4L−3　共有記憶型並列計算機上での共有二分決定グラフの並列化について*

木村 晋二、羽根田 博正†
神戸大学工学部‡

1　はじめに

論理関数を効率良く表す方法として、近年二分決定グラフ(BDD)が注目され、広く研究されている([1], [4])。多くの実用的な論理関数に対して、非常に効率良く関数を表現できるので、二分決定グラフの処理に関する時間は少ないが、問題によっては、問題のサイズの指数に比例する記憶量と処理時間を必要とする([2])。そこでここでは、二分決定グラフの処理の並列化により演算キャッシュおよび否定枝を有する共有二分決定グラフの構成([4])を高速に行なう手法を提案する。並列化手法は、文献[3]に従っているが、並列性を抽出するときに、[3]の静的な展開に対し、動的な展開を用いている。また、[3]に対して実行時間が約6分の1に短縮された。

2　二分決定グラフの並列構成手法

n 変数論理関数 f は、深さ n の二分木の 2^n 個の葉に、左から関数の値 $f(0, ..., 0, 0)$, $f(0, ..., 0, 1)$, ..., $f(1, ..., 1, 1)$ を対応づけた二分決定木で表される。なお、対応を明確にするために、図1(a)のように、接点に変数名のラベルを、枝に0, 1のラベルを付ける。

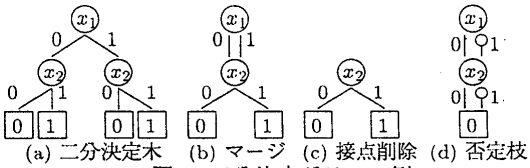
二分決定グラフは、二分決定木の論理的に等価な接点を一つにまとめたものである(図1(b))。また、ある接点 v から出る二つの枝が同じ接点を指している時には、v を短絡除去する。ある論理関数を表す二分決定グラフは、変数の順序を決めれば一意である。

共有二分決定グラフ([4])は、等価な接点のまとめ上げを複数の論理関数に対して行なったものであり、接点数の低減などの利点がある。否定枝([4])は、枝に否定を表す属性(論理値を反転させる)を付加するもので、これも、接点数を減らす上で有効である(図1(d))。演算キャッシュ([1])は、演算の(途中)結果をキャッシュ配列に記憶しておくもので、同じ演算は配列参照で済むため、演算の高速化に有効である。

与えられた論理式の表す二分決定グラフを構成するには、以下の手順を行なう。

1. 恒等的に0である論理関数と、入力変数に対応する二分決定グラフを構成する。
2. 論理式中の(2引数)論理演算を、実行できる順序に従って実行する。

実行順序は、入力変数のみからなる演算をレベル1の演算、入力変数とレベル1の演算結果を用いた演算をレベル2の演算、のように演算のレベルづけを行ない、レベルの小さい順に演算を実行する。実際には、演算を配列に入れておき、インデックスを更新しながら実行する。


(a) 二分決定木　(b) マージ　(c) 接点削除　(d) 否定枝
図1: 二分決定グラフの例

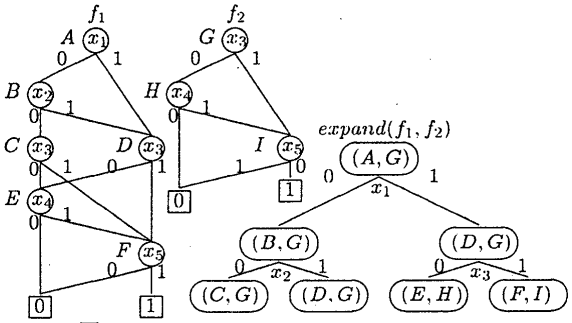


図2: An example of the dynamic expansion.

