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.

ヘテロジニアス MDD for ECFN マシンの消費電力遅延時間積に関する一考察

中原　啓貴† 　笹尾　勤†† 　松浦　宗寛††

† 鹿児島大学 大学院 理工学研究科 電気電子工学専攻 〒 890–0065 鹿児島県鹿児島市郡元 1–21–40
†† 九州工業大学 大学院 情報工学府 情報創成工学専攻 〒 820–8502 福岡県飯塚市大字川津 680–4

あらまし　本論文ではヘテロジニアス MDD for ECFN (Heterogeneous Multi-valued Decision Diagram for Encoded Characteristic Function for Non-zero outputs) を模擬する HMDD for ECFN マシンの消費電力遅延時間積について考察する．まず, 多出力論理関数を表現する HMDD for ECFN について述べる．次に, HMDD for ECFN マシンのアーキテクチャについて述べる．そして, HMDD for ECFN マシンの遅延時間と消費電力を実験的に測定し, 消費電力遅延時間積を求める．Intel 社の Core i5 プロセッサ (2.4GHz 動作) と比較を行った結果, 遅延時間に関して, HMDD for ECFN マシンは 1.40-4.27 倍優れており, 消費電力遅延時間積に関して, HMDD for ECFN マシンは 15.1-46.6 倍優れていた．

On a Power-Delay Product for a Heterogeneous MDD for ECFN Machine

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

† Faculty of Engineering, Kagoshima University, 1-21-40, Korimoto, Kagoshima 890–0065, Japan
†† Department of Creative Informatics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan

Abstract　This paper analyzes a power-delay product for a HMDD for an ECFN (Heterogeneous Multi-valued Decision Diagram for Encoded Characteristic Function for Non-zero outputs) Machine that emulates the HMDD for ECFN. First, we introduce the HMDD for ECFN representing the multi-output logic function. Then, we show an architecture for the HMDD for ECFN machine. Next, we obtain the delay time and the power consumption for the HMDD for ECFN machine using the MCNC benchmark function. Finally, we analyze the power-delay product for the HMDD for ECFN machine. Compared with the Intel's Core i5, whose clock frequency is 2.4 GHz, as for the delay time, the HMDD for ECFN machine is 1.40-4.27 times shorter than Core i5, and as for the power-delay product, it is 15.1-46.6 times smaller.

1. はじめに

ディープサブミクロン時代以前のプロセッサでは LSI のプロセスを微細化することで, 消費電力を抑えつつ性能向上を達成してきた．しかし, プロセス微細化が 90nm に突入するとリーク電流による消費電力の増加が問題となり, マルチコア等の消費電力性能に優れたプロセッサが採用されている [5]．一方, CMOS プロセスの微細化の限界が議論されるようになり, 2025 年に 10nm に達した時点が微細化の限界を迎えると予測されている [8]．以上のことから, 将来のプロセッサは

1. 短期的視点では消費電力遅延時間積に優れること,
2. 長期的視点では CMOS 微細化に変わる方法を採用できること

が求められる．

決定グラフマシン [1], [9] とは決定グラフ (DD: Decision Diagram) を評価する専用プロセッサである．決定グラフマシンの応用として, 産業用シーケンサ [20], 論理シミュレーションアクセラレータ [6], パケット分類器が報告されている [14]．各節点の入力数を自由に設定できる決定グラフをヘテロジニアス MDD (Heterogeneous Multi-valued DD) という [11]．HMDD は各節点の入力数を適切に決めることで, 2 値の決定グラフである BDD (Binary Decision Diagram) と同じメモリ量で平均して約 2 倍高速に評価できる [15]．HMDD は各節点の入力数を増やすとメモリ量は増加するが, パス長を短縮でき, BDD と比較して遅延時間を短縮できる．つまり, HMDD マシンはメモリ量を増加することで遅延時間を短縮できる．メモリは人手で設計可能であるメモリセルを規則的に並べた構造であるため, CMOS 微細化問題を克服する方法として採用できる．例えば, カーボン・ナノチューブや光素子などの新素材や 3 次元構造などの新構造を採用したメモリが次世代メモリとして注目されている．よって, HMDD マシンは CMOS 微細化問題を克服しつつ性能向上を継続できる可能性が高い．

本論文では, 短期的な視点で検討すべき項目である消費電力遅延時間積について述べる．多出力論理関数に関して, 補助変数を用いる BDD (BDD for ECFN) [17] を HMDD に拡張した HMDD for ECFN マシンが面積遅延時間積に関して優れていることが知られている [13]．本論文では, HMDD for ECFN マシンと Intel 社の Core i5 プロセッサの遅延時間と消費電力を実際の回路を用いて測定し, 消費電力遅延時間積を求め, 推定値と比較を行い, 性能差を考察する．

第 2 章では本論文で用いる用語の定義を行い, 第 3 章では HMDD for ECFN マシンについて述べ, 第 4 章では消費電力遅延時間積の比較を行い, 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 [2].

