アルゴリズム 48−11
（1995．11．17）

二分決定グラフのトップダウン構成法と並列化

定兼邦彦　早瀬千善　今井浩
東京大学理学部情報科学科

二分決定グラフ (BDD) は、論理関数を表現する有向無閉路グラフであり、多くの関数をコンパクトに表現でき、関数の等価性の判定を簡単に行うことができる。Bryant によって効率のいい操作が提案されてから、BDD は論理設計や人工知能、組合せ論などの分野で使われている。本稿では、論理関数が正リテラルのみの和積標準形 (正 CNF) の場合に、それを表す BDD をトップダウンに構成する方法を示し、それを非共有記憶型の並列計算機である AP-1000+ 上で実装する。そして、グラフの支配集合と独立点集合を表す関数の BDD について実験を行い、プロセッサ数に比例する速度向上がのぞめることを示す。

A Parallel Top-down Algorithm to Construct Binary Decision Diagrams

Kunihiko Sadakane, Kazuyoshi Hayase, Hiroshi Imai
Department of Information Science, University of Tokyo

Binary Decision Diagram (BDD) is a directed acyclic graph, which represents many boolean functions efficiently. After Bryant proposed an efficient algorithm to manipulate them, BDD is used not only in logic design area, but also in combinatorial problems. In this paper, we present an algorithm to construct a BDD representing positive conjunctive normal form (CNF) of a boolean function and implement it on AP-1000+, distributed memory parallel computer. We also experiment on BDD representing functions of independent sets and dominating sets of a graph. Computational results show that speedup is proportional to the number of processors.

1　はじめに

二分決定グラフ (BDD: Binary Decision Diagram) は、論理関数を表現する有向無閉路グラフであり、多くの関数をコンパクトに表現できる。また、現れる論理変数の順序を決めると表現が一意になるため、関数の等価性の判定を簡単に行うことができる。Bryant [2] によって効率のいい操作が提案され、また、SBDD [6] などのライブラリが配布されているので、BDD は論理設計や人工知能、組合せ論などの分野で使われている。

BDD を構成する方法は、従来は BDD 間で論理演算を繰り返してボトムアップに構成する方法があったが、BDD のノードをトップダウンに直接生成する方法 [5] も提案されている。ボトムアップに構成する方法では、論理演算を行う順序によって中間的に生成されるの BDD のノード数が変化する。その結果、最終的に構成される BDD のノード数に比べて中間の BDD のノード数が非常に多くなることがあり、記憶量や速度の点で問題になる。トップダウンに構成する方法では、論理変数に真理値を割り当てた部分問題を生成し、その部分問題同士の等価性の判定を行い、余分なノードを生成しないようにする。この手法では最終的な BDD のサイズに比例した記憶量で構成できるが、部分問題の等価性の判定は一般には co-NP 完全であるため、任意の論理関数についてこの方法を適用するのは難しい。

問題によっては BDD の操作に多くの処理時間が必要な場合があるので、その並列化が提案されている。共有記憶型の並列計算機での手法 [4] は、メモリへの同時書き込みを防ぐためのロックの操作が必要であり、そのコストが大きい。非共有記憶型の AP-1000 での手法 [1] では部分問題を表す BDD を別々のプロセッサで独立に構成し、最終的な結果の BDD を 1 つのプロセッサに構成する手法を用いている。しかしこの手法では演算の処理にともなう通信は必要ないが、すべてのノードを 1 つのプロセッサに集める部分がボトルネックになる。

我々の手法では、トップダウン構成法を用い、ノードを複数のプロセッサに分散させながら BDD を構成する。すべてのプロセッサで共通なハッシュ関数を使うことにより、等価なノードが複数できることはなく、さらに

BDD のノードを 1 つのプロセッサに集めなくてもよいので、出力サイズに比例する記憶領域で構成でき、大きな BDD を作ることもできる。

本稿では、論理関数が正のリテラルのみの和積標準形 (正 CNF) の場合に、その論理関数を表す BDD をトップダウンに構成する方法を示し、それを非共有記憶型の並列計算機である AP-1000+上で実現する方法を示す。そして、正 CNF 式で表せる、グラフの支配集合と独立点集合を表す関数の BDD について、グラフのサイズと BDD のノード数の関係や、プロセッサ数による速度向上を調べる。

