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.

並列ブランチング・プログラム・マシンを用いたパケット分類器について

中原　啓貴† 　笹尾　勤† 　松浦　宗寛† 　川村　嘉郁††

† 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
†† 　ルネサステクノロジ 〒 100–0004 東京都千代田区大手町 2-6-2

あらまし　ブランチング・プログラム・マシン (BM) とは 2 種類の命令 (分岐命令と出力命令) を有するプロセッサであり, 汎用プロセッサよりもアーキテクチャが単純である. BM は条件分岐を専用命令で実行するので, 特定のアプリケーションに対して汎用プロセッサよりも高速に処理できる. 本論文では, パケット分類器を並列ブランチング・プログラム・マシン (PBM) を用いて実現する. パケット分類器のルールの集合を分割し, 複数の BM で並列に処理する. ルール数が与えられたとき, 所定の性能を出すために必要な BM の台数を推定できることを示す. BM を 32 台用いた PBM32 を FPGA 上に実装し, Intel 社の Core2Duo@1.2GHz と比較を行った結果, PBM32 は最長実行時間に関して Core2Duo よりも 8.1〜11.1 倍高速であり, メモリ量に関して 9.7〜436.2 分の 1 であった.

A Packet Classifier Using a Parallel Branching Program Machine

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

† Department of Computer Science and Electronics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan
†† Renesas Technology Corp., Tokyo, 100-0004, Japan

Abstract　A branching program machine (BM) is a special-purpose processor that uses only two kinds of instructions: Branch and output instructions. Thus, the architecture for the BM is much simpler than that for a general-purpose microprocessor (MPU). Since the BM uses the dedicated instructions for a special-purpose application, it is faster than the MPU. This paper presents a packet classifier using a parallel BMs (PBM). To reduce computation time and code size, first, a set of rules for packet classifier is partitioned into subsets. Then, the PBM evaluates them in parallel. Also, the paper shows a method to estimate the necessary number of BMs to realize a given packet classifier. We implemented the PBM32, a system using 32 BMs, on an FPGA, and compared it with the Intel's Core2Duo@1.2GHz microprocessor. The PBM32 is 8.1-11.1 times faster than the Core2Duo, and the PBM32 requries only 0.2-10.3 percent of the memory for the Core2Duo.

1. はじめに

パケット分類 [18] は, ルータやファイアウォールが持つ機能の一つである. パケットのヘッダには, プロトコル番号, 送信元アドレス, 送信先アドレスやポート番号などの情報が含まれている. パケット分類器は, これらを参照し, ルールと呼ばれる予め登録されてたヘッダと一致した場合, 特定の動作を行う. パケット分類器は様々な用途で用いられており, 例えばセキュリティを改善するため用いられているファイアウォール (FW:Fire Wall), QoS 等で用いられるアクセス制御リスト (ACL:Access Control List), IP マスカレード等で行われるパケットのフィルタリング, 及び加工で用いられる IP チェーン (IPC: IP chain) が挙げられる.

パケット分類器は適用するネットワークの規模に応じて様々な実装が行われており, SOHO・中小企業向けにはネットワーク用途に特化した組込みマイコンが, 大企業向けには PC やサーバ用の汎用プロセッサが, データセンタや通信事業者向けには FPGA や ASIC が用いられている. 現在, 汎用プロセッサを用いたパケット分類器の処理速度は数百 Mbps 程度であり [4], ネットワーク速度の加速度的な発展に追いつかない.

本論文では並列ブランチング・プログラム・マシンを用いたパケット分類器について述べる. ブランチング・プログラム・マシン (以下 BM と略記) [1], [2], [20] は 2 種類の命令 (分岐命令と出力命令) を有するプロセッサである. BM は命令数が少数のため, 汎用プロセッサよりもアーキテクチャが単純である. また, 条件分岐を専用命令で実行するので, 条件分岐を多用するパケット分類器では汎用プロセッサよりも高速に処理できる.

第 2 章ではパケット分類器の定義を行い, 第 3 章では PBM の構成について述べ, 第 4 章では PBM を用いたパケット分類器の実現について述べ, 第 5 章では汎用 MPU との比較結果を示し, 第 6 章で本論文のまとめを行う.

2. パケット分類

2.1 用語定義

まず, パケット分類に関する用語定義を行う. **パケット分類表**は**ルール**の集合からなる. 各ルールは 6 組の**フィールド**で構成された入力と各入力に対するアクションを示すルールの番号 (値) からなる. 6 組のフィールドは送信先アドレス (SA), 送信元アドレス (DA), 送信先ポート番号 (SP), 送信元ポート番号 (DP), プロトコル番号 (PRT), 及びフラグ (FLG) を示す(注1). 各フィールドの要素は**エントリ**と呼ばれる. **パケット分**

(注1)：実際のパケット分類表にはパケットの向きを表すフローフィールドも存在する. 本論文では ClassBench [19] を用いたため, 扱わない.

表 1　パケット分類表の例.

in						out
SA	DA	SP	DP	PRT	FLG	Rule
1000	110*	[0:1]	[8:9]	ICMP	*	1
00**	1***	[3:8]	[6:8]	TCP	1111	2
010*	0010	[3:11]	[7:14]	UDP	0101	3
0***	10**	[8:9]	[4:11]	TCP	*	4
****	****	[0:15]	[0:15]	*	*	0

類器はパケットのヘッダから各フィールドに対応するデータを抽出し, パケット分類表を参照してマッチするルールを決定する. 送信元アドレスと送信先アドレスに関しては**最長プレフィックスマッチ**を行い, 送信元ポート番号と送信先ポート番号に関しては**範囲マッチ**を行い, プロトコル番号とフラグは**厳密マッチ**を行う. パケット分類表のどのルールにもマッチしない場合, **デフォルトルール**にマッチする. 本論文では, デフォルトルールの番号を 0 とする. 複数のルールにマッチする場合, ルール番号が最も小さいルール番号を生成する. ただし, デフォルトルールは除く.

