2R−7

Transduction法のベクトル化

齋藤 勉　久木元 裕治　田中 英彦
東京大学 工学部

1 はじめに

計算機による多段論理回路簡単化の一手法として、Transduction法がある。本論文では、このTransduction法をスーパーコンピュータ上で実行するものとし、そのベクトル化を考える。

2 目的

スーパーコンピュータ上でTransduction法を実行することの利点は、その大きな実メモリと、処理速度にある。

一般に、論理合成CADの処理は、論理関数の表現が莫大な記憶領域を必要とし、メモリの大きさで処理できる回路の規模が決定される。近年、Binary Decision Diagram (BDD)の使用で論理関数の表現が比較的小さくなっているものの、いまだ、処理できる回路規模はそのメモリの大きさに制限されており、それはTransduction法においても例外ではない。

そこでこの処理をスーパーコンピュータ上で実行することによって、その大きな実メモリを使っての、より大規模な回路へのTransduction法の適用が可能となり、Transduction法の適用対象規模が増大する。

もちろん現在でも大規模回路に対しては、それを処理できる大きさにまで分割し、分割されたそれぞれの回路を別々に簡単化した後、それらの結果をマージすると言う手法でTransduction法が適用されている。しかしこの方法では、分割された各回路間にまたがるDon't care情報を有効利用できないため、十分な簡単化が行なわれているとは言えない。

そこで、大きな実メモリを持つマシン上でのTransduction法の実行は、大きな意味を持つものと考えられる。

また、処理速度に関しては、現在のスーパーコンピュータにおいてはベクトル化率がその速度に大きく影響するため、アプリケーションにもよるが、例えば京大のBDDパッケージは、東京大学大型計算機センターのHITAC S-820/80で実行した場合、Sun3/60の70倍以上の処理速度が得られている。

3 背景

3.1 ベクトル計算機

一般に現在のスーパーコンピュータでは、ベクトル演算によってその処理を高速化している。ベクトル演算とは、ベクトルに対する演算を一度に行なうもので、その分、通常のスカラ演算に比較して高速に処理を行なうことができる。従ってスーパーコンピュータで処理速度を向上させるためには、いかに全体の処理のうちのベクトル演算の割合(ベクトル化率)を高めるか、いかにベクトルの長さ(ベクトル長)を長くするかが、大きな要因となっている。

3.2 ベクトル化BDD

Transduction法での論理関数表現としてはBDDを用いる。BDDはR. E. Bryantによって提案された論理関数の表現・操作法であるが、ここでは、京大で開発されたベクトル計算機向きのアルゴリズム[2]を使用する。これは、従来深さ優先で行なわれていたグラフの操作を幅優先で行なうことで、処理のベクトル化を行なうものである。

一般にBDDでの二つのグラフに対する演算は、それらのグラフの1のサブグラフ同士の演算と、0のサブグラフ同士の演算の結果から得られる。従来のアルゴリズムでは、この演算を深さ優先で再帰的に行なっていた。

一方[2]では、まず、expansion phaseとして、二つのグラフに対する演算の結果として仮のノード(temporal node)を作成してしまい、そのノードの1のサブグラフと0のサブグラフを作成する演算をqueueに登録する。以下、この処理をqueueが空になるまで続ける。次にreduction phaseとして、expansion phaseで作成されたtemporal nodeについて、不要なノードを取り除くなどの正規化を行なう。

このようにqueueを用いた幅優先アルゴリズムを用いることで、処理のベクトル化率をあげており、従来の手法に比べて3倍から16倍という高いベクトル化率が得られている。

3.3 Transduction法

Transduction法[1]は、1970年代にイリノイ大学で開発された多段論理簡単化手法で、回路の変形(Transformation)と冗長部分の削除(Reduction)を繰り返し行ない、多段論理を簡単化するものである。このとき、回路の各ゲートや接続ごとに、許容関数と呼ばれる一種のDon't Care条件を計算し、その関数によって許された範囲で論理の変更(回路の変形・削除)を行なう。[1]では、論理関数を真理値表を用いて表現しているが、BDDを用いることで、より効率的に処理を実行できることが[3]に示されている。

Transformation法の主な処理としては、次のものがある。

冗長な部分の削除　ゲート・接続の許容関数がON-SETやOFF-SETを全く含まない場合、そのゲート・接続を冗長なものとして削除し、定数関数に置き換える。

接続の追加・削除　冗長な部分が存在しない場合、ゲートに新たな接続を追加することで故意に冗長な部分をつくり出し、その後再び回路を簡単化することで、回路の構造を変化させる。新たな接続がそのゲートの出力に影響を及ぼすかどうかは、許容関数を用いて判断する。

Vectorization of Transduction Method on Super Computer
Tsutomu SAITOH,Yuji KUKIMOTO,Hidehiko TANAKA
the University of Tokyo