論理式中の論理演算の中には、同一レベルの演算など並列に実行できるものがあるので、まずはそれらを並列実行する。具体的には、演算の配列とインデックスを共有変数とし、インデックスの更新をCritical Sectionとして演算の実行を行なうプロセスを並列に起動すれば良い。

ただし、この方法で抽出できる並列度は低いので、演算を分割して並列度を上げる。例えば、$f_1(x_1, x_2, x_3, ..., x_n)$ と $f_2(x_1, x_2, x_3, ..., x_n)$ の論理演算を、

1. $f_1(0, x_2, x_3, ..., x_n)$ と $f_2(0, x_2, x_3, ..., x_n)$ の論理演算、
2. $f_1(1, x_2, x_3, ..., x_n)$ と $f_2(1, x_2, x_3, ..., x_n)$ の論理演算、
3. 上記の演算結果から $f_1(x_1, ..., x_n)$ と $f_2(x_1, ..., x_n)$ の論理演算結果を求める、

の3つの部分に分割することにより、1. と 2. の部分は並列に実行できる。同様に

1. $f_1(0, 0, x_3, ..., x_n)$ と $f_2(0, 0, x_3, ..., x_n)$ の論理演算、
2. $f_1(0, 1, x_3, ..., x_n)$ と $f_2(0, 1, x_3, ..., x_n)$ の論理演算、
3. $f_1(1, 0, x_3, ..., x_n)$ と $f_2(1, 0, x_3, ..., x_n)$ の論理演算、
4. $f_1(1, 1, x_3, ..., x_n)$ と $f_2(1, 1, x_3, ..., x_n)$ の論理演算、
5. 上記の演算結果から $f_1(x_1, ..., x_n)$ と $f_2(x_1, ..., x_n)$ の論理演算結果を求める、

とすると、4つの部分を並列に実行できる。以下、展開演算の結果から最終的な結果を求める部分をマージと呼ぶ。

実際は、各論理関数を表す二分決定グラフの接点のラベルにあわせて、図2に示すような動的な展開(4つに

*Parallel Binary Decision Diagram Manipulation on Shared Memory Multi-Processors
†Shinji Kimura and Hiromasa Haneda
‡Faculty of Engineering, Kobe University

表 1: A parallel execution of a 10-bit multiplier.

CPU 台数	1	5	10	15	20	25
CPU 秒	122.4	35.28	18.22	12.53	9.82	8.43
高速率	1	3.47	6.72	9.76	12.47	14.5

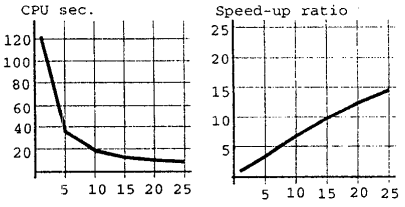


図 3: Experimental result for a 10-bit multiplier.

展開) を行ない、展開結果の対の各々を並列に実行する。適度の並列性を抽出するため、各レベルの演算数に応じて以下のように演算の分割数を決定している。展開自体は動的に行なうが、分割数の決定などは静的に行なわれる。分割された演算は分割の情報とともに演算の配列に入れられ、先ほどのプロセスで並列に実行される。

演算数	～3	～7	～15	～31	～127	～255	その他
分割数	128	64	32	8	4	2	1

本方式は、レベル毎の演算数に合わせて分割数を決定できるという利点の反面、無駄なマージを含むという欠点を持つ。なぜならば、マージした演算結果に対してつぎのレベルで展開することがあるからである。また図 2 の (D, G) のように、冗長な展開を含むという問題も持つ。現在は、冗長な展開かどうかの判定を行なわずに並列実行させており、高速化を阻害する原因になっている。

3 評価結果

以上の方式を、Sequent S-81 (CPU 80386 (16 MHz) × 28, 主記憶 86.75 MB) で実現し評価を行なった。

まず 10 ビットの乗算器 (20 入力, 20 出力, 680 演算 (分割後 8924 演算), 51 レベル, 生成された総接点数 538,753) に対する評価結果を表 1に示す。表中の 1 CPU のときの実行時間は、直列プログラムでの評価である。並列プログラムを 1CPU で実行すると 166.9 秒必要とし、直列プログラムの 122.4 秒と比較すると 1.3 倍時間がかかっている。これは、共有記憶の一貫性保持に必要なロックやアンロックに時間がかかるためである。並列プログラム間の比較では 25 CPU で 20 倍程度の高速化となっているが、直列プログラムとの比較では、14.5 倍程度に留まっている。実行時間と高速化率のグラフを図 3に示す。