[例 2.1]　表 1 にパケット分類表の例を示す. 表 1 において, $*$ はドントケアを表す. 論文のスペースの都合上, 各フィールドのサイズを 4 ビットとし, デフォルトルールの番号は 0 とした. (例終)

[例 2.2]　パケットのヘッダが $SA = 0000, DA = 1010, SP = 8, DP = 8, PRT = TCP, FLG = 1111$ のとき, ルール 2 とルール 4 とデフォルトルールにマッチするが, 優先度に従って, ルール 2 の番号が生成される. (例終)

本論文では IPv4 向けパケット分類器を考えるので, 送信元アドレス, 送信先アドレスは 32 ビット, 送信元ポート番号, 送信先ポート番号, フラグは 16 ビット, プロトコル番号は 8 ビットとする.

2.2　区間関数を用いたエントリの表現

パケット分類表の各フィールドは**区間関数**で表現できる. まず, 区間関数の定義を行う.

[定義 2.1][16] $x_i \in \{0,1\}$, $X = (x_1, x_2, \ldots, x_n)$, $Y = \sum_{i=0}^{n-1} x_{i+1} 2^i$ とする. **区間関数**を以下に定義する.

$$IN(X : A, B) = \begin{cases} 1 & (A \leqq Y \leqq B) \\ 0 & (otherwise) \end{cases} \quad (1)$$

ただし, A, B は $0 \leqq A \leqq B \leqq 2^n - 1$ を満たす整数とする.

次に, 区間関数を用いてパケット分類表の各フィールドのエントリを表現する. パケットのヘッダを 6 つの組 $(X_{SA}, X_{DA}, X_{SP}, X_{DP}, X_{PRT}, X_{FLG})$ で表現する.

ポート番号は範囲マッチなので区間関数で表現できる. プロトコル番号のエントリはプロトコル番号を b とすると

$$IN(X_{PRT} : b, b) \quad (2)$$

と表現できる. フラグも同様に定義できる. 送信元アドレスのエントリは $x_i \in \{0,1\}$, $y_i = *$, $v = (x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m)$, $A = \sum_{i=1}^{n} x_i 2^{i-1}$ とすると

$$IN(X_{SA} : A2^m, (A+1)2^m - 1) \quad (3)$$

と表現できる. 送信先アドレスも同様に定義できる.

パケット分類器では同時に複数のルールがマッチする場合がある. 複数のマッチを区別するために, **ベクトル化パケット分類関数**を定義する.

[定義 2.2]　フィールド数を k, ルール数を r とする. ベクトル化パケット分類関数 $\vec{F}$ は

$$\vec{F} = \bigvee_{i=1}^{r} \vec{e}_i \bigwedge_{j=1}^{k} IN(X_j : A_{(i,j)}, B_{(i,j)}). \quad (4)$$

と表現できる. ただし, $\vec{e}_i$ は i 番目の要素のみ 1 であり, その他の要素は 0 となる要素数 r の単位ベクトルである.

[例 2.3]　表 2 に表 1 に示したパケット分類表のエントリを式 (1), (2), (3) で表現したものを示す. なお, プロトコル番号は, $TCP = 1$, $UDP = 2$, $ICMP = 3$ と割当てている. 表 2 において $\vec{e}_i$ は式 (4) のベクトルを表す. (例終)

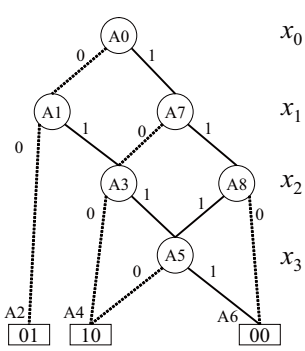

図 1　MTBDD の例.

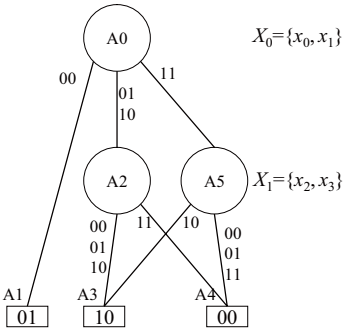

図 2　図 1 の MTBDD を MTQDD に変換した結果.

[例 2.4]　表 2 の各要素を式 (4) に代入すると, パケットのヘッダが $SA = 0000, DA = 1010, SP = 8, DP = 8, PRT = 1, FLG = 1111$ のとき $\vec{F} = (0, 1, 0, 1)$ となることがわかる. これはルール 2 と 4 にマッチしたことを表している. (例終)

3.　並列ブランチング・プログラム・マシン [13]

本論文では決定グラフを模擬するブランチング・プログラム・マシン (BM) を用いてパケット分類器を実現する. パケット分類表を決定グラフの一種である **MTQDD (Multi-Terminal Quaternary Decision Diagram)** で表現し, BM で模擬する. まず, MTQDD について述べる.

3.1　MTQDD

任意の n 入力論理関数は **BDD (Binary Decision Diagram)** [3] で表現できる. 多出力論理関数を表現する BDD は **MTBDD** [8] と呼ばれ, 高々n 回のテーブル参照で多出力を同時に評価できる. BDD の節点 v は**インデックス**, $index(v) \in \{0, 1, \ldots, n\}$ を持つ. 本論文では BDD の葉から根に向かって昇順にインデックスを割当てる. すなわち, $index(v) = 0$ となる節点 v は終端節点を表し, $index(v) = n$ となる節点 v は根節点を表す. パケット分類器では特定のパケットを大量に送信する DoS 攻撃や Broute Fource 攻撃が考えられるので, 本論文では, 最長評価時間を考える. BDD の最長評価時間は**最長パス長 (LPL: Longest path length)** に比例する [10]. 最長パス長の定義及び最適化手法は文献 [10] に述べられている.