[定義 2.4] [7] **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 の全て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.

[定義 2.6] 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)** [11] と呼ぶ.

2.3 平均パス長 (Average Path Length)

各節点の評価時間が全ての等しいと仮定すると, 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する. 従って, 評価時間は平均パス長に比例する.

[定義 2.7] 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 2.8] 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. $P(X_i=c)$ 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.

補助変数 z1 z0	評価する関数
0 0	f0
0 1	f1
1 -	f2

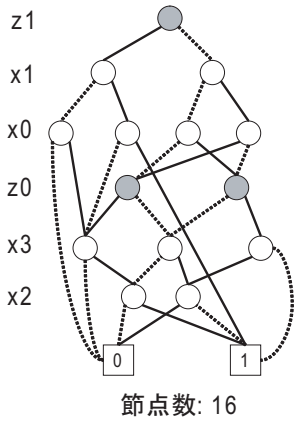


図 1 2 ビット加算器を表現する BDD for ECFN の例.

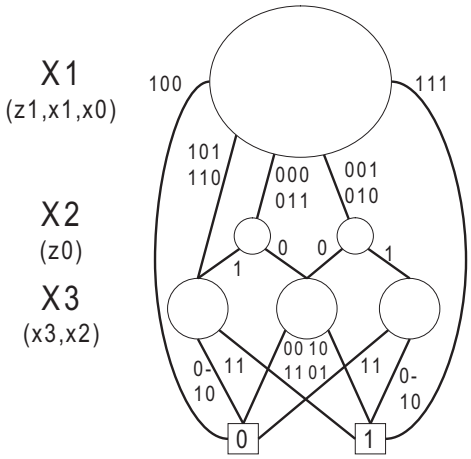


図 2 2 ビット加算器を表現する HMDD for ECFN の例.

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

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.

2.4 ヘテロジニアス MDD for ECFN

ECFN (Encoded Characteristic Function for Non-zero outputs) は写像 $F: B^n\times B^u \to B$ を表現する. ここで, $u=\lceil log_2 m\rceil$, $F(\vec{a},\vec{b})=1 \Leftrightarrow f_{\nu(\vec{b})}(\vec{a})=1$ であり, $\nu(\vec{b})$ は 2 進ベクトル $\vec{b}$ によって表現される整数である.

[例 2.1] 4 出力関数の ECFN の例を示す.

$$F = \bar{z_1}\bar{z_0}f_0 \vee \bar{z_1}z_0f_1 \vee z_1\bar{z_0}f_2 \vee z_1z_0f_3.$$ ■

ECFN [17] の定義を以下に示す.

[定義 2.10] $x^0=\bar{x}$, $x^1=x$.

[定義 2.11] m 出力関数 f_i(i=0, 1, ..., m-1) の ECFN (Encoded Characteristic Function for Non-zero outputs) は,

$$F = \bigvee_{i=0}^{m-1} z_{u-1}^{b_{u-1}} z_{u-2}^{b_{u-2}} \cdots z_0^{b_0} f_i, \qquad (1)$$

である. ここで $\vec{b}=(b_{u-1},b_{u-2},\ldots,b_0)$ は整数 i の 2 進表現であり, $u=\lceil log_2 m\rceil$ である.

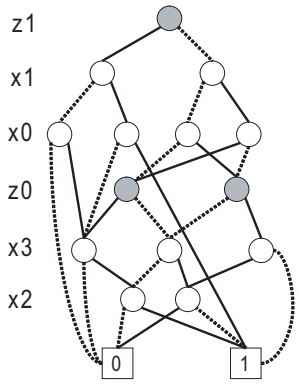
[例 2.2] 図 1 に 2 ビット加算器を表現する BDD for ECFN を示す. また, 図 2 に図 1 に示した BDD for ECFN を HMDD for ECFN に変換した例を示す. ■

HMDD for ECFN の評価は複雑である. HMDD for ECFN を評価するアルゴリズムを以下に示す.

[アルゴリズム 2.1] (HMDD for ECFN の評価) 入力を X, 出力数を m とし, 補助変数の値を AUX ($0 \leqq AUX \leqq \lceil \log_2 m\rceil - 1$) とする.

1. $AUX \leftarrow 0$ とする.

補助変数 z1 z0	評価する関数
0 0	f2
0 1	f1
1 -	f0



節点数: 14

図 4　図 1 の ECFN の符号化を最適化した結果.

