

ベクトル計算機による論理関数の素項の高速生成

越智裕之、高木直史、矢島脩三
京都大学工学部情報工学教室

論理関数の素項の生成は、論理回路の設計における論理関数の最小化において重要な位置を占めている。本稿では共有展開に基づく論理関数の素項の生成法として、テーブル併用共有展開法およびテーブル併用Morreale法を提案する。これらの手法では、処理の高速化と使用領域の節約を図るため、全ての4変数論理関数の全素項を与えるテーブルを使用している。また、ベクトル計算機向きに実現するために、データ構造を工夫している。

ベクトル計算機FACOM VP-400E上に実現したところ、18変数論理関数の全素項が、前者で平均約1.4秒で、後者で平均約5秒で得られた。また、任意の18変数論理関数の全素項の生成に必要な領域は、前者で74Mbyte、後者で169Mbyteである。

High-Speed Generation of Prime Implicants of Logic Functions on Vector Processors

Hiroyuki OCHI, Naofumi TAKAGI and Shozo YAJIMA
Department of Information Science, Faculty of Engineering
Kyoto University, Kyoto 606, JAPAN

The generation of prime implicants of a logic function is a fundamental subject in the minimization of logic functions. In this paper, by modifying consensus expansion, we propose two vector algorithms for the generation of all prime implicants of logic functions, which we call the consensus expansion method with table look-up and the Morreale method with table look-up. A complete table of all prime implicants of all 4-variable logic functions is used to reduce required time and space. Effective data structure suitable for vector processing is adopted.

For the two new algorithms, which were implemented on a vector processor FACOM VP-400E, the average computation time required to generate all prime implicants for 18-variable logic functions is about 1.4sec. and 5sec., respectively, and the required memory space for an arbitrary 18-variable logic function is 74Mbytes and 169Mbytes, respectively.

1. はじめに

近年の集積回路技術の進歩にともない、大規模な論理回路が実現可能になってきた。これにともない、論理関数の簡単化がいっそう重要な課題となっている。論理関数の簡単化でも、特に、最適解(最小積和形)を求める(最小化)ための多くの手法では、まず与えられた論理関数の全ての素項を生成することを必要としている。しかしながら、論理関数の全素項の数は一般に非常に多く、論理関数の全素項の生成には多くの時間と領域を必要とする。一方、近年、科学技術用のスーパー・コンピュータとして、高い最大性能をもつベクトル計算機が開発され、計算量の多い種々の問題に利用されている。特に、最近では、論理シミュレーションなどの非数値計算へも利用されるようになってきている[1]。本稿では、論理関数の全素項をベクトル計算機を用いて効率よく生成する手法について考える。

論理関数の素項の生成法は論理関数の最小化手法と関連して、古くから盛んに研究されている。主なものとして、Quine-McCluskey法[2][3]、和積形の展開に基づく手法[4]~[6]、共有展開に基づく手法[7][8]などがあげられる。共有展開は、論理関数を各変数についてそれぞれその肯定リテラルを含む項、否定リテラル含む項、いずれのリテラルも含まない項(その変数に依存しない項)の3つに展開して全ての内項を生成し、これらの素項としての可能性を調べていくものである。内項を生成する手段としては非常に効率がよいが、素項のみを生成するためには非素項を取り除く操作が必要である。そのため、従来の手法にも様々な工夫がみられる[8]~[10]。

本稿では共有展開に基づくベクトル計算機向きの論理関数の素項生成法として、テーブル併用共有展開法およびテーブル併用Morreale法を提案する。前者は、共有展開により内項を生成したのち、組織的に非素項を除去するものであり、後者は、共有展開をおこなう際に、非素項の生成を防止するTagging Function[8]を生成するものである。本稿で提案するこれらの方法は、特に、ベクトル計算機の高い性能を引き出すため、共に以下の2点が工夫されている。

1. 全ての4変数論理関数の全素項を与えるテーブルを使用し、処理の高速化と使用メモリの節約を図っている
2. 効率よく共有展開や非素項除去ができるデータ構造を採用している

テーブル併用共有展開法及びテーブル併用Morreale法を京都大学大型計算機センターのFACOM VP-400E上に実現したところ、乱数で発生した18変数論理関数の全素項が前者で平均約1.4秒、後者で平均約5秒で得られた。また、任意の18変数論理関数の全素項の生成に必要な領域は前者で74メガバイト、後者で172メガバイトである。

以下、2章で準備として論理関数と素項に関する基本的用語の定義と本稿の実験で扱うベクトル計算機の説明をおこない、3章および4章でテーブル併用共有展開法およびテーブル併用Morreale法について述べ、5章でその評価と考察をおこなう。

2. 準備

2.1 基本的用語

論理変数とは、0または1の値をとる変数であり、 x_i ($i=1, 2, \dots, n$)で表す。 x_i およびその否定 $\bar{x}_i$ をそれぞれ論理変数 x_i のリテラルという。 n 変数論理関数とは、 $\{0, 1\}^n$ から $\{0, 1\}$ への写像であり、 $f(x_1, \dots, x_n)$ や $g(x_1, \dots, x_n)$ 、あるいは単に f や g などで表す。以下、論理変数、論理関数をそれぞれ単に変数、関数と呼ぶことがある。

