設 計 自 動 化 77－12
(1995. 10. 20)

トランスダクション法の並列化に関する研究

松本 太　　木村 晋二　　渡邊 勝正

奈良先端科学技術大学院大学 情報科学研究科
〒630–01 奈良県生駒市高山町 8916-5
TEL: 07437-2-5301　　FAX: 07437-2-5309
E-mail: {futosi-m, kimura, watanabe}@is.aist-nara.ac.jp

あらまし　本稿では、並列 BDD 処理手法を基にした、トランスダクション法の並列化手法を提案する。トランスダクション法では、ゲート間の接続の自由度を表す関数の集合 (許容関数) を用いて外部出力に関して等価な変換ができるかどうかの判定を行ない、回路の外部出力等価な変換を繰り返すことによって回路の簡単化を行なっている。本稿では、トランスダクション法のうち代表的な変換手法である Connectable/Disconnectable 法を中心に、トランスダクション法の並列化手法を示す。

キーワード トランスダクション法、許容関数、BDD、並列化 BDD、並列トランスダクション法、論理合成

A Parallel Transduction Method using Parallel BDD Manipulation

Futoshi Matsumoto, Shinji Kimura and Katsumasa Watanabe

Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630–01 JAPAN
TEL: +81-7437-2-5301　　FAX: +81-7437-2-5309
E-mail: {futosi-m, kimura, watanabe}@is.aist-nara.ac.jp

Abstract　We propose a parallel transduction method based on parallel BDD manipulation for shared memory multi-processor systems. The transduction method is a kind of logic optimization method using the transformation and the reduction of logic circuits. In these operations, permissible functions are used. We consider the parallel algorithm to compute the permissible functions of logic gates, and parallel algorithms to transform and reduce the circuit. We focus on transduction methods based on the connectable/disconnectable method.

key words　Transduction Method, Permissible Function, BDD, Parallel BDD Manipulation, Parallel Transduction, Logic Synthesis

1 はじめに

VLSI技術の進歩にともない、人手設計が困難であるような大規模回路を計算機支援によって自動設計する手法が重要になっている。またハードウェアについてあまり詳しくない人でも、プログラミング能力があればハードウェア記述言語によって回路を作成することが容易になりつつある。

ハードウェア記述言語による記述は、論理合成手法により論理回路に自動的に変換されるが、簡単な論理回路の合成で数百秒程度、複雑なものになると数千秒程度と、合成に非常に多くの時間を必要とする。処理の主たる部分は、自動合成で生成される回路が人手設計に比較してかなり無駄の多いものであるため、生成された回路の最適化に費やされる。合成の品質を人手設計に近付けようとすればするほど多くの最適化処理を必要とし、高速な最適化手法の開発は重要な問題である。

そこで、本稿では、論理回路の最適化に広く用いられているトランスダクション法 ([1], [2], [3]) に着目し、トランスダクション法の並列化に関する研究を行なう。トランスダクション法では、許容関数とよばれる概念を用い、回路の変形を繰り返すことによって最適化を行なう。変形においては回路中のすべてのゲートの全数検査に近い処理が行なわれ、ゲート数の 3 乗に比例した程度の計算量を必要とする。このため、並列化の効果は非常に高いと考えられ、これまでにも文献 [4] などで並列化手法が提案さている。しかし、[4] での並列化手法では、比較的粗いレベルでの並列化を行なっているため、シングルプロセッサでの実行時間が 1500 秒から 6000 秒程度と非常に大きいにもかかわらず、回路によっては 8 プロセッサで 5 倍程度と十分な並列性を抽出できていない。

本稿では、トランスダクション法で必要な論理関数処理を並列化二分決定グラフ処理手法で行なう並列化手法を提案する。本手法では、[4] の手法に比較してより細かいレベルで並列性を抽出できる可能性がある。本稿では主に [2] に従い、Gate Merge と Connectable/Disconnectable を用いたトランスダクション法の並列化について述べる。