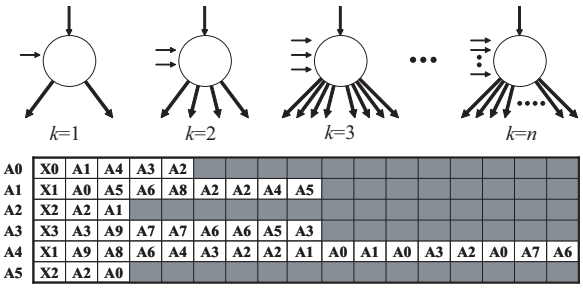


図 5　HMDD を直接分岐方式で実現した例.

2.　HMDD for ECFN の入力を $\{X, AUX\}$ とし, 出力値 f_{AUX} を得る.

3.　$AUX \leftarrow AUX + 1$ とする.

4.　$AUX = m$ ならば停止. そうでなければ Step 2. に戻る.

[例 2.3]　図 3 に 2 ビット加算器を表現する HMDD for ECFN の評価例を示す.　まず, f_0 を評価するため, 補助変数を $(z_1, z_0) = (0, 0)$ とする. HMDD for ECFN を評価し, $f_0 = 1$ を得る. 次に, f_1 を評価するため, 補助変数を $(z_1, z_0) = (0, 1)$ とする. HMDD for ECFN を評価し, $f_1 = 0$ を得る. 最後に, f_2 を評価するため, 補助変数を $(z_1, z_0) = (1, 0)$ とする. HMDD for ECFN を評価し, $f_2 = 1$ を得る. ■

HMDD for ECFN を評価するマシンは, HMDD を評価する回路に加えて, 補助変数をインクリメントする回路, 及び, m 回繰返し評価を行うための制御回路が必要である.

2.5　ECFN の補助変数の符号化

式 (1) において, $z_0, z_1, \ldots, z_{u-1}$ は出力を表現する補助変数である. 定義 (2.11) では, 整数 i を 2 進ベクトル $\vec{b}$ でそのまま符号化しているが, 符号化法を工夫することで表現を簡単化できる. 最適な補助変数の割当てを求めることは困難であることが知られており, 発見的な手法を用いて, 準最適な補助変数の割当てを求める手法が提案されている [18]. 本論文も, [18] の手法を用いて, 補助変数の割当てを行う.

[例 2.4]　図 1 の ECFN の符号化を最適化した結果を図 4 示す. 符号化を最適化することで, 変数順序が同一でも節点数を削減できる. ■

3.　ヘテロジニアス MDD for ECFN マシン

3.1　直接分岐方式と間接分岐方式 [15]

決定グラフマシンは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. 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り, メモリ使用効率が悪い.

[例 3.5]　図 5 は HMDD を直接分岐方式で実現した例である. 節点の入力数を k とすると, 分岐数が 2^k となる. k の値が異なると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 [15]. 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は 1 回でよいが, 率直な間接分岐方式ではメモリ参照を 2 回行う. ただし, メモリの利用効率は上がる.

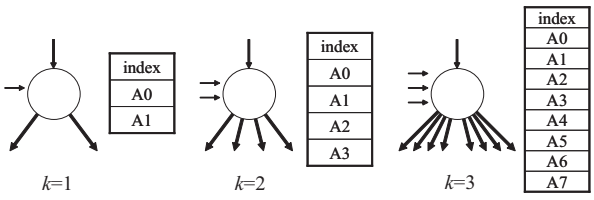


図 6　HMDD の非終端節点を評価する間接分岐方式.

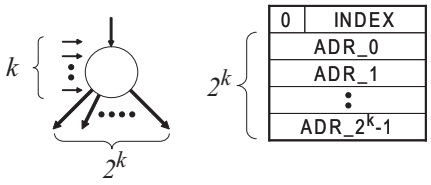


図 7　間接分岐命令.

図 8　単一出力・ジャンプ命令.

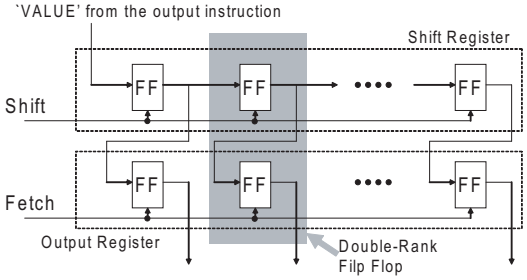


図 9　ダブルランク・シフトレジスタ.

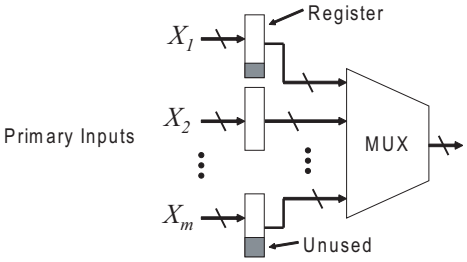


図 10　入力選択回路.