積項とは、各変数のリテラルをただか一つしか含まないリテラルの論理積であり、 p や q などで表す。

積項などに変数 x_i が含まれていないとき、このことを明示的に表現するために、必要に応じて疑似リテラル $\bar{x}_i (=x_i + x_i)$ を用いる。疑似リテラルを用いた積項 p の表現を p の疑似リテラル表現という。例えば3変数のとき、積項 $x_1 x_2 x_3$ 、 $x_1 x_3$ 、 x_2 、1の疑似リテラル表現は、それぞれ $x_1 x_2 x_3$ 、 $\bar{x}_1 x_2 x_3$ 、 $x_1 x_2 x_3$ 、 $x_1 x_2 x_3$ である。以後、積項は全て疑似リテラル表現で表すものとする。

論理関数 f と積項 p があり、 $p=1$ を満足する全ての変数の組み合わせに対して $f=1$ が成立するとき、積項 p を関数 f の内項という。積項 p と q があり、 $p=1$ を満足する全ての変数の組み合わせに対して $q=1$ が成立するとき、 p は q に包含されるという。論理関数の包含関係も同様に定義される。論理関数 f の内項で、 f の他のいかなる内項にも包含されないものを関数 f の素項という。また、論理関数 f の内項のうち、 f の素項でないものを f の非素項という。

$f(x_i=0)$ は、 $f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$ すなわち、論理関数 f の変数 x_i に定数0を代入して得られる $(n-1)$ 変数論理関数を表す。 $f(x_i=1)$ も同様である。

Shannonの展開定理を一般化した次式を(変数 x_i に関する)共有展開という。

$$f = \bar{x}_i \cdot f(x_i=0) + x_i \cdot f(x_i=1) + f(x_i=0) \cdot f(x_i=1)$$

以下、 $f(x_i=*)$ ($= f(x_i=0) \cdot f(x_i=1)$)を用いて、上式の第3項を $\bar{x}_i \cdot f(x_i=*)$ と書く。

2.2 ベクトル計算機

ベクトル計算機は、配列データ(ベクトル・データ)に対する繰り返し処理を演算パイプラインで高速に実行するスーパー・コンピュータであり、大規模な科学技術計算の要求を満たすべく開発されたものである。しかし、ベクトル計算機の高い最大性能を引き出すためには、(1)ベクトル化率を高くする、(2)ベクトル長を長くする等の工夫が必要であり、コーディングやデータ構造を工夫するだけでなく、アルゴリズムを設計する段階からこれらのことを考慮することが重要である。

本稿の実験で使用したベクトル計算機は、京都大学大型計算機センターのFACOM VP-400Eである。VP-400Eは、広範な応用を指向してベクトル処理の範囲を広げるために、様々な処理機能を備えている。データ型に関しては、浮動小数点数だけでなく、整数、論理データもベクトル演算の対象となっており、32ビット1語

のデータに対し、加減乗除はもとよりビット・ワイズ論理演算をも高速に処理することが可能である。主記憶は256メガバイト実装(うち、ユーザ領域は200メガバイト)されており、これに対し、(1)連続アクセス、(2)等間隔アクセス、(3)リスト・ベクトル・アクセスの3種のベクトル・アクセスが可能である。また、配列要素のうち特定の条件を満たすものだけに処理を行う条件付きベクトル演算や、特定の条件を満たすものだけを集めて新たな配列を構成するベクトル収集操作なども可能である。

上記のベクトル処理機能はVP-400Eに限られたものではなく、HITAC S-820等のベクトル計算機にも備えられているものである。

3. テーブル併用共有展開法

3.1 アルゴリズム

ある $m(\geq 0)$ について、 m 変数論理関数の全素項が既知のとき、 n 変数論理関数 f_{given} の全素項は、以下のアルゴリズムによって生成できる($n > m$ とする)。以下、 $F_i, P_i (m \leq i \leq n)$ はそれぞれ、関数の集合、積項の集合を表す。

[アルゴリズム1]

《入力》 $f_{given}(X_1, \dots, X_n)$: n 変数論理関数

《出力》 P_n : f_{given} の全素項の集合

step1. $F_n := \{f_{given}\}$ (F_n は1要素の集合)

step2. for $k: n$ downto $m+1$ do

$F_{k-1} := \{X_k \cdot f(X_k=*), \bar{X}_k \cdot f(X_k=0), X_k \cdot f(X_k=1) \mid f \in F_k\}$

step3. $P_m := \{f \text{ の全ての素項 } \mid f \in F_m\}$

($f \in F_m$ は、 m 変数関数 f' と f' が依存しない変数よりなる積項との論理積で表される。従って、 $f \in F_m$ の素項は仮定より既知である。)