2 BDD とその並列処理

2.1 BDD

BDD (Binary Decision Diagram, 二分決定グラフ, [5]) は、論理関数を表現する手法の一つであり、定数ノード、変数ノード、0 枝、1 枝からなる有向グラフである。定数ノードは 0, 1 の二つがあり、各々恒等的に 0, 1 である論理関数を表す。また変数ノードは、論理変数をラベルとして持ち、0 枝と 1 枝の二つの有向辺を有する。BDD のあるノード (変数ノードあるいは定数ノード) を根とする部分グラフは論理関数を表す。ノード v の表す論理関数を f_v 、v のラベルを x_v、v の 0 枝および 1 枝が指す部分グラフが表す論理関数を f_{v0}, f_{v1} とすると、$f_v = \overline{x_v} \cdot f_{v0} + x_v \cdot f_{v1}$ の関係が成立する。すなわち、0 枝が指す部分グラフは x_v が 0 という値をとる場合の論理関数を表し、1 枝が指す部分グラフは x_v が 1 という値をとる場合の論理関数を表す。

Shannon の展開定理からわかるように、BDD は、変数の順序を固定すると論理関数固有のカノニカルな表現になり、論理回路が実現する論理関数の等価性判定などに用いることができる。

論理回路が表す論理関数に対する BDD は、以下のように構成される。まず、入力変数からのレベルを定義する。

$$level\_in(g) = \begin{cases} 0 & g \text{ が外部入力の時} \\ (\text{入力に結合されているゲートのレベルの最大値}) + 1 & \text{上記以外} \end{cases}$$

つぎに、レベルの小さい順にゲートの順番を決める。同じレベルの場合にはどのように決めても良い。外部入力に対する BDD を与え、上記で決めた順番にゲートに対応する論理演算を BDD で行なえば、各ゲートに対応する論理関数を表す BDD が構成できる。

2.2 BDD の並列処理

論理回路に対応する BDD を構成する処理の並列化については、[6] などで研究が行なわれている。並列化の基本は、レベルに基づくゲートの評価において、同じレベルにあるゲートの評価や、直接依存関係が存在しない異なるレベルにあるゲートの評価を並列に行なうことである。ただし、このような回路内部にある並列度だけでは十分な並列性の抽出が困難であるので、[6] では、各ゲートに対応する論理演算の BDD 上での実行で、Shannon 展開を BDD の構造に即して行なうことでより高い並列度を抽出する動的 Shannon 展開手法を提案している。通常の BDD では、ある変数で展開した結果の論理関数は次の順序の変数に依存しない場合があるので、実際に依存している変数で、決められた深さだけ展開することで、一つの論理演算を複数の論理演算へ分割して並列実行する手法である。展開の深さは、レベル毎の演算の数を目安にして決められ、演算が少ない場合は展開する深さを深くする。

[6] の手法では、動的 Shannon 展開手法において、実行するべき演算を展開した後の形で陽に持つために、演算の数がシングルプロセサでの実行に比較して 10 倍程度になってしまうという問題点があった。ここではそれを改良して、展開および展開結果のマージ用に変数を用意して、演算実行の並列プロセスがこの変数をロック付きで一つずつ減らして行くことで、どの展開を行なっているかをわかるようにすることで、演算を記憶するための容量の問題点を回避している。

3 トランスダクション法

トランスダクション法は、論理回路最適化手法の一つであり、許容関数と呼ばれる概念を用いて回路を変形する。本章では、[2] に基づき、本稿で用いるトランスダクション法の概念について述べる。

3.1 許容関数

回路中のある場所 (ゲートの入力あるいは出力) v における論理関数 f を他の論理関数 f' に置き換えても回路の外部出力の論理関数が変化しないとき、f' を v における許容関数と呼ぶ。