[例 3.5]　図 1 に MTBDD の例を示す. $index(x_0) = 4, index(x_1) = 3, index(x_2) = 2, index(x_3) = 1$ である. (例終)

BDD の評価時間を削減するために, **MDD (Multi-valued Decision Diagram)** [9] を用いる. q 個の 2 値変数を組にして 1 つの変数と考えるとそれは 2^q 値の**超変数**を表す. 2^q 値の超変数で表現した決定グラフを $MDD(q)$ で表す. 従って, BDD は MDD(1) と等価である. 論理関数を MDD(q) で表現すると, BDD で表現した場合に比べメモリ参照回数をしばしば $\frac{1}{q}$ まで削減できる [7]. 従って, q を増加させると評価時間を短縮できる. しかし, 1 つの節点を表現するメモリ量は $O(2^q)$ で増加する. 多くのベンチマーク関数において, $q = 2$ の時 MDD(q) の総メモリ量が最小になる [11]. 従って, 論理関数の評価では MDD(2) のほうが BDD よりも適している. MDD(2) は 4 分岐を持つため, 本論文では MDD(2) を **QDD (Quaternary Decision Diagram)** と表現する. 各グループにおける 2 値変数の個数が全て q である MDD を**ホモジニアス MDD**(q) と呼ぶ. 一方, 各グループにおける 2 値変数の個数が q 以下である MDD を**ヘテロジニアス MDD**(q) と呼ぶ. ヘテロジニアス MDD では変数のグルーピングを工夫するとホモジニアス MDD よりも参照回数と節点数を削減できる [12]. 以降, 本論文ではヘテロジニアス MTQDD を単に QDD と呼ぶ.

[例 3.6]　図 2 に図 1 で示した MTBDD を MTQDD に変換した結果を示す. (例終)

3.2　BM の命令セット [17]

MTQDD の非終端節点を評価するため, **2 アドレス方式 2 分岐命令 (B_BRANCH)** と **3 アドレス方式 4 分岐命令 (Q_BRANCH)** を用いる. また, 終端節点を評価するため**データセット命令 (DATASET)** を用いる. 以下に, これらのニーモニックと内部表現を示す.

表 2　表 1 のパケット分類表のエントリを区間関数で表現した結果.

SA	DA	SP	DP	PRT	FLG	Rule	$\vec{e}_i$
$IN(X_{SA}:8,8)$	$IN(X_{DA}:12,13)$	$IN(X_{SP}:0,1)$	$IN(X_{DP}:8,9)$	$IN(X_{PRT}:3,3)$	$IN(X_{FLG}:0,15)$	1	1000
$IN(X_{SA}:0,3)$	$IN(X_{DA}:8,15)$	$IN(X_{SP}:3,8)$	$IN(X_{DP}:6,8)$	$IN(X_{PRT}:1,1)$	$IN(X_{FLG}:15,15)$	2	0100
$IN(X_{SA}:4,5)$	$IN(X_{DA}:2,2)$	$IN(X_{SP}:3,11)$	$IN(X_{DP}:7,14)$	$IN(X_{PRT}:2,2)$	$IN(X_{FLG}:5,5)$	3	0010
$IN(X_{SA}:0,7)$	$IN(X_{DA}:8,11)$	$IN(X_{SP}:8,9)$	$IN(X_{DP}:4,11)$	$IN(X_{PRT}:1,1)$	$IN(X_{FLG}:0,15)$	4	0001
$IN(X_{SA}:0,15)$	$IN(X_{DA}:0,15)$	$IN(X_{SP}:0,15)$	$IN(X_{DP}:0,15)$	$IN(X_{PRT}:0,15)$	$IN(X_{FLG}:0,15)$	0	0000

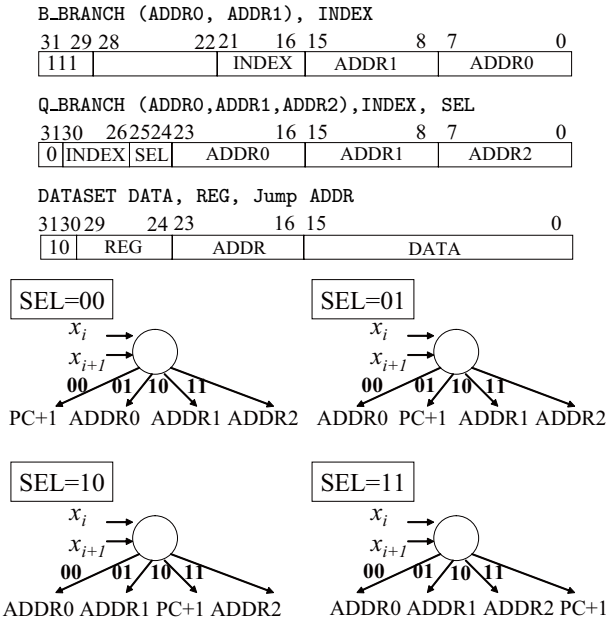

図 3　3 アドレス方式 4 分岐命令における分岐先アドレスの組合せ.

B_BRANCH 命令は INDEX で指定された変数の値が 0 であれば ADDR0 に指定されたアドレスにジャンプし, そうでなければ ADDR1 に指定されたアドレスにジャンプする. *Q_BRANCH* 命令は 4 つの分岐先アドレスのうち 1 つのアドレスにジャンプする. 4 つの分岐先アドレスのうち, 3 つは *ADDR0, ADDR1, ADDR2* で指定し, 残り 1 つの分岐先アドレスは現在の命令のアドレスの次のアドレス (PC+1) とする. 3 アドレス方式を採用した理由は, 現在の組込みプロセッサの標準的な語長 (32 ビット) に合わせるためである. 評価時間とステップ数削減のため, 4 種の *Q_BRANCH* 命令を用いる (図 3). *Q_BRANCH* 命令の SEL はそのうちの 1 つを指定する. i を INDEX で指定された変数の値とすると, SEL=i のとき, PC+1 にジャンプする. アドレスの割付けを工夫することでステップ数や実行時間を最適化できる [21]. 現在の命令のアドレスの次のアドレスが他の *Q_BRANCH* 命令の PC+1 と競合した場合, **無条件ジャンプ命令**が必要となる. 無条件ジャンプは *B_BRANCH* 命令を流用する. *DATASET* 命令は REG で指定されたレジスタに DATA (16 ビット) を書込む. そして ADDR で指定されたアドレスにジャンプする.