step4. for $k: m+1$ to n do

P_{k-1} の要素(積項)から以下の条件を満たすものを除いたものの集合を P_k とする。

(条件) 疑似リテラル表現に $\bar{X}_k$ または X_k を含み、かつ、その $\bar{X}_k$ または X_k を X_k^* におきかえて得られる積項が集合 P_{k-1} の要素であること□

step2において、 $(n-m)$ 個の変数 $(X_n, \dots, X_{m+1})$ に関する f_{given} の共有展開がおこなわれている。一回の展開で要素数は(最大)3倍になるから、 F_m の要素数は(最大) 3^{n-m} である。step3で得られる集合 P_m の要素は、 f_{given} を共有展開して得られる関数の全素項の集合である。したがって、集合 P_m の要素は全て f_{given} の内項であり、また、集合 P_m は、 f_{given} の全ての素項を含む。集合 P_m から f_{given} の非素項を取り除くのがstep4である。

以下の定理1より、帰納的に $P_k = \{f \text{ の全ての素項 } \mid f \in F_k\}$ ($m \leq k \leq n$)であることがいえる。つまり、 $P_{k-1} = \{f \text{ の全ての素項 } \mid f \in F_{k-1}\}$ であるならば、 $P_k = \{f \text{ の全ての素項 } \mid f \in F_k\}$ となるよ

う、step4の一回の非素項除去がstep2の一回の共有展開と対応しているのである。従って、step4で得られる P_n は、 f_{given} の全ての素項だけからなる集合である。(定理1の証明は、付録を参照されたい。)

定理1 f を n 変数論理関数 $f(X_1, \dots, X_n)$ とする。ある変数 $X_k (1 \leq k \leq n)$ に注目するとき、 f の素項は、つぎの3種類のうちのいずれかであり、また、それだけである。

(1) $X_k \cdot f(X_n=*)$ の素項。

(2) $\bar{X}_k \cdot f(X_n=0)$ の素項で、 $\bar{X}_k \cdot f(X_k=*)$ に含まれないもの。

(3) $X_k \cdot f(X_n=1)$ の素項で、 $X_k \cdot f(X_k=*)$ に含まれないもの。 □

3.2 ベクトル化手法

step2において、 $f \in F_k$ は、

$$f = p \cdot f'(X_k, \dots, X_1)$$

但し、 p は $L_n \cdot L_{n-1} \cdot \dots \cdot L_{k+1}$ なる積項

L_i は $\bar{X}_i, X_i, X_i^*$ のうちのいずれか

の形で表すことができる。この f' は、真理値表形式の表現、すなわち 2^kbit の0,1の系列で表し、積項 $p = L_n \cdot L_{n-1} \cdot \dots \cdot L_{k+1}$ は、次式により整数で表し、1語に格納する。

$$\sum_{i=k+1}^n r_i \cdot 3^{i-m-1}$$

但し、 $r_i = 0$ ($L_i = \bar{X}_i$ のとき)

$r_i = 1$ ($L_i = X_i$ のとき)

$r_i = 2$ ($L_i = X_i^*$ のとき)

また、集合 F_k の要素数は(最大) 3^{n-k} 個であるから、 F_k を表現するためには $2^k \cdot 3^{n-k} \text{bit}$ あればよい。5変数分の真理値表を1語で表せば、 $2^k \cdot 5 \cdot 3^{n-k}$ 語で表現できる(1語=32bit)。今回使用した F_k のデータ構造を図1に示す。

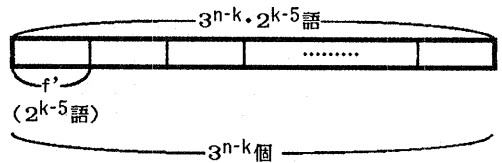


図1. F_k のデータ構造

F_k の要素の中に恒偽であるものがあれば、これを取り除いても解には影響しない。したがって、この処理(枝刈り)を効率よくおこなえば、処理の高速化が期待できる。しかし実験の結果から、枝刈りは必ずしもstep2の各展開ごとにおこなえばよいのではなく、 F_5 のみをその対象としたときに平均的に効率がよいということがわかった。これは、(1)展開の初期において、恒偽である F_1 の要素(関数)の存在する確率が低い、(2)展開の初期の変数の多い(真理値表の大きい)関数の恒偽性判定には時間がかかる、などの理由によるものと考えられる。枝刈りは、ベクトル計算機のベクトル収集機能を用いれば容易に実現できる。

step3は、 $m=4$ 程度であれば、全ての m 変数論理関数の全素項を与えるテーブルを用いて処理することができる。このとき、step3は、step2の最後の展開(F_5 の X_5 に関する展開)とともに処理すれば効率がよい。すなわち、 F_5 の下位16bit($X_5=0$)、上位16bit($X_5=1$)、およびこれらの論理積を指標としてテーブルを参照するようにすればよい。テーブル参照は、ベクトル計算機のリスト・ベクトル・アクセスにより実現できる。

k 変数からなる積項は 3^k 種類あるので、一つの k 変数論理関数の全素項は 3^k bitの0,1の系列(各bitがそれぞれ対応する積項が素項であるか否かを示す)で表す。これを(論理関数のマップ表現にならって)素項のマップ表現と呼ぶことにする。1語で、3変数分の積項(27個)を表せば、 P_k は 3^{n-3} 語で表現できる。