2 準備

2.1 BDD

BDD は、論理関数を表す有向無閉路グラフである。BDD の各ノードは論理関数を表す。ノードには定数ノードと変数ノードがある。定数ノードには 0,1 がラベル付けされており、それぞれ恒偽、恒真関数を表す。変数ノード v には変数 x_l がラベル付けされており、v の 0 枝、1 枝の指すノード $v.l$, $v.r$ は、v の表す論理関数において変数 x_l に 0,1 を代入した関数を表す。BDD のルートのノードは変数 x_n がラベル付けされており、そこから定数ノードまでのパスに現れるノードのラベルが全順序 $x_n > \cdots > x_1$ に矛盾しないとき、この BDD を全順序付き二分決定グラフ (OBDD) という。変数 x_l がラベル付けされたノードをレベル l のノードと呼ぶとする。定数ノードのレベルは 0 とする。BDD の各レベルでの幅とは、そのレベルのノードの数を表す。

n 変数の論理関数 f を表す BDD のレベル l のノードは、f の論理変数 $x_n, \cdots, x_{l+1}$ に真理値を割り当てた関数、すなわち部分関数を表す。レベル l の 2 つのノード v_1 , v_2 が等価とは、v_1 , v_2 の表す論理関数 f_1 , f_2 が等価であること、すなわち f_1 , f_2 の変数 $x_l, \cdots, x_1$ に 0,1 をどのように割り当てても f_1 , f_2 の値が同じであることをいう。また、あるノード v の 0 枝、1 枝の指すノードが等価な場合、v を冗長なノードという。等価なノードがなく、ルートから定数ノードまでの全てのパスの長さが n である OBDD を QOBDD、等価なノードも冗長なノードもない OBDD を ROBDD という。

2.2 正 CNF 式

本稿では正のリテラルのみの和積標準形 (正 CNF) の論理関数を表す BDD の構成を示す。変数集合を $X = \{x_1, \cdots, x_n\}$ とすると、

$$\begin{aligned} F &= C_1 \wedge \cdots \wedge C_m \\ C_i &= (x_{i1} \vee x_{i2} \vee \cdots \vee x_{ik_i}) \quad (x_{ij} \in X) \end{aligned}$$

という形の式 F を正 CNF 式といい、C_i を F の節という。

補題 1 *F を正 CNF 式、C を F が含む節とする。$F = C \wedge G$ と書けるとすると、F, G が表す関数 f, g が等価 ($f \equiv g$) $\iff$ G は $D \to C$ となる節 D を含む。*

証明: $\Leftarrow$) $G = D \wedge H$ とすると、$D \to C$ であるから $C \wedge D \equiv D$ 。よって $f \equiv C \wedge D \wedge H \equiv D \wedge H \equiv g$ となり成り立つ。

$\Rightarrow$) f と g は等価とし、G の全ての節 D が $D \not\to C$ とする。C に現れる変数を 0 、その他の変数を 1 にしたとき、C は充足されず $f \equiv 0$ となる。$D \not\to C$ であり、F は正リテラルしか現れないので、G の節 D はすべて充足される。よって $g \equiv 1$ となり、$f \equiv g$ に矛盾する。 □

3 BDD のトップダウン構成法

n 変数の論理関数 f を表す QOBDD をトップダウンに構成するアルゴリズムは、次のようになる。

1. f を表すルートノードを作る。
2. $l=n$ とする。
3. レベル l のすべてのノード v に対して、次のことを行う。
3-1. v の表す関数 g に x_l に $0,1$ を代入してできる関数 f_0, f_1 を求める。
3-2. レベル $l-1$ のノードで、f_0 , f_1 と等価な関数を表すノード v_0 , v_1 を探す。
存在しない場合はレベル $l-1$ に新しくノードを作る。
3-3. $v.l \leftarrow v_0$, $v.r \leftarrow v_1$ とする。
4. $l>1$ ならば $l \leftarrow l-1$ として3へ。
5. 終了。

3.1 CNF 式の等価性の判定

f を n 変数の正CNF式、変数の割り当て a を、$(a_{l+1}a_{l+2}\cdots a_n) \in \{0,1\}^{n-i}$ とする。f , C_i の変数 $x_{l+1},\cdots,x_n$ に $a_{l+1},\cdots,a_n$ を代入した関数を $f|_a$, $C_i|_a$ で表すとする。節 C_i から変数 $x_{l+1},\cdots,x_n$ を除いたものを $C_i^{(l)}$ とすると、$f|_a$ の各節 $C_i|_a$ は次のいずれかである。

$$C_i|_a = \begin{cases} 1 & (\ a_j=1 \text{ となる } x_j \text{ が } C_i \text{ に含まれる}) \\ 0 & (\ a \text{ によって } C_i \text{ が恒偽になる}) \\ C_i^{(l)} & (\text{それ以外}) \end{cases}$$