[例 3.6]　図 6 に HMDD の非終端節点を評価する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 6 からも明らかなように, 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. 本論文で提案する HMDD マシンは, 非終端節点を間接分岐方式で評価する.

3.2　HMDD for ECFN マシン

HMDD for ECFN の非終端節点は, 図 7 に示した間接分岐命令で, 終端節点は, 図 8 に示した単一出力・ジャンプ命令で模擬できる. 図 11 に **HMDD for ECFN マシン (HMDDM for ECFN)** を示す. 図 11 において, **命令メモリ**は命令を格納し, **命令レジスタ**は命令メモリから読み出した命令を格納す

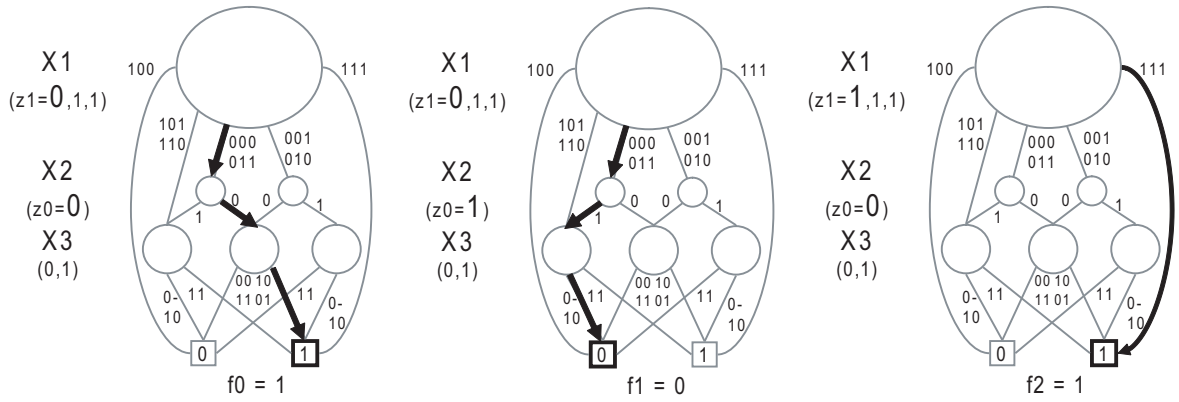


図 3　2 ビット加算器を表現する HMDD for ECFN の評価例.

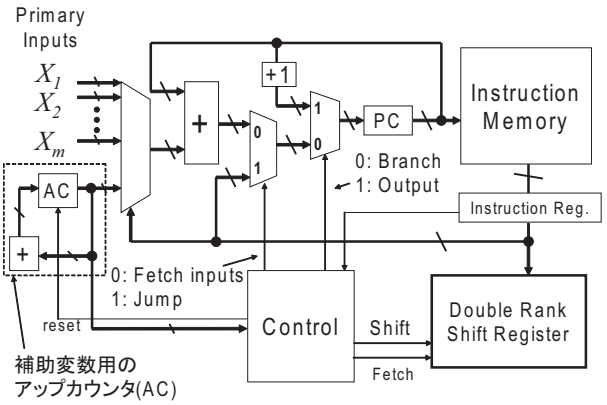


図 11　HMDD for ECFN マシン.

る. HMDD for ECFN を評価するには, 補助変数をインクリメントしながら評価するので, **補助変数カウンタ (AC: AUX variable Counter)** を用いる. 各超変数のサイズは自由に設定可能なので, 図 10 に示す**変数選択回路**を用いる. 出力値は図 9 に示す**ダブルランク・シフトレジスタ**で保持する. ダブルランク・シフトレジスタは, シフトレジスタと**出力レジスタ**から構成され, 各フリップ・フロップは**ダブルランク・フリップフロップ** [16] で構成される. 評価した出力をシフトしながら保持し, 全ての出力の評価が終ったときに, シフトレジスタの値を出力レジスタに転送する.

HMDD for ECFN マシンの分岐命令の実行を以下に示す.

［アルゴリズム 3.2］ (2^k 間接分岐命令の実行)

1. 分岐先アドレスを決定する.

1.1 PC で指定された番地からインデックスを読み出す.

1.2 読み出した値とインデックスの値を加算し, PC に格納する.

2. ジャンプを実行する.

2.1 PC で指定された番地からジャンプする番地を読み出す.

2.2 ジャンプする番地を PC にセットする.

HMDD for ECFN マシンの単一出力・ジャンプ命令の実行を以下に示す.

［アルゴリズム 3.3］ (単一出力・ジャンプ命令の実行) AC を補助変数カウンタの値, m を出力数とする.