テーブル参照によって得られたデータは、上述の積項 p を数値化したものを指標として 3^{n-3} 語の領域に格納すれば、step4の(条件2)において、疑似リテラル表現に X_k や $\bar{X}_k$ を含む内項に対応する X_k を含む内項のアドレスが線形演算によって一意に与えられる。つまり、この配列の参照は等間隔アクセスによりベクトル化して実現することができる。このとき、step4に必要なメモリ参照の回数は $O(n \cdot 3^n)$ である。図2に今回使用した P_k のデータ構造を示す。

step4を実現する際には、演算パイプラインの並列

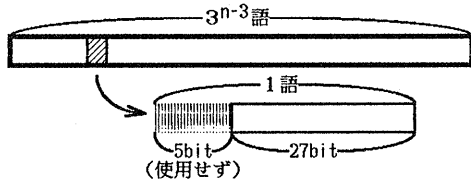


図2. P_k のデータ構造

性を考慮し、非素項除去2回分ごとの処理をまとめておこない、処理の高速化を図っている。

テーブルに必要な領域は、 $3 \cdot 2^{24} = 196,608$ 語である。しかし、ベクトル計算機上に実現したところ、テーブル参照において主記憶のアクセスのバンク競合がみられたため、同じテーブルを4枚用意し、交互にアクセスするようにしてこれを防いでいる。

このデータ構造において、 n 変数の任意の論理関数の全素項を生成するのに必要な領域は、 $O(3^n)$ である。これは、 $O(3^n/n)$ 個の素項をもつ関数が存在する^[11]ことを考えれば、妥当であるといえる。

4. テーブル併用Morreale法

4.1 アルゴリズム

ある $m(\geq 0)$ について、 m 変数論理関数の全素項が既知のとき、 n 変数論理関数 f_{given} の全素項は、以下のアルゴリズムによって生成できる($n > m$ とする)。以下、 $[f, g_1, \dots, g_j]$ は、関数 $f, g_1, \dots, g_j$ からなる組(tuple)を表し、 $F_i(m \leq i \leq n)$ 、 P はそれぞれ、組の集合、積項の集合を表す。

[アルゴリズム2]

《入力》 $f_{given}(X_1, \dots, X_n)$: n 変数論理関数

《出力》 P : f_{given} の全素項の集合

step1. $F_n := \{[f_{given}]\}$

(F_n は1関数からなる組1つを要素とする集合)

step2. for $k: n$ downto $m+1$ do

$F_{k-1} := \{[X_k \cdot f(X_k=*),$

$\bar{X}_k \cdot g_1(X_k=*), \dots, \bar{X}_k \cdot g_j(X_k=*)],$

$[X_k \cdot f(X_k=0),$

$\bar{X}_k \cdot g_1(X_k=0), \dots, \bar{X}_k \cdot g_j(X_k=0),$

$\bar{X}_k \cdot f(X_k=*)],$

$[X_k \cdot f(X_k=1),$

$X_k \cdot g_1(X_k=1), \dots, X_k \cdot g_j(X_k=1),$

$X_k \cdot f(X_k=*)]$

$| [f, g_1, \dots, g_j] \in F_k \}$

step3. $P := \{f \text{ の素項のうち、} g_1, \dots, g_j \text{ いずれの素項でもないもの} \mid [f, g_1, \dots, g_j] \in F_m\}$

($[f, g_1, \dots, g_j] \in F_m$ なる f や $g_1, \dots, g_j$ は、 m 変数関数とそれが依存しない変数よりなる積項との論理積で表される。従って、これらの関数の素項は仮定より既知である。)

step2において、 $(n-m)$ 個の変数($X_n, \dots, X_{m+1}$)に関する f_{given} の共有展開がおこなわれている。このとき同時に、非素項の生成を防ぐためのTagging Function g_i が生成される。step3で得られる集合 P は、 f_{given} の全ての素項のみからなる。このため、非素項を除去する手続きは不要である。なお、このアルゴリズムは $m=0$ のとき、Morrealeの提案したアルゴリズム[8]（以下、Morreale法という）と等価である*。この意味でテーブル併用Morreale法はMorreale法を一般化したものであるといえる。

以下の定理2より、上のアルゴリズムで得られる P が f_{given} の全ての素項のみからなる集合であることが示される(step2の各展開に定理2を適用すればよい)。なお、定理2は $j=0$ のとき、定理1と等価である。(定理2の証明は付録を参照されたい。)

定理2 f を n 変数論理関数、 $g_1, \dots, g_j$ を f に包含される n 変数論理関数とする。 f の素項で、 $g_1, \dots, g_j$ に包含されないものは、つぎの3種類のうちのいずれかであり、また、これだけである。