例として、回路の中に 2 入力 AND ゲートがあり、ある外部入力値に対するそのゲートの出力が 1 であるとする。この場合、AND ゲートの入力の値は、すべて 1 でなければならない。一方、ある外部入力値に対するゲートの出力が 0 である場合には、入力のいずれかが 0 であれば良く、実際に回路としてはそのようになっているはずである。この時、0 となっている入力以外の入力がどのような値になっているかは回路に依存するが、AND ゲートの性質からは、0 でも 1 でも良い。すなわち、回路構成に自由度があることになる。このような、0 でも 1 でもよい場合を * で表し、ドントケアと呼ぶ。この例のように、ゲートの入力の許容関数は、他方の入力の値に依存して決まる。また、この例が一つの外部入力値に対してであることに注意する必要がある。

ある場所における許容関数は複数存在し、一般に集合で表される。なお、許容関数集合は、0, 1, * の 3 値の論理関数で表せることが知られている。* により、0 を入れた結果の論理関数と、1 を入れた結果の論理関数の集合を表す。以下では許容関数集合とそれを表す関数とを同一視する。

許容関数の集合のとりかたには **MSPF** (Maximum Set of Permissible Functions) および **CSPF** (Compatible Set of Permissible Functions) がある。

MSPF は、すべての許容関数からなる集合である。ただし、ゲートの入力の許容関数が他の入力に依存して決まることから、一箇所の回路変更が他の MSPF に影響を与えるので、一箇所変更する毎に MSPF を計算しなおさなければならない。

一方、CSPF では、回路の複数の場所で同時に変形を行うことが可能となるように、すべての許容関数集合の部分集合をとる。具体的にはゲートの入力に順序をつけて、順序の先のものが出力を駆動するという条件の下で、許容関数集合を計算

する。

先の 2 入力 AND ゲートの例でいえば、出力 0 に対して、現在の入力がどちらも 0 であった場合、MSPF では許容関数が両方とも * となるが、CSPF では、入力に順序をつけ、片方 (順序の先の方) の許容関数が 0 に固定され、もう一方が* となる。MSPF では両方とも * なので、(1, 1) を選ぼうとすると回路の関数が変わってしまうため一箇所しか変更できない。一方、CSPF では (1, 1) は選べないようになっている。

CSPF を用いた場合、MSPF と比較して制限が厳しいため、MSPF では行なうことができた回路の変形が不可能と判断されることがある。一方で、MSPF ではできない複数箇所の同時変更が可能となり、回路変更の高速化に有利である。以降では、許容関数集合としては CSPF を用いることとする。

CSPF は以下のように計算される。まず、回路の外部出力の CSPF は、外部出力の論理関数一つだけからなる集合である。つぎに、ゲートの出力の CSPF とゲートの入力の論理関数がわかっていると、ゲートの各入力の CSPF を以下のようにして求めることができる。

まずゲートの入力に順序をつけておき、出力の CSPF と自分の論理関数とを比較して、自分が駆動している部分はそのままに、他で駆動されている部分は * に変更する。駆動の条件はゲートの種類によるが、AND ゲートでは、出力が 1 の場合にはすべての入力で 1 でなければ出力を駆動できず、出力が 0 の場合は入力の一つだけが 0 であれば駆動できる。出力が * ならば、すべての入力も * にできる。OR ゲートは AND ゲートの双対であり、NOT ゲートは出力の CSPF の値が*でない限り入力が * となることはない。さらに、あるゲートの出力のファンアウト数が 2 以上である場合、出力の許容関数はすべての出力先の許容関数の * を入れた AND 演算で求められる。

3.2 枝刈り

枝刈りは、回路の冗長部分を削除する処理である。例えば、AND ゲートの入力の CSPF が 1 と * のみになってしまった場合や、OR ゲートの入力の CSPF が 0 と * のみになってしまった場合は、それらの CSPF の中からすべてが 1 あるいはすべてが 0 である関数を選ぶことで、入力そのものを削除することができる。また、この処理によって一入力のゲートになれば、ゲート自体を削除し、入力を出力に直接つなぐこともできる。また、配線の削除により、外部出力に経路を持たないゲートが発生した場合、そのゲートも削除可能である。