1. リセット後, $AC \leftarrow 0$.

2. 出力を行う.

2.1 PC で指定された番地から出力値とジャンプする番地を読み出す.

2.2 出力値をダブルランク・シフトレジスタにセットする.

2.3 $AC \leftarrow AC + 1$.

2.4 全ての出力がセットされれば ($AC = m$), 評価中の出力を出力レジスタに転送し, $AC \leftarrow 0$ とする.

3. ジャンプを実行する.

3.1 PC にジャンプする番地をセットする.

外部入力数を n, 外部出力数を m, HMDD for ECFN マシンの非終端節点数を p, 節点 i のインデックスのサイズをを k_i とする. SO-HMDDsM と同様に, HMDD for ECFN マシンの命令メモリのアドレス数 a_{ECFN} は

$$a_{ECFN} = 2 + \sum_{i=1}^{p} (2^{k_i} + 1)$$

となる. ECFN は $n + \lceil log_2 m \rceil$ ビットの入力を持つので, 命令メモリのワード長 w_{ECFN} は

$$w_{ECFN} = max(\lceil log_2 a_{ECFN} \rceil + 2, \lceil log_2 (n + \lceil log_2 m \rceil) \rceil + 1)$$

である. よって, HMDD for ECFN マシンのメモリ量は

$$a_{ECFN} \times w_{ECFN} \qquad (2)$$

である.

4. 実 験 結 果

HMDD for ECFN マシンを Altera 社 Cyclone III スタータキット上に実装し, MCNC ベンチマーク関数 [19] を用いて消費電力遅延時間積を求め, Intel 社 Core i5 2520M (2.4GHz 動作) と比較を行った. 表 1 に遅延時間, 及び消費電力遅延時間積を示す. 表 1 において In はベンチマーク関数の入力数を; Out は出力数を; 節点数は HMDD for ECFN の節点数を; APL は HMDD for ECFN の APL を; 及び, $Ratio$ は $\frac{Core\ i5}{HMDD for ECFN\ マシン}$ の比率を表す. なお, HMDD for ECFN は [18] の手法を用いて補助変数の割当てを行った BDD for ECFN を生成し, メモリ量制約 3MBytes (注1) としてダイナミック・プログラミング [10] を用いて生成した.

4.1 遅延時間の実験結果

MCNC ベンチマーク関数を実現に対して 100 万個のランダムテストベクトルを評価する時間 ([nsec]) を測定し, **1 テストベクトル当りの評価時間を遅延時間 ([nsec/work])** とした. HMDD for ECFN マシンは Altera 社 Cyclone III スタータキットに外付け SRAM (4MB) を取り付けて 100 MHz で動作させた. 一方, Core i5 は Pasanosic 社 Let's note CF-S9 上でバイナリコードに変換した HMDD for ECFN を動作させた. コンパイラは gcc を用い, 最適化オプションは-O3 とした. 表 1 より, HMDD for ECFN マシンは Core i5 よりも 1.40-4.27 倍遅延時間が短かった.

4.2 遅延時間の考察

HMDD for ECFN マシンと Core i5 の遅延時間を推定し, 遅延時間の差を考察する.

HMDD for ECFN マシンの遅延時間を必要サイクル数で推定する. APL を HMDD for ECFN の APL; m を出力数; $C_{Machine}$ を HMDD for ECFN マシンが 1 節点を評価するの

(注1)：比較に用いた Core i5 (2520M) のスマートキャッシュのサイズ (3MByte) と同じサイズである.

表 1　HMDD for ECFN マシンと Intel 社 Core i5 の比較.

Name	In	Out	節点数	APL	遅延時間 [nsec/work]			消費電力遅延時間積 [W·nsec/work]		
					Core i5 2.4GHz	HMDD マシン 100MHz	Ratio	Core i5 2.4GHz	HMDD マシン 100MHz	Ratio
alu4	14	8	141	1.57	717	251	2.85	7275.2	236.1	30.8
apex2	39	3	260	4.42	593	265	2.24	6017.0	249.3	24.1
cc	21	26	27	2.16	2871	1123	2.56	29131.1	1055.8	27.5
lal	26	19	60	2.58	3027	980	3.09	30714.0	921.6	33.3
pcler8	27	17	30	2.10	2714	714	3.80	27538.1	671.2	41.0
spla	24	21	170	1.50	2355	630	3.74	23895.4	592.2	40.3
ttt2	16	46	51	2.25	6801	2070	3.29	69007.6	1945.9	35.4
ts10	22	16	31	1.30	1775	416	4.27	18010.4	391.1	46.6
C1355	41	32	10296	8.16	7316	5222	1.40	74233.2	4909.2	15.1
C1908	33	25	3299	5.06	4742	2530	1.87	48115.6	2378.3	20.2
C3540	50	22	17802	5.70	4801	2511	1.91	48714.2	2360.5	20.6