(1) $X_n \cdot f(X_n=*)$ の素項で、 $X_n \cdot g_1(X_n=*), \dots, X_n \cdot g_j(X_n=*)$ いずれにも包含されないもの。

(2) $\bar{X}_n \cdot f(X_n=0)$ の素項で、 $\bar{X}_n \cdot g_1(X_n=0), \dots, \bar{X}_n \cdot g_j(X_n=0)$ および $\bar{X}_n \cdot f(X_n=*)$ いずれにも包含されないもの。

(3) $X_n \cdot f(X_n=1)$ の素項で、 $X_n \cdot g_1(X_n=1), \dots, X_n \cdot g_j(X_n=1)$ および $X_n \cdot f(X_n=*)$ いずれにも包含されないもの。

* 改良共有展開法[9]は、Morreale法[8]と等価であると考えられる。

4.2 ベクトル化手法

アルゴリズム2の F_k はTagging Functionがあることを除けば、アルゴリズム1の F_k とほぼ同様に表現できる。 F_k の各組のTagging Functionの数 j は、最大 $(n-k)$ 個である。図3に、今回使用した F_k のデータ構造を示す。ここで、Tagging Functionの数が $(n-k)$ 個よりも少ない組もあるが、未使用領域は全て0(恒偽)とすれば得られる結果は変わらない。

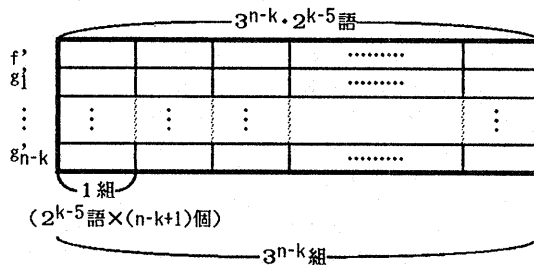


図3. F_k のデータ構造

Pのデータ構造は図2と全く同じである。その他、テーブルの構成や、step1からstep3の処理の実現法なども、ほぼ3.2節で述べたのと同様である。

5. 評価と考察

5.1 評価

今回作成したテーブル併用共有展開法およびテーブル併用Morreale法のプログラムを京都大学大型計算機センターのベクトル計算機FACOM VP-400Eで実行したときの処理時間をそれぞれ図4、図5に示す(但し、

後者ではテーブルを1枚だけ使用)。横軸は真理値表全体に占める1の比率である。このベンチマークでは、Lehmerの合同法で得られる疑似一様乱数[12]で生成した論理関数を使用した。図の各プロットは、10個の関数の平均値を表す(恒真および恒偽は各1個)。これによると、テーブル併用共有展開法およびテーブル併用Morreale法で18変数論理関数の全素項を生成するのに必要な時間は、前者で平均約1.4秒、後者で平均約5秒であった。

また、他のアルゴリズムとのベクトル計算機上での処理速度の比較を図6に、ベクトル加速率(スカラー実行所要時間/ベクトル実行所要時間)を表1に示す[9][13][14]。

表2に、今回作成したプログラムが実際に必要とするメモリ容量を示す。これは、FORTRAN77でコーディングしたときに宣言した配列変数の領域であり、非配列変数やオブジェクトコードは含めていない。

また、今回作成したプログラムは移植性もよく、プログラムをほとんど変更せずに、効率を損なうことなく、東京大学大型計算機センターのベクトル計算機HITAC S-820/80上で実行できた。VP-400Eと同様のベンチマークをとったところ、例えば18変数論理関数の全素項はテーブル併用共有展開法で平均約0.4秒で生成できた。また、一般の汎用計算機上でも比較的効率が良く、本学情報工学科の汎用計算機FACOM M-360R上でテーブル併用共有展開法で同様のベンチマークをとったところ、12変数論理関数の全素項を平均約0.2秒で生成できた。