[例 3.7]　図 4 は図 2 に示した MTQDD に 3 アドレス方式 4 分岐命令のアドレス割当てを行った結果である. SEL は図 3 に示したものと同一である. A6 が無条件ジャンプ命令である. ここでは *B_BRANCH* 命令で代用している. QDD を模擬するコードを示す.

```
A0:  Q_BRANCH (A2,A2,A5),X0,00
A1:  DATASET 01,0,A0
A2:  Q_BRANCH (A3,A3,A4),X1,00
A3:  DATASET 10,0,A0
A4:  DATASET 00,0,A0
A5:  Q_BRANCH (A4,A4,A4),X1,10
A6:  B_BRANCH (A3,A3),--
```

(例終)

3.3　ブランチング・プログラム・マシン (BM)

QDD を模擬する BM を図 5 に示す. **命令メモリ**は語長が 32 ビットの BM 命令を 256 ステップ格納する. **命令デコーダ**は命令メモリから読み出した命令を解釈し, 実行を行う. **プログラムカウンタ (PC)** は現在のプログラムカウンタを保持する. **レジスタファイル**は出力レジスタや状態レジスタで構成される. レジスタの更新は *DATASET* 命令で行う. 出力及び状態レジスタは**ダブルランク・フリップフロップ** [14] と呼ばれる特殊なレジスタで構成する. 図 8 にダブルランク・フリップフロップを示す. L_1, L_2 は D ラッチである. *DATASET* 命令は 16 ビット毎にしか値を更新できないため, 16 ビットより大きな出力や状態変数を持つ回路を評価する場合, 前後の状態の値が混ざってしまう. そこで, ダブルランク・フリップフロップを用いて, 状態遷移信号 (S_Clock) が High のときに一度に状態レジスタの値を更新する. BM はレジスタの出力をフィードバックし順序回路を実現できる. **入力レジスタ**にはこれらのフィードバックされた値や外部入力や隣接した BM からの出力が入る. 試作した BM では 1 命令を処理するために 2 クロック使用し, 用途は以下の通りである.

第 1 クロック: 命令メモリから命令を読み出す
第 2 クロック: プログラムカウンタ (PC) に次のアドレスを取り込む

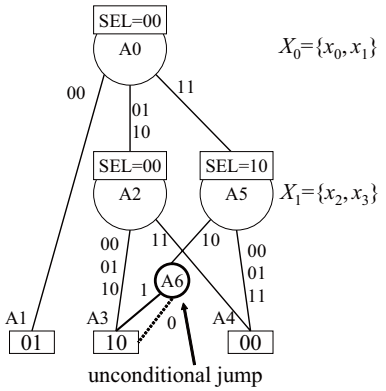

図 4　3 アドレス方式 4 分岐命令のアドレスを割当てた結果.

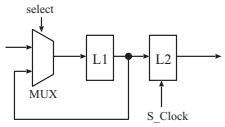

図 8　ダブルランク・フリップフロップ.

3.4　8_BM

大量の BM を直接相互に接続し通信可能とした場合, 接続回路は非常に大きくなり現実的ではない. 本論文で考案する**並列ブランチング・プログラム・マシン (PBM)** は階層的な構造を持つ. BM を 8 台並べた **8_BM** を 1 ユニットとし, それらを複数台並べ階層構造を構成する. **プログラマブル接続回路**を用いて他の 8_BM を接続する.

図 6 に 8_BM を示す. 8_BM は 8 台の BM から構成される. BM の出力値とフラグレジスタの値はカスケード接続された**プログラマブル・ルーティングボックス**を経由し, 隣接する BM のレジスタに格納される. レジスタに格納された値は, フィードバックされ各 BM の入力に送られる. また, 256 ビットの外部入力は各 BM にブロードキャストされる. 8_BM の出力をフィードバックして各 BM で参照することもできる. 従って, 各 8_BM を独立に動作させることもできる.

プログラマブル・ルーティングボックスはコンフィギュレーションを変えることで前段の BM の出力と現在の BM の出力のビットワイズ AND やビットワイズ OR 演算ができる. また, 定数出力もできる. ビットワイズ AND 演算を行う場合, 末端 (図 6 で灰色で示された部分) のプログラマブル・ルーティングボックスに定数 1 を出力させる. 一方, ビットワイズ OR 演算を行う場合, 定数 0 を出力させる.

3.5　並列ブランチング・プログラム・マシン

図 7 に 32 個の BM を用いた並列ブランチング・プログラム・マシン (PBM32) を示す. PBM32 は 4 ユニットの 8_BM とそれらを接続するプログラマブル接続回路から構成されている. 256 ビットの外部入力とコンフィギュレーション信号は各 8_BM にブロードキャストされる. コンフィギュレーション信

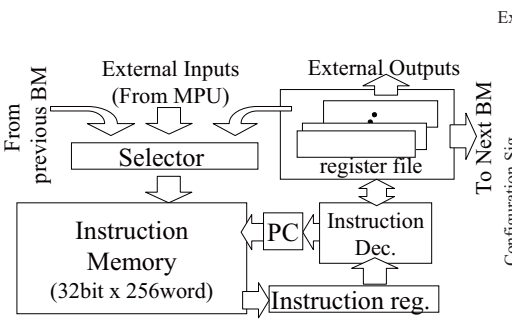

図 5　ブランチング・プログラム・マシン.

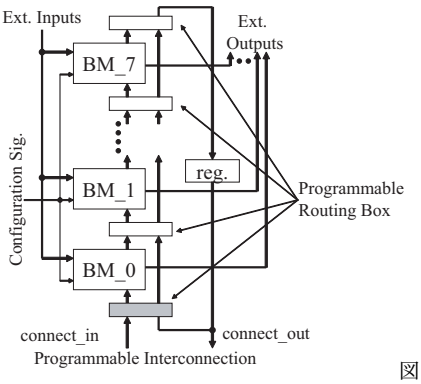

図 6　8_BM.