$C_i|_a$ の節の中に0になっているものが1つでもある場合は $f|_a=0$ となり、$C_i|_a$ の節がすべて1になれば $f|_a=1$ となる。よってこの場合は等価性の判定は簡単なので別に考える。それ以外の場合は $f|_a$ はいくつかの $C_i^{(l)}$ $(1 \le i \le m)$ の積で表される。そこで、ベクトル $h'_a=(h'_{a1},\cdots,h'_{am})$ を次のように決める。

$$h'_{aj} = \begin{cases} 1 & (\ \exists i\ C_i|_a = C_i^{(l)} = C_j^{(l)}\ , \forall k((C_j^{(l)} = C_k^{(l)}) \rightarrow j \le k)) \\ 0 & (\ C_j|_a \neq C_j^{(l)} \text{ または } \exists k\ k<l\ ,\ C_k^{(l)} = C_j^{(l)}) \end{cases}$$

すると $f|_a$ は $h'_{ai}=1$ となる $C_i^{(l)}$ の積で表される。

補題1より、f の節 D,C において、$D \rightarrow C$ ならば C を除いたCNF式が等価な関数を表す。よって、f から冗長な節 C をなくすために、次のように h'_a から h_a をつくる。

$$h_{ai} = \begin{cases} 1 & (h'_{ai}=1\ , \forall j(h'_{aj}=1 \rightarrow (C_j^{(l)} \not\rightarrow C_i^{(l)}))) \\ 0 & (h'_{ai}=0 \text{ または } \exists j(h'_{ai}=h'_{aj}=1\ ,\ C_j^{(l)} \rightarrow C_i^{(l)})) \end{cases}$$

こうしても、h_a の表す論理式は $f|_a$ と等価であり、次の定理が成り立つ。

定理 1 変数の割り当て a_1 , a_2 があり、$f|_{a_1}$ と $f|_{a_2}$ がともに恒真関数でも恒偽関数でもないとき、$f|_{a_1}$ と $f|_{a_2}$ が等価 $\iff$ $h_{a_1}=h_{a_2}$

証明: $h_{a_1}=h_{a_2}$ のとき $f|_{a_1}=f|_{a_2}$ なので等価。$f|_{a_1}$ と $f|_{a_2}$ が等価のとき、$h_{a_1} \neq h_{a_2}$ とする。補題1より h_{a_1} , h_{a_2} の表す論理式 f_1 , f_2 は $f|_{a_1}$, $f|_{a_2}$ と等価であり、それらは冗長な節を含まない。$h_{a_1} \neq h_{a_2}$ より、一般性を失わずに、f_1 に含まれ、f_2 に含まれない節 $C_i^{(l)}$ があるとできる。f_1 と f_2 は等価なので $f_1 \wedge f_2 \equiv f_2$ だが左辺は $C_i^{(l)}$ を含むので矛盾する。 □

以上より、f_a を表すBDDのノードを作る際に、h_a をハッシュ表のキーとして登録すれば、等価なノードを複数個作らずにBDDを構成できる。

4 並列化

4.1 アルゴリズム

我々のトップダウン手法を用いたBDD構成を並列に行うには、上の h_a によってノードをどのプロセッサに作るかを決めればよい。こうすれば複数のプロセッサに等価なノードができることはなくなる。各プロセッサの動作は次のようになる。

1. ある変数の割り当て a とレベル l が送られてきたら、$f|_a$ を表すノード v がレベル $l-1$ にすでに存在するか調べる。
2. v がなければ作り、レベル $l-1$ の変数に $0,1$ を割り当て、v の 0 枝、1 枝の先のノードが作られるべきプロセッサに 変数の割り当てとレベル $l-1$ を送る。
3. 親のノードに v の存在するアドレスを返す。
4. 終了のメッセージが送られてきたら終了。

この手法では、プロセッサ間で同期をとる必要が無いので高速になり、ロードバランスもよくなる。しかし、アルゴリズムがいつ停止したかを判定できない。そこで、ホストのプロセッサが、すべてのプロセッサが受信待ちになっているかを監視することで終了の判定を行う。

h_a によってプロセッサを決めるには、h_a を 2 進数とみなし、それを 8 ビットずつにわけたものを $h_1,\cdots,h_k$ とすると

$$pid = ((((h_1 \cdot s_1 + h_2)s_1 + h_3)\ldots) + h_k + (n-l)\cdot s_2) \bmod N$$