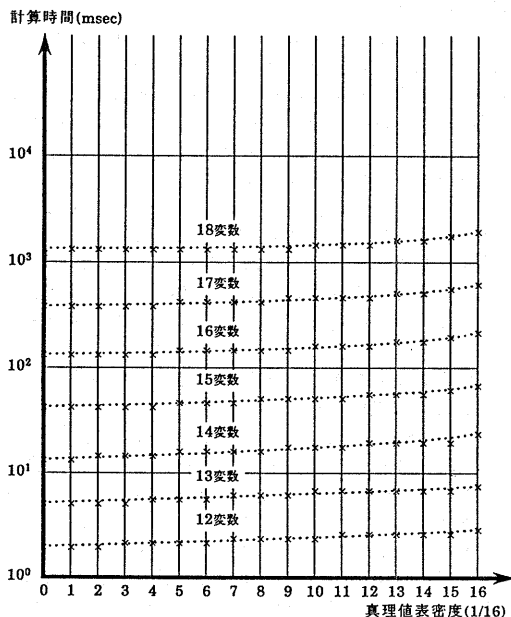


図4. テーブル併用共有展開法の計算時間

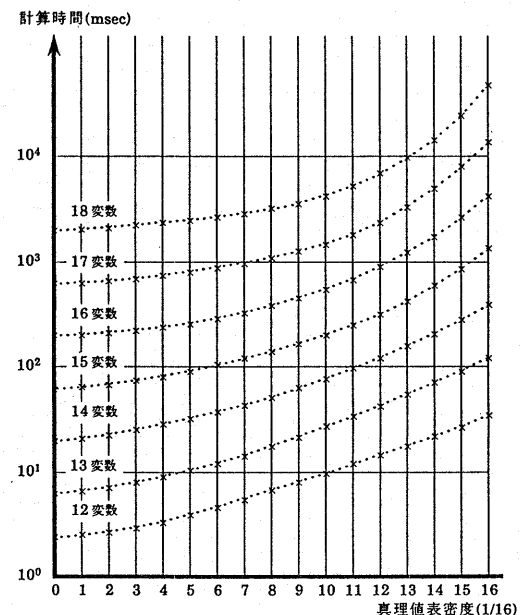


図5. テーブル併用Morreale法の計算時間

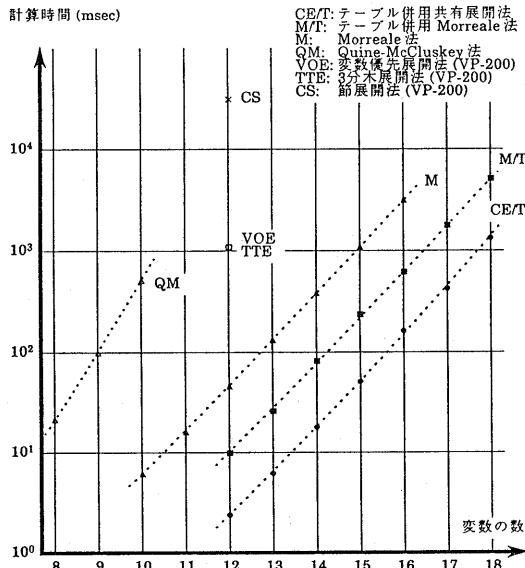


図6. 計算時間の比較

表1. ベクトル加速率の比較

アルゴリズム	測定条件	スカラー実行	ベクトル実行	ベクトル加速率
テーブル併用 共有展開法	12変数 VP-400E	33.3msec.	2.31msec.	14.42
テーブル併用 Morreale法	12変数 VP-400E	65.9msec.	9.69msec.	6.81
Morreale法	12変数 VP-400E	571.4msec.	47.5msec.	12.03
Quine- McCluskey法	10変数 VP-400E	5,187msec.	523msec.	9.92
変数優先展開法	12変数 VP-200	10,851msec.	1,186msec.	9.15
3分木展開法	12変数 VP-200	10,579msec.	1,183msec.	8.94
節展開法	12変数 VP-200	46,750msec.	32,009msec.	1.46

表2. 必要なメモリ容量

変数の数	テーブル併用 共有展開法	テーブル併用 Morreale法	Morreale法
12変数	3,242KB	948KB	2,076KB
13変数	3,434KB	1,332KB	6,228KB
14変数	4,011KB	2,537KB	18,684KB
15変数	5,744KB	6,304KB	56,051KB
16変数	10,940KB	18,068KB	168,152KB
17変数	26,529KB	54,743KB	-----
18変数	73,296KB	168,920KB	-----

5.2 考察

表1から明らかなように、本稿で提案した手法は、いずれも非常に高いベクトル加速率を示しており、ベクトル計算機の性能を十分に引き出しているといつてよい。これは、(1)アルゴリズム全体が比較的単純な処理の繰り返しからなっており、プログラムの主要なループが全てベクトル化できたこと、(2)真理値表や素項のマッピングなどの大きな配列に対する繰り返し処理が多く、ベクトル長を十分に長くできたこと、などの理由によるものと考えられる。

Morreale法は、非素項が生成しないよう工夫をほどこした共有展開法であると考えられる。これに対し、テーブル併用共有展開法は、非素項の生成を許し、これの除去に工夫を凝らしたものである。結果的にテーブル併用共有展開法がテーブル併用Morreale法より高速であったことは、アクセスすべきメモリが減ったことから考えれば当然のこととはいえ、興味深い。

データ構造はほぼ同様のマップ表現であるが、テーブル併用共有展開法において非素項が高速に除去できたのは、論理関数のみならず、(素)内項の表現にもマップ表現を用いたためである。マップ表現は、ベクトル計算機のビット・ワイス論理演算および等間隔アクセスとの整合性がよい。

テーブル参照が処理速度に及ぼした効果は、図6でMorreale法とテーブル併用Morreale法の処理速度を比較すれば明らかである。また、約0.8メガバイトのテーブルを使用したことによって、共有展開の回数が4回減り、例えば18変数のとき数百メガバイトの領域が節約された。従来の手法では、大型計算機を用いても12~16変数程度の論理関数までしか扱えなかったが、本稿の手法では、テーブル参照により18変数までの論理関数が扱えるようになったのである。