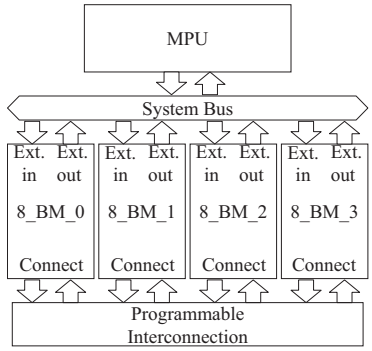

図 7　並列ブランチング・プログラム・マシン (PBM32).

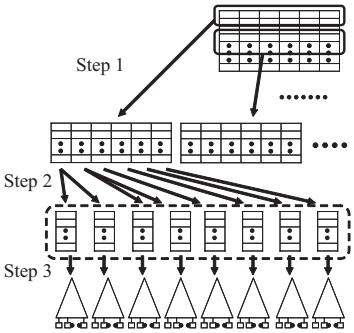

図 9　パケット分類表の分割.

号は, アドレス信号, 書込みイネーブル信号, プログラマブル接続回路の**接続指定信号**からなる. 各 8_BM の 64 ビットの出力は上位 32 ビットが外部に直接出力され, 下位 32 ビットはプログラマブル接続回路を通してフィードバックされ, 指定された 8_BM に接続される. また, フラグレジスタの値はビットワイズ AND またはビットワイズ OR され, 各 8_BM にフィードバックされる. これらはコンフィギュレーション信号で指定する. 全ての 8_BM の演算が終った後にプログラマブル接続回路を起動させるため, 各 BM のレジスタファイルの特定ビットの AND をとった**接続信号**をプログラマブル接続回路に転送する.

4. 並列ブランチング・プログラム・マシンを用いたパケット分類器の実現

4.1 パケット分類表の分割

パケット分類表を表現するベクトル化分類関数を直接決定グラフで表現した場合, 節点数が多くなり, BM に格納できない場合が多い. 本論文では, まず, ルールの集合を複数の**サブルールセット**に分割する (図 9 Step 1). 次に, サブルールセットをフィールド毎に分割して (図 9 Step 2)QDD で表現し 8_BM に格納する (図 9 Step 3). フィールド毎に分割することで並列処理による高速化が可能である.

［定理 4.1］ k をフィールドの個数, r をルール数とすると, ベクトル化パケット分類関数に関して関係

$$\vec{F} = \bigvee_{i=1}^{r} \vec{e}_i \bigwedge_{j=1}^{k} IN(X_j : A_{(i,j)}, B_{(i,j)})$$
$$= \bigwedge_{j=1}^{k} \bigvee_{i=1}^{r} \vec{e}_i IN(X_j : A_{(i,j)}, B_{(i,j)})$$

が成立する.

(証明) $f_{i,j} = \vec{e}_i IN(X_j : A_{(i,j)}, B_{(i,j)})$ とおくと,

$$\vec{F} = \bigwedge_{j=1}^{k} \bigvee_{i=1}^{r} \vec{e}_i IN(X_j : A_{(i,j)}, B_{(i,j)})$$
$$= (f_{1,1} \vee f_{2,1} \vee \cdots \vee f_{r,1}) \wedge (f_{1,2} \vee f_{2,2} \vee \cdots \vee f_{r,2}) \wedge \cdots \wedge (f_{1,k} \vee f_{2,k} \vee \cdots \vee f_{r,k})$$

となる. 上式を展開すると, 各和項から関数 f を 1 つづつ選択して組合せた積項の論理和となる. ここでベクトル化パケット分類関数の定義より, $\vec{e}_i$ は i 番目の要素のみ 1 の単位ベクトルである. 従って, 上式を展開しても $f_{\alpha,1} \wedge f_{\alpha,2} \wedge \cdots \wedge f_{\alpha,k}$ $(\alpha = 1, 2, \ldots, r)$ 以外の積項は全て 0 となる. よって上式を展開すると

$$= (f_{1,1} \wedge f_{1,2} \wedge \cdots \wedge f_{1,k}) \vee (f_{2,1} \wedge f_{2,2} \wedge \cdots \wedge f_{2,k}) \vee \cdots \vee (f_{r,1} \wedge f_{r,2} \wedge \cdots \wedge f_{r,k})$$
$$= \bigvee_{i=1}^{r} \vec{e}_i \bigwedge_{j=1}^{k} IN(X_j : A_{(i,j)}, B_{(i,j)})$$

を得る. (証明終)

定理 4.1 より, 表 2 に示したパケット分類表は

$$\vec{F} = \bigvee_{i=1}^{r} \vec{e}_i IN(X_{SA} : A_{SA_i}, B_{SA_i}) \cdot \bigvee_{i=1}^{r} \vec{e}_i IN(X_{DA} : A_{DA_i}, B_{DA_i}) \cdot \bigvee_{i=1}^{r} \vec{e}_i IN(X_{SP} : A_{SP_i}, B_{SP_i}) \cdot \bigvee_{i=1}^{r} \vec{e}_i IN(X_{DP} : A_{DP_i}, B_{DP_i}) \cdot \bigvee_{i=1}^{r} \vec{e}_i IN(X_{PRT} : A_{PRT_i}, B_{PRT_i}) \cdot \bigvee_{i=1}^{r} \vec{e}_i IN(X_{FLG} : A_{FLG_i}, B_{FLG_i}). \quad (5)$$