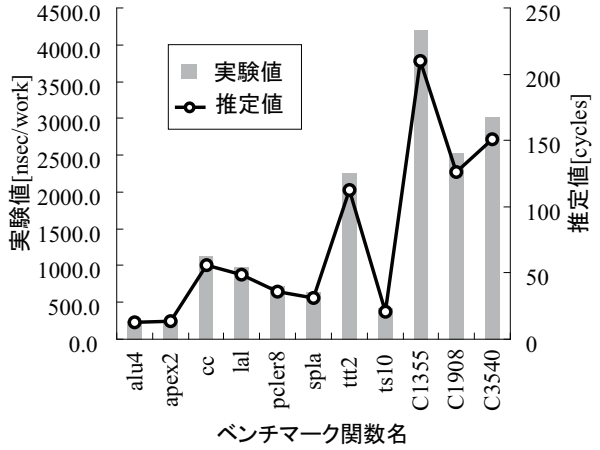


図 12　HMDD for ECFN マシンの遅延時間の実験値と推定値.

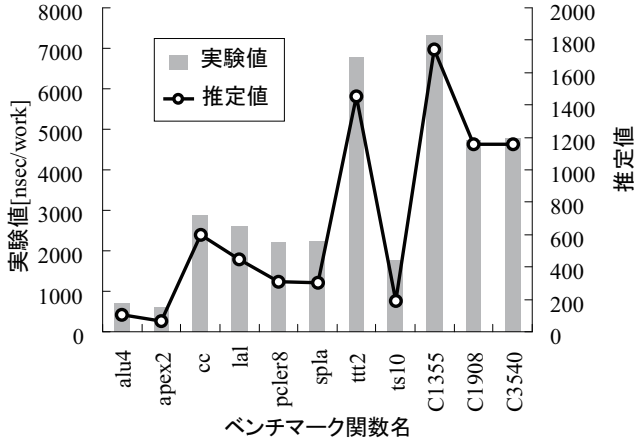


図 14　Core i5 の遅延時間の実験値と推定値.

に必要なサイクル数とすると, HMDD for ECFN マシンのサイクル数の推定値 $Cycle_{Machine}$ ([cycles]) は

$$Cycle_{Machine} = APL \times m \times C_{Machine} \quad (3)$$

と推定できる. 実装より, 先読み HMDD for ECFN マシンは $C_{Machine} = 2$ であった [12]. 図 12 に HMDD for ECFN マシンの実験値と推定値を示す. 図 12 より, HMDD for ECFN マシンの遅延時間を精度良く推定できていることがわかる.

次に, Core i5 の遅延時間を必要サイクル数で推定する. 図 13 に x86 命令を用いた HMDD for ECFN の 1 節点を模擬する C コードとアセンブリ (ASM) コードを示す. 図 13 より, HMDD の 1 節点を模擬するためには入力値読出し (mov, sal, and 命令) を行い, 読み出した値に応じて間接分岐 ($test, jmp$ 命令) を行う. ここで, 命令とデータは十分大きなキャッシュに格納されており, 常にヒットすると仮定する. 仮定を成立させるため, 本実験でもキャッシュサイズ (3MBytes) に納まるように HMDD for ECFN を設計した. 従って, 入力値読出しは一定時間で実行可能である.

```
// C-code                          // Assembly (ASM)-code
switch((in[x] >> y)& z){           mov in+x, %eax
case 1:  goto N1;                  sal $y, %eax
case 2:  goto N2;                  and $z, %eax
case 3:  goto N3;                  test $-858993460, %eax
                                   jmp *Label (,%eax,4)
⋮                                  Label:
                                   .long N1
                                   .long N2
case 2^(k-1):  goto N_(2^(k-1))    ⋮
}                                  .long N_(2^(k-1))
```

図 13　HMDD for ECFN の 1 節点を模擬する C コードと ASM コード.