また本稿では、アルゴリズム1やアルゴリズム2のstep3で、m変数論理関数の素項(既知と仮定されている)を与えるために、主記憶上の配列に用意したテーブルを用いたが、他にも種々の方法が考えられる。例えば、m=18のテーブル併用Morreale法のstep3をn=18のテーブル併用共有展開法を用いて実現すれば、200メガバイトの領域で20変数以上の論理関数の素項を生成することが可能であると考えられる。また、既存のベクトル計算機に専用ハードウェアを付加し、例えば6変数論理関数の素項を与えるベクトル命令を追加すれば、さらに高速に素項を生成することができると考えられる。

また、本稿のアルゴリズムは不完全指定論理関数の素項生成にも応用できる。テーブル併用共有展開法では、don't-care setとon setをともにon setとみなして素項を生成し、そのうちdon't-care setに含まれるもの(don't-care setの素項であるもの)を取り除けばよい。つまり、このアルゴリズムで2回素項を生成すればよいのである(実際には、後者のdon't-care setの素項の生成でstep4の処理は不要である)。また、テーブル併用Morreale法ではdon't-care setをTagging Function g_0 によって与えるだけでよく[8]、プログラムをわずかに変更すれば実現できる。

6. おわりに

本稿で提案した手法により、論理関数の素項がベクトル計算機を用いて高速に生成できることが示された。これにより、論理関数の最小化をより高速に実現できる可能性が示されたといえる。また、このプログラムを用いて論理関数の素項の統計的な諸性質を明らかにし、ヒューリスティックによる論理関数の単純化をはじめとする種々の組合せ問題に理論的基礎を与えることも考えられる。また、本稿の結果から組合せ問題に対するベクトル計算機の有用性が示されたといえる。

謝辞

ご討論いただいた本学平石裕実助教授に感謝いたします。また、アルゴリズムの検討からコーディングの細部まで、有益なご助言をいただいた石浦菜岐佐氏をはじめ、矢島研究室の諸氏に感謝いたします。また、図5のQuine-McCluskey法のプログラムは、本学下研究室の網谷勝俊氏が作成したものに筆者が手を加えたものであることを申し添えておきます。

参考文献

- [1] 石浦 安浦 矢島: "High-speed logic simulation on vector processors", IEEE Trans. Computer-Aided Design, Vol. CAD-6, No. 3, pp. 305-321, (May 1987)
- [2] W.V. Quine: "A way to simplify truth functions", Amer. Math. Monthly, Vol. 62, pp. 627-631, (Nov. 1955)
- [3] E.J. McCluskey, Jr.: "Minimization of Boolean functions", Bell System Tech. J., Vol. 35, No. 6, pp. 1417-1444, (Nov. 1956)
- [4] R.J. Nelson: "Simplest normal truth functions", J. Symbolic Logic, Vol. 20, No. 2, pp. 105-108, (June 1955)
- [5] J.R. Slagle C.L. Chang R.C.T. Lee: "A new algorithm for generating prime implicants", IEEE Trans. Computers, Vol. C-19, No. 4, pp. 304-310, (April 1970)
- [6] 上林 岡田 矢島: "節展開法を用いた論理関数の主項の生成", 信学論, Vol. J62-D, No. 2, pp. 89-96, (Feb. 1979)
- [7] P. Tison: "Generalization of consensus theory and application to the minimization of Boolean functions", IEEE Trans. Electron. Computers, Vol. EC-16, No. 4, pp. 446-456, (Aug. 1967)
- [8] E. Morreale: "Recursive operators for prime implicant and irredundant normal form determination", IEEE Trans. Computers, Vol. C-19, No. 6, pp. 504-509, (June 1970)
- [9] 矢板 積田 西村: "共有展開による論理関数の主項の生成", 信学技報, EC85-3, pp. 23-29, (May 1985)
- [10] 加賀谷 高木 矢島: "ベクトル計算機向きの論理関数の素項の生成法", 情報処理学会 設計自動

化34-4 (Oct. 1986)

- [11] B. Dunham R. Fridshal: "The problem of simplifying logical expressions", J. Symbolic Logic, Vol. 24, No. 1, pp. 17-19, (Mar. 1959)
- [12] 富士通: "FACOM FORTRAN SSL-II 使用手引書", 第5版, pp. 448-450, (Dec. 1980)
- [13] 加賀谷: "Vector algorithms for generating prime implicants of logic functions", 修士論文, 京都大学工学部情報工学教室, (Feb. 1987)
- [14] 高木 越智 矢島: "Vector algorithms for generating prime implicants of logic functions", Proceedings of Third International Conference on Supercomputing (May 1988) (発表予定)

付録 (定理の証明)