と表現できる. 式 (5) は 6 個の和項の論理積として表現されており, パケット分類表のフィールド数 (6 個) と一致する. 各和項を表現する関数を**ベクトル化フィールド関数**という. PBM を構成する 8_BM は 8 個の BM を持つので, 各ベクトル化フィールド関数をそれぞれ BM で実現した場合, 2 個の BM が未使用となり, 使用効率が悪い. そこで, 送信元アドレスと送信先アドレスを偶数ビットと奇数ビットで分割する. 送信元アドレスの偶数ビットを X_{SAE}, 奇数ビットを X_{SAO} とし, 同様に送信先アドレスの偶数ビットを X_{DAE}, 奇数ビットを X_{DAO} とすると, 式 (5) を

$$\vec{F} = \bigvee_{i=1}^{r} \vec{e}_i IN(X_{SAE} : A_{SAE_i}, B_{SAE_i}) \cdot \bigvee_{i=1}^{r} \vec{e}_i IN(X_{SAO} : A_{SAO_i}, B_{SAO_i}) \cdot \bigvee_{i=1}^{r} \vec{e}_i IN(X_{DAE} : A_{DAE_i}, B_{DAE_i}) \cdot \bigvee_{i=1}^{r} \vec{e}_i IN(X_{DAO} : A_{DAO_i}, B_{DAO_i}) \cdot \bigvee_{i=1}^{r} \vec{e}_i IN(X_{SP} : A_{SP_i}, B_{SP_i}) \cdot \bigvee_{i=1}^{r} \vec{e}_i IN(X_{DP} : A_{DP_i}, B_{DP_i}) \cdot \bigvee_{i=1}^{r} \vec{e}_i IN(X_{PRT} : A_{PRT_i}, B_{PRT_i}) \cdot \bigvee_{i=1}^{r} \vec{e}_i IN(X_{FLG} : A_{FLG_i}, B_{FLG_i}). \quad (6)$$

と変形できる. 送信先アドレスを偶数ビットと奇数ビットで分割するので, 区間関数の定数部もそれぞれ A_{SAE_i}, B_{SAE_i} と A_{SAO_i}, B_{SAO_i} に再定義される. 送信元アドレスに関する区間

表 3　プライオリティ・エンコーダ関数の例

input	output
1***	001
01**	010
001*	011
0001	100
0000	000

関数の定数部も同様に再定義される．式 (6) では 8 個のベクトル化フィールド関数の論理積として表現され, 8 個の BM を持つ 8_BM とビットワイズ AND で実現できる．ビットワイズ AND は 8_BM のプログラマブル・ルーティングボックスを用いる．

4.2　プライオリティ・エンコーダ関数

例 2.2 に示したように, パケット分類では複数のルールがマッチする場合がある．設計する回路は, 優先度が最も高いルールの番号を生成するものとする．式 (6) を直接 8_BM 上に実現した場合, 優先度を判定するプライオリティ・エンコーダが必要となる．r 個のルールの優先度を決定する関数を**プライオリティ・エンコーダ関数**と呼ぶ．

[定義 4.3]　プライオリティ・エンコーダ関数とは r ビットの入力に対して最上位のビットの位置を示す $\lceil log_2 r \rceil$ ビットの値を出力する関数である．

[例 4.8]　表 3 に $r = 4$ のときのプライオリティ・エンコーダ関数の例を示す．(例終)

[例 4.9]　例 2.4 のベクトル化パケット分類関数 $\vec{F} = (0, 1, 0, 1)$ に表 3 のプライオリティ・エンコーダ関数を適用すると $(0, 1, 0)$ を得る．表 2 のパケット分類表, 式 (4), 及び表 3 を用いることで優先度を考慮したパケット分類器が実現できる．(例終)

ルール数 r が数百〜千個の場合, プライオリティ・エンコーダ関数を BM で実現すると, ステップ数が多くなり実行時間が長くなる．多くの場合, ルールは人手で管理するが, ルールの衝突が発生しないようにルールの集合を設定するのは困難である．従って, ルール管理の容易さとプライオリティ・エンコーダ関数を実現する BM の実行時間のトレードオフを考慮し, 以下に示す方針でルールの競合を部分的に許容する．

仮定 1　サブルールセット内ではルール間の衝突は許容しない(注2)．

仮定 2　サブルールセット間ではルール間の衝突は許容する．

8_BM でサブルールセットを実現するため, 8_BM 内ではプライオリティ・エンコーダを使用しない．一方, サブルールセット間では衝突が発生する可能性があるため, 8_BM 間の優先度を決定するプライオリティ・エンコーダを BM で実現する．この場合, プライオリティ・エンコーダの入力数はサブルールセットの個数となる．サブルールセットは高々数個であるため, プライオリティ・エンコーダ関数を実現する BM は処理のボトルネックにはならない．

4.3　ベクトル化フィールド関数の解析

ベクトル化フィールド関数を表現する QDD の節点数を推定することでパケット分類器を実現するために必要な BM の台数を推定できる．

[定義 4.4]　複数の区間関数を組合せて別の区間関数を生成する場合, 互いに重複しないように区間を分割したものを **disjoint な区間**という．

[例 4.10]　表 2 に示したパケット分類表を式 (6) に代入し, ベクトル化フィールド関数を求めた．図 10 に SA と DA の区間とベクトル化した出力を示す．(例終)

例 4.10 から明らかなように, 二つの区間を組合せた場合に, 全体での区間数が増えるのは二つの区間間に共通部分が存在し, かつ一方の区間が他の区間の一部のみを包含する場合である．図 10 では, 例えば DA の区間 [8:9] と [6:8] が該当する．一方, SA では一方の区間が他方を完全に包含するか又は共通部分を有しないかのいずれかであり, 区間数は増えない．以上の観察より, 区間数に関して以下の定理と系を得る．

[定理 4.2]　s 個の区間を組合わせた場合の全体での disjoint な区間数は高々$2s$ 個である．

(証明) 組合わせる区間の個数に関して数学的帰納法を用いる．$s = 1$ のとき, 区間数は 2 個である．今, 区間数が s 個のとき,

(注2)：自動化ツールを用いてルールを分割するので, ユーザはルール間の衝突を考慮する必要はない．

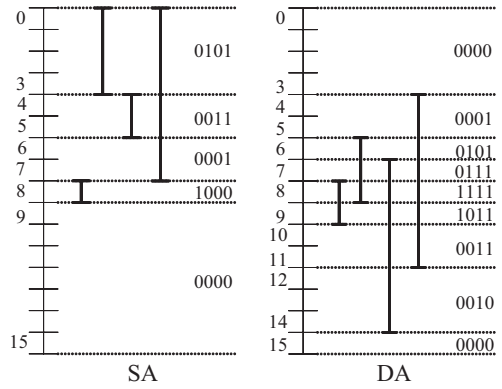

図 10　区間を組合わせた場合の disjoint な区間数.