現在のプロセッサは分岐によるパイプラインストールを回避するため, 分岐先アドレス予測を行う. **分岐先アドレスバッファ (BTB: Branch Target Buffer)** と呼ばれるキャッシュに分岐先アドレスを格納しておき, BTB にヒットした場合, BTB から分岐先アドレスを読み出して分岐先を高速に確定する. 従って, 常に特定のアドレスにジャンプする分岐は BTB にヒットしやすく, 高速に処理できる. 一方, 不特定のアドレスにジャンプする分岐は BTB にヒットしにくく, 処理速度が低下する. アルゴリズム 2.1 より, HMDD for ECFN を評価するには補助変数をインクリメントしながら HMDD for ECFN を m 回評価するため, 補助変数を含む節点は常に分岐先アドレスが大きく変わり, BTB にミスするため, 分岐先アドレス予測に失敗すると考えられる. 一方, 補助変数を含まない節点 (入力変数だけの節点) は補助変数の値が変化しても分岐先アドレスは一定の範囲内であり, BTB にヒットするため, 分岐先アドレス予測に成功すると考えられる. n を入力数, m を出力数, $u = \lceil log_2 m \rceil$ を補助変数の個数とする. プロセッサが十分な大きさの BTB を持つと仮定すると, **分岐先アドレス予測ミス確率** P_{miss} は

$$P_{miss} = \frac{u}{n+u}$$

となる. Core i5 が HMDD for ECFN を評価するのに必要なサイクル数を $Cycle_{MPU}$ とすると

$$Cycle_{MPU} = APL \times Y \times (C_{hit}(1 - P_{miss}) + C_{miss}P_{miss})$$

となる. ここで, C_{hit} は分岐先アドレス予測成功時の間接分岐に必要なサイクル数であり, C_{miss} は分岐先アドレス予測失敗時の間接分岐に必要なサイクル数である. 間接分岐に必要なサイクル数を実験的に求め, $C_{hit} = 4$, $C_{miss} = 16$ とし, 表 1 に示した実験値と比較した結果を図 14 に示す. 図 14 よりほぼ遅延時間を推定できている. 実験値と異なる理由は, キャッシュに全てヒットすると仮定して推定値を求めているからである. Core i5 は 1 次, 2 次, 3 次と 3 段のキャッシュを備えているためキャッシュに関する正確なモデルを考慮する必要がある.

HMDD for ECFN マシンと Core i5 の推定値から, 遅延時間の差を考察する. 式 (3) から HMDD for ECFN マシンは専用

命令と専用アーキテクチャを用いることで一定時間で間接分岐を実行できる. 一方, 式 (4) から Core i5 は BTB のキャッシュヒットに依存し, ミスミット時のペナルティが低いアーキテクチャである. 資源に制約の無い理想的な BTB を用いたとしても, SPEC95 ベンチマークにおける予測精度は 64 最も頻繁に間接分岐が実行される割合は 47 命令に 1 度である [4]. 図 13 が示すように, HMDD for ECFN を模擬する場合, 5 命令に 1 度と高頻度で間接分岐命令が実行されるため, BTB にヒットする確率は低く, 間接分岐命令の実行は遅い. 以上のことから, 従来のプロセッサのモデルであるパイプラインを用いた高クロック動作を想定したアーキテクチャは HMDD for ECFN の評価に適さず, 遅延時間に差が出た原因であると考えられる.

本論文では, Core i5 のキャッシュに納まるように HMDD for ECFN を設計したが, キャッシュのサイズを超える場合はキャッシュミスも頻発し, 性能が低下してしまう. 一方, HMDD for ECFN マシンはキャッシュが存在せず, メモリ量を増加させても遅延時間は APL と出力数のみに依存する. 従って, HMDD for ECFN マシンはメモリ量をさらに増加させて遅延時間を短縮可能であるのに対して, 従来のプロセッサはメモリ量を増加させたとしても遅延時間を削減できるとは限らない.

4.3 消費電力の実験結果

測定対象のボードと電源の間に抵抗を挿入して動作時の電源電圧 ([V]) と消費電流 ([A]) を測定し消費電力 ([W]) を実験的に求めた. HMDD for ECFN マシンの消費電力は Altera 社 Cyclone III スタータキットと外付け SRAM ボードの合計の消費電力とした. 測定から, 電源電圧が 9.017 [V] であり消費電流が 0.104 [A] であったため, 消費電力を 0.937 [W] とした. 一方, Core i5 の消費電力は Panasonic 社 Let's note CF-9 の消費電力とした. なお, 測定時にはバックライトを切り, 不要なタスクは全て停止させ, ネットワークは切断した. 測定から, 電源電圧が 16.201 [V] であり消費電流が 0.104 [A] であったため, 消費電力を 10.141 [W] とした. 従って, HMDD for ECFN マシンの消費電力 Core i5 の約 10 分の 1 といえる.

4.4 消費電力の考察

消費電力はスイッチングしたトランジスタ数に比例する動的消費電力とリークによるトランジスタ数に比例する静的消費電力の和で表される.

まず, 動的消費電力に関して考察する. HMDD for ECFN マシンに関してはアーキテクチャが単純であることから動作部分はわずかである. さらに, 動作クロックが 100 MHz と低速であるため, 動的消費電力が小さいと考えられる. 一方, Core i5 に関してはアーキテクチャが複雑であり動作クロックが 2.4 GHz と高速であるため, 多くのトランジスタが高速にスイッチングすると想定され, 動的消費電力が大きいと考えられる.