とし、この値を使う。ここで l はレベル、N はプロセッサ数であり、s_1 , s_2 は実験により 49999 , 29999 とした。

4.2 実験

グラフの支配集合を表す BDD について実験を行った。グラフ $G(V,E)$ の全ての支配集合を表す関数 (DS) は、グラフの点を論理変数にわりあてると、

$$DS(G) = \bigwedge_{u\in V}(u \vee (\bigvee_{uv\in E} v))$$

とかけ、これは正 CNF 式になっている。$n\times n$ の格子状のグラフの場合、変数の数は n^2 、節の数も $m=n^2$ である。また、BDD の変数順は行優先とした。プロセッサ数 256 の AP-1000+で $n\times n$ 点の格子状のグラフでの支配集合を表す BDD を構成する時間とノード数は表 1のようになる。この表から、グラフが大きくなるにつれて 1つのノードをつくる時間が遅くなっていることがわかる。これは等価性の判定時に冗長な節をはぶく操作が $O(m^2)$ つまり $O(n^4)$ かかっているからである。また、表 1は、等価性の判定に h_a でなく h'_a を使った場合、つまり冗長な節があるがある時は等価でないとした場合の構成時間とノード数も表している。格子状のグラフの場合、2つの BDD のノード数はほとんど差がなく、冗長な節をはぶく操作をしない方が非常に高速になっている。よって、等価性の判定を h'_a で行い、BDD を構成したあとで等価なノードをなくすという方法が考えられる。これは、従来の手法と同様に、BDD の各ノード v の 0 枝、1 枝の先のノードのあるプロセッサ番号とプロセッサ内のアドレスによって v を生成するノードを決めるという作業を行えばよく、これはノードを 1つのプロセッサに集めずに分散させたまま行うことができる。

グラフの独立点集合 (IS) は

$$IS(G) = \bigwedge_{uv\in E(G)}(\bar{u}\vee\bar{v})$$

となり、これは負リテラルの 2-CNF 式になっている。この場合は冗長な節をはぶく操作が簡単になる。表 1は格子状のグラフの独立点集合を表す BDD の構成する時間とノード数を表しているが、時間はノード数にほぼ比例していることがわかる。また、DS と比べて 1つのノードを作る時間が非常に短くなっていることがわかる。

BDD のノード数は、IS ではほぼ $n^2\left(\frac{1+\sqrt{5}}{2}\right)^n$ 、DS では $n^2 3^n$ 以下になることはわかっているが [3]、表 1により DS のノード数が $n^2(2.3962442)^n$ 程度になることがわかった。この実験から、$n\times n$ の格子状のグラフでの支配集合の BDD の各レベルの幅の大きさ W_n が、実験した範囲内では $W_n = 2W_{n-1} + W_{n-2} - 2$ ($W_1 = 4$, $W_2 = 8$) となることがわかった。

図 4.2は、格子状のグラフでの支配集合の BDD を構成する時のプロセッサの台数と実行時間の関係を示している。この図より、BDD のノード数が多いほど理想的な速度向上がのぞめることがわかる。これは、等価性の判定が 1 つのプロセッサにノードを集めずに行えるからである。

表 2は平面上に点をランダムに n 点発生させ、すべての点の間を枝で結んだグラフでの交グラフでの支配集合を表す BDD を構成する時間とノード数を表す。ここでいう交グラフ I とは、I の頂点はグラフ G の枝に対

表 1: 格子グラフでの BDD の構成時間とノード数

DS を構成する時間とノード数					
		等価性を h_a で判定		等価性を h'_a で判定	
n	変数	時間 (sec)	ノード数	時間 (sec)	ノード数
6	36	0.08	5024	0.07	5332
7	49	0.20	17793	0.11	18623
8	64	0.77	59317	0.27	61507
9	81	3.53	189234	0.97	194947
10	100	17.8	583822	6.16	598618
11	121	89.1	1754380	12.9	1792498
12	144	425.4	5161022	50.3	5258808
13	169	—	—	165.1	15169029

IS を構成する時間とノード数			
n	変数	時間 (sec)	ノード数
9	81	0.06	6012
10	100	0.07	12179
11	121	0.08	24137
12	144	0.11	46975
13	169	0.43	90041
20	400	18.74	6457744
21	441	38.63	11563428
22	484	79.87	20553175
23	529	160.21	36269830