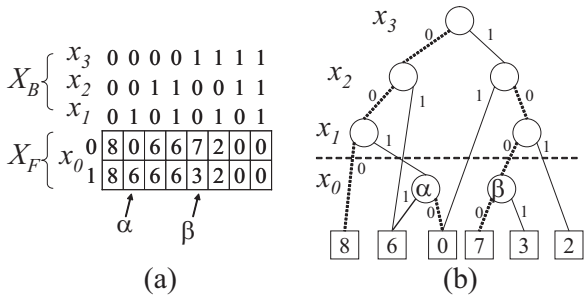

図 11　分解表の非定数パターンと MTBDD の節点数の関係.

定理が成立すると仮定する．区間を 1 つ追加した場合, 区間数は高々2 個しか増えない．従って, $(s+1)$ 個の区間に対して, 組合わせた場合の全体での区間数は $2(s+1)$ 個である．(証明終)

[例 4.11]　図 10 の DA では $s = 4$ 個の区間に対して, disjoint な区間数は定理 4.2 に示した上界である 8 個である．(例終)

[定理 4.3] [15] パケット分類表のフラグフィールド, 及びプロトコル番号フィールドを表すベクトル化フィールド関数が式 (2) で示した s 個の区間関数で構成されているとき, disjoint な区間数は高々$s + 1$ 個である．

[定理 4.4] [15] パケット分類表のアドレスフィールドを表すベクトル化フィールド関数が式 (3) で示した s 個の区間関数で構成されているとき, disjoint な区間数は高々$s + 1$ 個である．

あるインデックスにおける MTQDD の節点数の上界は, **分解表**の**列複雑度**から求まり, 列複雑度の上界はベクトル化フィールド関数の区間数から求まる．区間数に対する QDD の節点数の上界を導くため, 分解表を定義する．

[定義 4.5]　関数 $F(X) : B^n \to \{0, 1, \ldots k\}$, $B = \{0, 1\}$, $X = (x_1, x_2, \ldots, x_n)$ が与えられているものとする．$X = (X_B, X_F)$ を X の分割とする．F の**分解表**とは列のラベルに X_B を, 行のラベルに X_F を割り当てた表であり, 分解表の値は $F(X_B, X_F)$ の値に等しい．分解表の異なる列パターンの個数を**分解表の列複雑度**という．X_B を**束縛変数**, X_F を**自由変数**という．2 個以上の異なる値を持つ列パターンを**非定数パターン**といい, 全て同じ値を持つ列パターンを**定数パターン**という．

[例 4.12]　図 11(a) に図 10 に示した DA のベクトル化フィールド関数の分解表を示す．図 11(a) では $X_B = (x_3, x_2, x_1), X_F = (x_0)$ としている．分解表での関数値は図 10 で示したベクトルの 16 進数で表現している．図 11(a) において, 非定数パターンは X_B が 001, 100 となる列である．(例終)

非定数パターンは MTBDD の非終端節点が表す関数と一致し (図 11(b) の α, β が示す節点と非定数パターン), 定数パターンは MTBDD の終端節点が表す定数と一致する．従って, 関数 f における非定数パターンの上界を推定することで f を表現する MTBDD の節点数を推定できる．

[補題 4.1]　disjoint な区間数が t 個であるベクトル化フィールド関数の異なる非定数パターンの数は高々$t - 1$ 個である．

(証明) 束縛変数に上位ビットを, 自由変数に下位ビットを割当てた分解表を考える．非定数パターン数が最大となるのは, 各列パターン中に異なる値が 2 個存在する場合である．従って, t 個の区間数に対しては非定数パターンは高々$t - 1$ 個である．(証明終)

[定理 4.5]　入力数 $|X| = n$, ルール数 r であるベクトル化フィールド関数を表現する MTQDD の節点数は高々

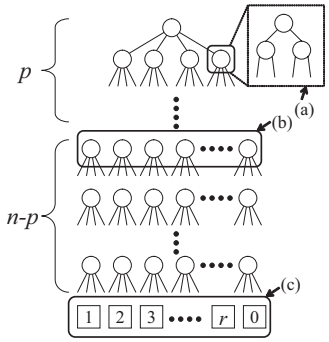

図 12　定理 4.5 の説明.

表 4　汎用 MPU との比較.

Rule	PBM32		Core2Duo		Ratio (C2D/PBM)	
	Time [nsec]	Mem [KB]	Time [nsec]	Mem [KB]	Time	Mem
acl1	98	8.9	945	86.6	9.6	9.7
acl2	98	7.8	945	127.7	9.6	16.3
acl3	98	10.1	801	143.9	8.1	14.2
acl4	98	10.0	801	138.3	8.1	13.7
acl5	98	7.9	945	107.2	9.6	13.5
fw1	98	3.0	1089	624.6	11.1	203.7
fw2	98	4.5	801	261.8	8.1	57.6
fw3	98	1.8	1089	708.6	11.1	379.5
fw4	98	2.6	945	538.5	9.6	202.7
fw5	98	2.5	1089	1104.1	11.1	436.2
ipc1	98	12.9	1089	142.6	11.1	11.0
ipc2	98	1.4	1089	67.5	11.1	46.6

$$\lceil\frac{2^p-1}{3}\rceil+\lceil\frac{n-p}{2}\rceil(2r-1)+(r+1)$$

である. ここで, p は関係 $r-1\leqq 2^p$ を満たす整数である.

(証明) 定理 4.2 と補題 4.1 より, MTQDD の各インデックスに対して非終端節点は高々$2r-1$ 個しか存在しない (図 12(b)). インデックス n (根) から列複雑度が $2r-1$ となるインデックス $n-p$ までの節点数は高々2^p-1 である. QDD は BDD の節点を 3 つまで纏めることができるので (図 12(a)), インデックス $n-p$ までの QDD の節点数は高々$\lceil\frac{2^p-1}{3}\rceil$ である. インデックスが $n-p$ から 0(終端節点) の間では非終端節点が高々$2r-1$ 個しか存在しない. MTQDD は 2 インデックスを同時に評価するので, インデックス $n-p$ 間では MTQDD の節点数は高々$\lceil\frac{n-p}{2}\rceil(2r-1)$ である. 終端節点数は $r+1$ 個である (図 12(c)). 従って, これらの節点数の総和が MTQDD の節点数の上界となる. (証明終)

ルール数 r が与えられたとき, ベクトル化フィールド関数を表現する MTQDD の節点数の上界が求まり, 必要な BM の台数が推定可能である.