4 Transduction 法のベクトル化

3.3でも述べたように、Transduction 法での処理は、以下のようなものが主となる。

- 回路の出力関数の計算
- 各ゲート・接続の許容関数の計算
- 許容関数と出力関数の包含関係の check

ここでは、Transduction 法をスーパーコンピュータ上で行なうため、以下のようなベクトル化を考える。

4.1 各ゲート・接続の出力関数・許容関数の計算

回路の出力関数や各ゲート・接続の許容関数の計算は、基本的には BDD の演算であるため、幅優先アルゴリズムを使用することでベクトル化が可能である。

さらにこのアルゴリズムを、複数の BDD 演算を同時に実行できるように拡張することにより、よりベクトル長を長くしてベクトル化率を高め、高速に実行できるようにすることが可能である。[2] では、トップレベルでは一度に一つの演算しか行なわないため、演算初期のベクトル長が非常に短い。これを、トップレベルで複数の演算を queue に入れることができるように拡張することで、演算の初期段階から長いベクトル長を確保できることになり、処理時間の短縮が図れる。

しかしその一方で、複数の BDD 演算を同時に実行することは、temporal node がそれだけ同時に生成されることとなり、メモリ使用量が増大することになる。このため、回路規模が大きい場合、メモリ不足を招く結果にもなりかねない。したがって、この拡張を行なうには、実行時間と使用メモリ量のトレードオフを考える必要がある。

4.2 包含関係 check のベクトル化

論理関数の包含関係の check は、Don't care を含む論理関数を二つとり、真か偽かを返す処理で、BDD では直接グラフを操作することなく行なうことができる。この処理は図 1 のアルゴリズムを用いてベクトル化が可能である。

なお、BDD で Don't care を扱う方法は、3 値 BDD に拡張するか、2 値のグラフを二つ持つことで Don't care を表すかの 2 種類に大別されるが、ここでは 3 値 BDD を用いている。

またこの処理も、BDD 演算のように、複数の check を同時に実行できるように拡張可能である。このとき、ある処理が偽となった場合は、以降 queue からタプルを取り出す際に、その処理を示しているタプルは実行しないように、フィルタリングを行なうと良いであろう。

4.3 connectability の計算

上に述べた手法を用いることによって、Transduction 法における connectability の計算がベクトル化できることになる。ここで connectability とは、冗長な接続を付加する時にそこに付加できるかどうか、ということである。

ゲート間で冗長な接続を付加できる可能性があるのは、その接続により cyclic な回路ができない時に限る。従って、あるゲート G に注目して、その入力に接続をつなごうとした場合、G のトランジティブ・ファンアウト以外のすべてのゲートについて、その出力を G の入力に接続できる可能性があるわけである。これらの接続が可能かどうかは、前にも述べたように、接続後の

check したい包含関係 $f \subseteq g$ に対して、f, g をそれぞれ指すグラフを考えたとき、

1. タプル (f, g) を queue に入れる。
2. queue からタプルを一つとり出す。これを (p, q) とする。
3. p と q が同じグラフの場合、包含関係が成り立つので 6 にいく。また、互いに他方の否定になっている場合は包含関係が成り立たないので、この $f \subseteq g$ は偽となり、処理を終了する。
4. 両方のグラフが 1、*、または 0 の場合、図 2の真理値から、包含関係が成り立つ場合は 6 にいく。成り立たない場合は、この $f \subseteq g$ は偽となり、処理を終了する。
5. そうでない場合、(p_1, q_1) および (p_0, q_0) を queue にいれる。ただし、p_1、p_0 はそれぞれ p の 1 のサブグラフ、0 のサブグラフを表している。
6. 2 〜 5 を queue にタプルがなくなるまで繰り返す。
7. $f \subseteq g$ は真であるとして、処理を終了する。

図 1: 包含関係 check のアルゴリズム

		p	
		1	0
q	1	1	0
	*	1	1
	0	0	1

図 2: $p \subseteq q$ の trivial な場合の真理値

G の出力関数が、そのゲートの許容関数に包含されているかどうかを調べれば良く、上記の手法を使えば、これらの check がまとめてベクトル化可能である。

5 終りに

多段論理簡単化手法である Transduction 法を、スーパーコンピュータ上で実行する時のベクトル化について考察した。

今後は、実際にベクトル化 BDD が動いている、東京大学大型計算機センターの HITAC S-820/80 上に処理系を実装し、ベクトル化手法と実行時間・メモリ使用量などの関係を調べていく予定である。

6 謝辞

ベクトル化 BDD パッケージを提供して下さった、京都大学の越智裕之氏および大阪大学の石浦菜岐佐氏に深く感謝の意を表します。

参考文献

[1] Muroga S., Kambayashi Y., Lai H. C. and Culliney J. N.: The Transduction Method – Design of Logic Network Based on Permissible Functions: In *IEEE Trans. Comput*, Vol. 38, No. 10, pp. 1404–1424, (Oct. 1989).

[2] Ochi H., Ishiura N. and Yajima S.: A Breadth-First Manipulation of SBDD of Boolean Functions for Vector Processing: In *Proc. DAC '91* , pp. 413–416, 1991.

[3] Matsunaga Y. and Fujita M.: Multi-level Logic Optimization Using Binary Decision Diagrams: In *Proc. ICCAD '89* , pp. 556–559, (Nov. 1989).