つぎに、ISCAS のベンチマークに対して本手法を適用した結果を示す。表 2 にはベンチマークの各データの情報が、また表 3 にはベンチマークの各データに対する実行時間および高速化率が示されている。並列プログラムの 1 CPU での実行時間は、乗算器と同様直列プログラムの 1.3 倍程度である。高速化率については、c1908 で 12 倍程度の高速化を達成しているが、後は 7 倍程度に留まっている。無駄な展開、およびマージと展開の無駄を避ける手法を開発する必要があると考えられる。

表 2: ISCAS benchmark data.

Data	#op (expand)	#level	#nodes Generated
c432	216 (2452)	29	27,531
c499	246 (3294)	15	65,561
c880	435 (5923)	30	527,689
c1355	590 (5558)	28	189,835
c1908	1057 (7413)	44	264,912
c3540	1983 (9947)	54	962,465
c5315	2973 (5101)	52	155,094

表 3: Experimental results for ISCAS benchmarks.

Data	CPU sec. (Speed-up ratio)						
	1		5	10	15	20	25
	single	para					
c432	5.76	7.22	2.5	2.0	1.4	1.2	1.2
	1	—	2.3	2.89	4.11	4.8	4.8
c499	7.85	10.27	2.43	1.72	1.40	1.07	1.06
	1	—	3.23	4.56	5.60	7.34	7.40
c880	48.22	70.45	27.22	19.47	16.77	15.98	15.97
	1	—	1.77	2.47	2.87	3.01	3.01
c1355	18.07	24.52	5.83	3.35	3.02	2.80	2.41
	1	—	3.10	5.40	6.00	6.45	7.50
c1908	51.466	67.05	15.98	8.72	6.13	4.92	4.42
	1	—	3.22	5.90	8.39	10.47	11.65
c3540	159.18	231.47	55.63	34.22	27.18	24.18	22.60
	1	—	2.86	4.65	5.86	6.58	7.04
c5315	14.20	19.37	6.08	4.18	3.35	2.88	2.67
	1	—	2.33	3.39	4.24	4.93	5.33

4 おわりに

本稿では、二分決定グラフの並列処理アルゴリズムおよびその評価を示した。論理演算列中の並列度が十分でないときに動的に演算を分割して並列度を上げる方式を採用している。ISCAS のベンチマークに対して評価を行なったが、良い場合で 12 倍程度、他のものに対しては 7 倍程度の高速化を達成した。今後本文中で述べた並列化の無駄の除去により、さらに高速化を行う必要がある。

謝辞 本研究について助言および議論していただいた カーネギーメロン大学の Edmund M. Clarke 教授に深謝いたします。また、日頃から御討論いただく京都大学 濱口 清治様、京都大学 越智 裕之 様、神戸大学 太田 有三 助教授はじめ羽根田研究室の皆様に感謝いたします。なお、本研究は一部文部省科学研究費 (奨励研究 (A)04750340)、實吉奨学金、兵庫県科学技術振興助成金による。

参考文献

[1] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a bdd package. In *Proc. of 27-th DA Conf.*, pp. 40–45, June 1990.

[2] R. E. Bryant. On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Trans. on Comput.*, C-40(2):205–213, Feb. 1991.

[3] S. Kimura and E. M. Clarke. A parallel algorithm for constructing binary decision diagram. In *Proc. of ICCD '90*, pp. 220–223, Sept. 1990.

[4] 湊, 石浦, 矢島. 論理関数の共有二分決定グラフによる表現とその効率的処理手法. 情報処理学会論文誌, 32(1):77–85, Jan. 1991.