5. 実 験 結 果

5.1 PBM32 の実装結果

PBM32 を Altera 社の FPGA に実装した. 実装に用いた評価ボードは Altera 社の Cyclone III LS 開発キットであり, 使用 FPGA は Cyclone III: EP3CLS200F780C7N (LE: 198464 個, M9k: 891 個) である. Quartus II (v.9.1) による実装では, LE を 23105 個, M9k を 32 個使用した. また, 最大動作周波数は 183.42 MHz であった. 組込みプロセッサは NiosII/f を用いた. 組込みプロセッサに必要な ALUT は 4225 個であった.

5.2 汎用 MPU との比較

ClassBench [19] を用いてパケット分類器を生成し, PBM32 上に実現した. ルール数を増加した場合, ルールの衝突が発生したり性能が劣化してしまうので [5], 組込みプロセッサや汎用の MPU を用いたパケット分類器のルール数は 100-300 個に設定されている [6]. 本実験ではルール数を 200 個に設定した. ClassBench のコマンド'db_generator.exe -bc *rulefile* 200 2 -0.5 0.1 *outputfile*' を用いてルールを 200 個生成し, ステップ数と実行時間について汎用プロセッサと比較を行った. 比較に用いた汎用プロセッサは Intel 社の Core2Duo U7600(1.2GHz, Cache L1 data 32KB, L1 inst. 32KB, L2 2MB) である. コンパイラは gcc を用い, 最適化オプションは O3 とした. Core2Duo で実現したデータ構造は MTBDD を用い, 非終端節点を if-then-else 分岐命令に置換えた C コードを生成した. これは汎用プロセッサで実現する場合, BDD の方がコードが単純となり, キャッシュのヒット率も高くなり, QDD よりも高速であったためである. 最長パスを通過するようなテストベクトルを生成し, 10 万個のテストベクトルを評価する時間を測定し, 1 ベクトルを評価する平均時間を求めた.

実験結果を表 4 に示す. 表 4 において, *Rule* はパケット分類器のルール名を, *Max. LPL* は最大 LPL を, *Time* は 1 テストベクトルを評価する平均時間を, *Node* は決定グラフの節点数を, *Mem* は必要なメモリ量を表す. 表 4 より, 最長実行時間に関して PBM は Core2Duo よりも 8.1〜11.1 倍高速であり, メモリ量に関して 9.7〜436.2 分の 1 であった.

6. ま と め

本論文では, 並列ブランチング・プログラム・マシン (PBM) を用いたパケット分類器の実現について述べた. 提案手法はパケット分類器のルールの集合を分割し, 複数の BM で並列にパケットを処理する. パケット分類器のルールの集合が与えられたとき, 必要な BM の台数を推定できることを示した. BM を 32 台用いた PBM32 を FPGA 上に実装し, Intel 社の Core2Duo@1.2GHz と比較を行った結果, PBM32 は最長実行時間に関して PBM は Core2Duo よりも 8.1〜11.1 倍高速であり, メモリ量に関して 9.7〜436.2 分の 1 であった.

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] P. C. Baracos, R. D. Hudson, L. J. Vroomen, and P. J. A. Zsombor-Murray, "Advances in binary decision based programmable controllers," *IEEE Transactions on Industrial Electornics,* Vol. 35, No. 3, pp. 417-425, Aug., 1988.

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

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

[4] CISCO: ASA5500 series, http://www.cisco.com/

[5] K. Golnabi, R. K. Min, L. Khan, L. and E. Al-Shaer, "Analysis of Firewall Policy Rules Using Data Mining Techniques," *NOMS2006*, pp. 305-315, April, 2006.

[6] P. Gupta and N. McKeown, "Packet classification on multiple fields," *ACM Sigcomm*, August, 1999.

[7] 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.

[8] Y. Iguchi, T. Sasao, and M. Matsuura, "Evaluation of multiple-output logic functions," *ASPDAC2003* Kitakyushu, Japan, pp. 312-315, Jan. 21-24, 2003.

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

[10] S. Nagayama and T. Sasao, "On the minimization of longest path length for decision diagrams," *IWLS2004*, June 2-4, Temecula, California, U.S.A., pp. 28-35.

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

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

[13] H. Nakahara, T. Sasao, M. Matsuura, and Y. Kawamura, "Emulation of sequential circuits by a parallel branching program machine," *ARC2009*, Karlsruhe, Germany, March 16-18, 2009. *LNCS5443*, pp. 261-267, March 2009.

[14] T. Sasao, H. Nakahara, M. Matsuura and Y. Iguchi, "Realization of sequential circuits by look-up table ring," *MWSCAS2004*, Hiroshima, pp.I:517-I:520, July 25-28, 2004.

[15] T. Sasao and J. T. Butler, "Implementation of multiple-valued CAM functions by LUT cascades," *ISMVL2006*, May, 2006.

[16] T. Sasao, "On the complexity of classification functions," *ISMVL2008*, May 22-24, 2008.

[17] T. Sasao, H. Nakahara, M. Matsuura, Y. Kawamura, and J.T. Butler, "A quaternary decision diagram machine and the optimization of its code," *ISMVL2009*, May, 2009, pp.362-369.

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

[19] D. E. Taylor and J. S. Turner, "ClassBench: a packet classification benchmark," *INFOCOM2005*, Vol. 3, pp. 2068-2079, 13-17, March, 2005.

[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.

[21] 福山 泰介, 笹尾 勤, 松浦 宗寛, "3 アドレス QDD マシン用コードの最適化アルゴリズムについて," 電子情報通信学会 *RECONF* 研究会, 2009-12.