次に, 静的消費電力に関して考察する. 実験では, HMDD for ECFN マシンと Core i5 がともに 3MB のメモリ (キャッシュ) に HMDD for ECFN を格納したので, メモリ部の静的消費電力はほぼ同一と仮定する. コアのトランジスタ数に関しては, Core i5 は複雑な命令を多数実行させるため HMDD for ECFN マシンよりも圧倒的に多い. 従って, 静的消費電力も Core i5 が大きいと考えられる.

以上の考察から, Core i5 が HMDD for ECFN マシンよりも消費電力が 10 倍も大きかったのは Core i5 が大量のトランジスタを高速に動作させて処理するのに対し, HMDD for ECFN マシンは少数のトランジスタを低速に動作させて処理するからであると考えられる.

4.5 消費電力遅延時間積

表 1 より, HMDD for ECFN マシンは Core i5 よりも 15.1-46.6 倍消費電力遅延時間積が優れていた. 従って, 決定グラフの模擬に関しては HMDD for ECFN マシンは Core i5 よりも消費電力が優れ遅延時間が短くなることがわかった.

5. ま と め

本論文では, 多出力論理関数を表現する HMDD for ECFN を模擬する HMDD for ECFN マシンの消費電力遅延時間積を実際の回路を用いて求め, 推定値と比較を行った. Intel 社の Core i5 プロセッサ (2.4GHz 動作) と比較を行った結果, 遅延時間に関して, HMDD for ECFN マシンは 1.40-4.27 倍優れており, 消費電力遅延時間積に関して, HMDD for ECFN マシンは 15.1-46.6 倍優れていた.

HMDD for ECFN マシンが Core i5 よりも消費電力遅延時間積が優れていた理由は, Core i5 が苦手とする間接分岐命令が頻繁に実行される決定グラフの模擬を ECFN マシンでは専用命令とアーキテクチャを用いて, 低クロックで実行したからである.

今後の課題の 1 つは, 一般の関数を効率よく表現する決定グラフの考案である.

6. 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] K. Driesen, U. Hölzle, "Accurate Indirect Branch Prediction," *Proc. of the 25th Int'l Symp. on Computer Architecture (ISCA98)*, 1998, pp. 167-178.

[5] P. P. Gelsinger, "Microprocessors for the New Millennium: Challenges, Opportunities, and New Frontiers," *ISSCC'01*, pp. 22-25.

[6] 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.

[7] 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.

[8] T. Kuroda, "Panel Discussion: The Semiconductor Industry in 2025," *ISSCC'10*, Feb., 2010.

[9] 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.

[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] 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.

[12] H. Nakahara, T. Sasao, and M. Matsuura, "On a prefetching heterogeneous MDD machine," *54th IEEE Int'l Midwest Symposium on Circuits and Systems (MWSCAS2011)*, Korea, August 7-10, 2011.

[13] H. Nakahara, T. Sasao, and M. Matsuura, "A Comparison of heterogeneous multi-valued decision diagram machines for multiple-output logic functions," *41st Int'l Symp. on Multiple-Valued Logic (ISMVL2011)*, Tuusula, Finland, May 23-25, 2011, pp.125-130.

[14] H. Nakahara, T. Sasao, and M. Matsuura, "Packet classifier using a parallel branching program machine," *13th EUROMICRO Conference on Digital System Design (DSD-2010)*, Lille, France, Sept., 2010, pp.745-752.

[15] H. Nakahara, T. Sasao and M. Matsuura, "A comparison of architectures for various decision diagram machines," *40th Int'l Symp. on Multiple-Valued Logic (ISMVL2010)*, Barcelona, Spain, May, 26-28, 2010, pp.229-234.

[16] T. Sasao, H. Nakahara, M. Matsuura and Y. Iguchi, "Realization of sequential circuits by look-up table ring," *The 2004 IEEE Int'l Midwest Symp. on Circuits and Systems (MWSCAS 2004)*, Hiroshima, July 25-28, 2004, pp.I:517-I:520.

[17] T. Sasao, M. Matsuura, Y. Iguchi, and S. Nagayama, "Compact BDD representations for multiple-output functions and their applications to embedded system," *IFIP VLSI-SOC'01*, Montpellier, France, December 3-5, 2001, pp. 406-411.

[18] T. Sasao, "Compact SOP representations for multiple-output functions: An encoding method using multiple-valued logic," *31th Int'l Symp. on Multiple-Valued Logic (ISMVL2001)*, Warsaw, Poland, May 22-24, 2001, pp.207-212.

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

[20] 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.