定理1 f を n 変数論理関数とする。 f の素項は、つぎの3種類のうちのいずれかであり、また、これだけである。

- (1) $X_n \cdot f(X_n = *)$ の素項。
- (2) $\bar{X}_n \cdot f(X_n = 0)$ の素項で、 $\bar{X}_n \cdot f(X_n = *)$ に含まれないもの。
- (3) $X_n \cdot f(X_n = 1)$ の素項で、 $X_n \cdot f(X_n = *)$ に含まれないもの。 □

< 定理1の証明 >

f の素項で、その疑似リテラル表現に X_n を含むものは $X_n \cdot f(X_n = *)$ の内項であり、 $\bar{X}_n$ を含むものは $\bar{X}_n \cdot f(X_n = 0)$ の、 X_n を含むものは $X_n \cdot f(X_n = 1)$ の内項である。ここで仮に、 $X_n \cdot f(X_n = *)$ や $\bar{X}_n \cdot f(X_n = 0)$ や $X_n \cdot f(X_n = 1)$ の内項で、 f の素項を包含するものがあつたとする。ところが、 $f = X_n \cdot f(X_n = *) + \bar{X}_n \cdot f(X_n = 0) + X_n \cdot f(X_n = 1)$ であるから、 $X_n \cdot f(X_n = *)$ や $\bar{X}_n \cdot f(X_n = 0)$ や $X_n \cdot f(X_n = 1)$ の内項で、 f の内項でないものはない。よって、 f の素項を包含する f の内項が存在することになり、素項の定義に矛盾する。よって、 f の素項は $X_n \cdot f(X_n = *)$ 、 $\bar{X}_n \cdot f(X_n = 0)$ 、 $X_n \cdot f(X_n = 1)$ のいずれかの素項である。

また、 $X_n \cdot f(X_n = *)$ の素項は、その疑似リテラル表現に X_n を含むから、 $\bar{X}_n \cdot f(X_n = 0)$ や $X_n \cdot f(X_n = 1)$ の素項には含まれない。また、 $\bar{X}_n \cdot f(X_n = 0)$ や $X_n \cdot f(X_n = 1)$ の素項で f の素項でないものは、 f の非素項であるが、これを包含する f の素項は $X_n \cdot f(X_n = *)$ の素項以外にはありえない。

以上より、定理1が得られる。 □

定理2 f を n 変数論理関数、 $g_1, \dots, g_l$ を f に包含される n 変数論理関数とする。 f の素項で、 $g_1, \dots, g_l$ いずれにも包含されないものは、つぎの3種類のうちのいずれかであり、また、これだけである。

- (1) $X_n \cdot f(X_n = *)$ の素項で、 $X_n \cdot g_1(X_n = *)$ 、 $\dots$ 、 $X_n \cdot g_l(X_n = *)$ いずれにも包含されないもの。
- (2) $\bar{X}_n \cdot f(X_n = 0)$ の素項で、 $\bar{X}_n \cdot g_1(X_n = 0)$ 、 $\dots$ 、 $\bar{X}_n \cdot g_l(X_n = 0)$ および $\bar{X}_n \cdot f(X_n = *)$ いずれにも包含されないもの。

(3) $\bigvee_n \cdot f(X_n=1)$ の素項で、 $\bigvee_n \cdot g_1(X_n=1), \dots, \bigvee_n \cdot g_{j-1}(X_n=1)$ および $\bigvee_n \cdot f(X_n=*)$ いずれにも包含されないもの。 $\square$

< 定理2の証明 >

j に関する数学的帰納法による。

j = 0 のとき、定理 1 より定理 2 は正しい。

j > 0 のとき、j - 1 において正しいと仮定する。

つまり、f の素項で、 $g_1, \dots, g_{j-1}$ に包含されないものは、つぎの3種類のうちのいずれかであり、また、これだけであるとする。

(1) $\bigvee_n \cdot f(X_n=*)$ の素項で、 $\bigvee_n \cdot g_1(X_n=*), \dots, \bigvee_n \cdot g_{j-1}(X_n=*)$ いずれにも包含されないもの。

(2) $\bigwedge_n \cdot f(X_n=0)$ の素項で、 $\bigwedge_n \cdot g_1(X_n=0), \dots, \bigwedge_n \cdot g_{j-1}(X_n=0)$ および $\bigvee_n \cdot f(X_n=*)$ いずれにも包含されないもの。

(3) $\bigvee_n \cdot f(X_n=1)$ の素項で、 $\bigvee_n \cdot g_1(X_n=1), \dots, \bigvee_n \cdot g_{j-1}(X_n=1)$ および $\bigvee_n \cdot f(X_n=*)$ いずれにも包含されないもの。

すると、f に包含される関数 g_j に関して、次のことがいえる。

上の(1)のうち、 g_j に包含されるものは、 $\bigvee_n \cdot g_j(X_n=*)$ に包含される。

上の(2)のうち、 g_j に包含されるものは、 $\bigwedge_n \cdot g_j(X_n=0)$ に包含される。

上の(3)のうち、 g_j に包含されるものは、 $\bigvee_n \cdot g_j(X_n=1)$ に包含される。

よって、j のときも正しい。

以上より、定理 2 が得られる。 $\square$