実行時間を最小にするコードを生成する手法を述べた. FPGA と外付けメモリを用いて標準 HMDDM と先読み HMDDM を実装した結果, 先読み HMDDM では制御回路を単純化できたため, 標準 HMDDM に比べ 18.2%高速化することができた. また, MCNC ベンチマーク関数を用いて比較を行った結果, 先読み HMDDM は QDDM と比較して 9.57-11.85 倍高速であり, Core2Duo と比較して 16.22-20.08 倍高速であった.

提案手法は, 外付けメモリの大容量化による決定グラフマシンの高速化の検討であるが, 実装した HMDDM は比較的小規模なので, 複数台実装することでさらなる高速化が期待できる.

7. 謝　　辞

本研究は, 一部, 日本学術振興会・科学研究費補助金, および, 文部科学省・知的クラスター創成事業 (第二期) の補助金による.

文　　献

[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. Compt.*, Vol. 54, No. 9, pp. 1041-1053, Sep. 2005.
[4] Y. Iguchi, T. Sasao, and M. Matsuura, "Implementation of multiple-output functions using PROMDDs," *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," *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," *ISMVL2003*, May, 2003, pp.247-255.
[9] S. Nagayama and T. Sasao, "On the optimization of heterogeneous MDDs," *IEEE Transactions on CAD*, Vol.24, No.11, Nov., 2005, pp.1645-1659.
[10] H. Nakahara, T. Sasao, and M. Matsuura, "Packet classifier using a parallel branching program machine," *DSD2010*, Lille, France, Sept. 1-3, 2010, pp.745-752.
[11] H. Nakahara, T. Sasao and M. Matsuura, "A comparison of architectures for various decision diagram machines," *ISMVL2010*, Barcelona, Spain, May 26-28, 2010, pp.229-234.
[12] H. Nakahara, T. Sasao, M. Matsuura, and Y. Kawamura, "A parallel branching program machine for sequential circuits: Implementation and evaluation," *IEICE Transactions on Information and Systems*, Vol. E93-D, No.8, Aug. 2010, pp.2048-2058.
[13] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *ICCAD'93*, Nov., 1993, pp. 42-47.
[14] T. Sasao, H. Nakahara, M. Matsuura, Y. Kawamura, and J.T. Butler, "A quaternary decision diagram machine: Optimization of its code," *IEICE Transactions on Information and Systems*, Vol. E93-D, No.8, Aug. 2010, pp.2026-2035.
[15] T. Sasao and M. Fujita (ed.), *Representations of Discreate Functions*, Kluwer Academic Publishers, 1996.
[16] A. Thayse, M. Davio, and J. P. Deschamps, "Optimization of multi-valued decision algorithms," *ISMVL'79*, Rosemont, IL., May, 1979, pp.171-177.
[17] S. Yang, "Logic synthesis and optimization benchmark user guide version 3.0," *MCNC*, Jan. 1991.
[18] 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.