表 2: 交グラフでの BDD の構成時間とノード数

DS を構成する時間とノード数				
	等価性を h_a で判定		等価性を h'_a で判定	
変数	時間 (sec)	ノード数	時間 (sec)	ノード数
28	0.06	338	0.07	838
36	0.10	1417	0.39	5175
45	0.14	5336	0.31	36042
55	0.96	42808	7.69	468296
66	3.92	248582	—	—
78	227.9	5331803	—	—

IS を構成する時間とノード数		
変数	時間 (sec)	ノード数
45	0.05	872
55	0.06	3116
66	0.07	7842
78	0.16	73018
153	14.9	1577631
171	61.4	8300295
190	142.4	11278900

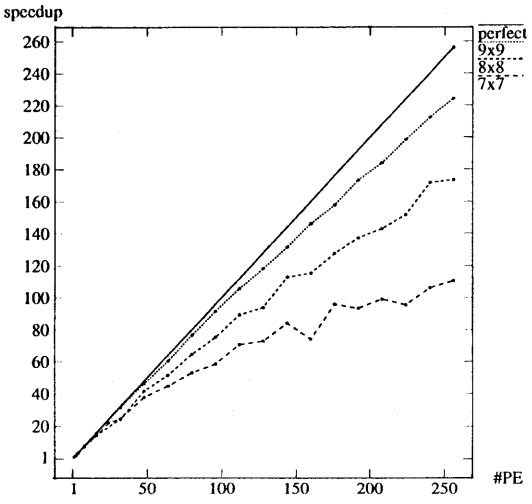

図 1: プロセッサ数とスピードアップ

応し、G の枝同士が交差しているときにその枝に対応する I の頂点を枝で結んだグラフを表し、G の点の三角形分割に対応している。冗長な節をはぶかない場合はノード数が非常に大きくなり、構成時間も長くなる。そして大きなグラフではノード数が多くなり計算ができなかった。これは支配集合を表す論理式の節の中の変数の数が、格子状のグラフでは高々5であるのに対し、交グラフでの支配集合では変数の数程度になることがあるからと思われる。つまり、等価性の判定を h_a によって行えば、等価なノードを複数作らずに BDD を構成できるので、最終的なノードを記憶できるならば必ず BDD を構成できるという利点がある。

5 結論

本稿では、論理関数が正のリテラルのみの和積標準形 (正 CNF) の場合に、その論理関数を表す BDD をトップダウンに構成する方法を示し、それを AP-1000+ 上で実装した。そして、グラフの支配集合と独立点集合を表す関数の BDD について実験を行った。その結果、BDD のノード数が多ければ、プロセッサ数に比例する程度の速度向上がのぞめることがわかった。

正 CNF 式の等価性の判定には冗長な節をはぶく操作が必要だが、節の数が多いと時間がかかることがわかった。そこで冗長な節をはぶかずに BDD を構成し、その後に等価なノードをなくす方法が考えられるが、ノード数が非常に多くなることがあるので、常にこの方法がいいとはいえない。

我々の手法では、正 CNF 式の BDD しか構成できない。任意の論理関数を表す BDD を構成するには、我々の手法で構成した BDD 間の論理演算が行えればよい。つまり、従来の手法での等価性の判定をノードを分散させたまま行うことができればよい。よって、今後の課題としては、冗長な節をはぶく操作の高速化と、従来の手法での等価性の判定の並列化の実装がある。

謝辞

本研究の一部は、文部省科学研究費の援助を受けた。

参考文献

[1] 木村 晋二: BDD の並列処理技術, 情報処理, Vol. 34, No. 5,pp. 624-630 (1993)

[2] R.E.Bryant: "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Trans. on Computers*, vol. C-35(1986), pp.677-691.

[3] K.Hayase, K.Sadakane and S.Tani, "Output-Size Sensitiveness of OBDD Construction Through Maximal Independent Set Problem", in Proc. of COCOON'95, LNCS 959, pp.229-234, 1995.

[4] S.Kimura, T.Igaki, and H.Haneda: "Parallel Binary Decision Diagram Manipulation", *IEICE Transactions on Fundamentals*, vol. E75-A, No. 10 (1992), pp.1255-1262.

[5] S.Tani and H.Imai: "A Reordering Operation for an Ordered Binary Decision Diagram and an Extend Framework for Combinatorics of Graphs", ISAAC'94, LNCS 834 (1994), pp.575-583.

[6] S.Minato, N.Ishiura and S.Yajima: "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation", *Proc. 27th Design Automation Conference* (1990), pp.52-57.