実際に削除を行なうには、CSPF のみが必要である。CSPF の値が 0 と * のみ、1 と * のみのときはそれぞれ恒等的に 0 である論理関数と恒等的に 1 である論理関数に置き換え、不要な配線は削除する。その結果出力が使われなくなったゲート自体を削除し、そのゲートの入力につながる部分回路も不要であれば削除する。

この結果得られた回路については BDD および CSPF の再計算が必要である。

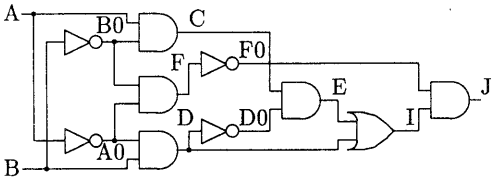
3.3 Connectable/Disconnectable 法

Connectable/Disconnectable 法とは、冗長な配線を加えることにより新たに不要となった別の配線を削除することで回路を変換し、最適な回路に近づける手法である。具体的には、あるゲート g_i に着目し、g_i が削除できるように配線の追加や削除を行なう。

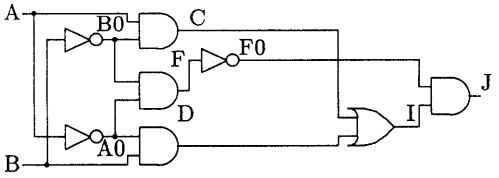
配線の追加に当たっては、以下の条件を満たしていることが必要である。

- 配線の追加によって、回路内にループが構成されない。
- 追加する配線の論理関数と追加先のゲートの出力の CSPF を見て、配線の追加によって出力の CSPF が影響を受けない。例えば AND ゲートでは、出力の CSPF が 1 の部分ではかならず 1 になっている論理関数を持つ配線は入力に追加できる。

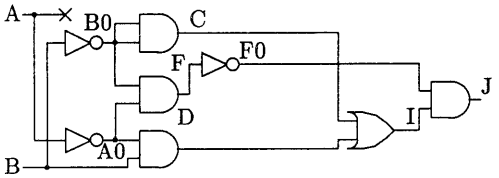
この手法を用いることにより直接 g_i が消されれば回路が簡単になる他、g_i が消えない場合でも、他の最適化が適用できる可能性がある。



(a) Before Reduction.



(b) Apply Reduction.



(c) Apply Connectable/Disconnectable.

図 1: トランスダクション法の適用例

図1にトランスダクション法の適用例を示す。図1(a) が回路の初期状態である。許容関数を計算した結果、D0 の許容関数の値が 1 と * のみとなるため NOT ゲートが削除され、AND ゲートも 1 入力となり削除される。この枝刈りを適用した結果が 図1(b) である。また、C を出力する AND ゲートにおいて入力 A は B0 で置き換えることができる。この Connectable/Disconnectable 法に基づく変換を行なった結果が 図 1(c) である。この後、ゲート C を削除する等の変形も可能である。

4 トランスダクション法の並列化

トランスダクション法では、先に示した変換を順次繰り返して行くことにより、回路の簡単化を行なう。Connectable/Disconnectable で探索する空間は一応回路全体と考えられるので、各ゲート毎に回路のゲート数に比例した計算を必要とする。以下では繰り返しの各部分の並列化について述べた後、全体の並列アルゴリズムについて述べる。

4.1 レベル付けの並列化

ゲートの出力の論理関数および CSPF の計算では、順序の決定に対して、入力あるいは出力からのレベルを用いる。これは、同じレベルにある演算は基本的に独立であり、並列に実行することができるためである。なお、レベルが異なる場合でも直接の順序関係がないものは並列に実行できるので、レベルを基準にして、同じレベルのものは順序を適当につけておき、その全順序のリストを先頭からたどることでそのような並列度も抽出して用いる。

ここでは、レベル付け処理の並列化について述べる。まず、入力側からのレベル付けは 2.1 に定めた *level_in* に基づき、以下のように行なわれる。

1. 外部入力とのみ接続されているゲートのレベルを 0 とし、レベル付けキューに加える。
2. レベル付けキューからゲートをとりだし、出力側のゲートへレベルを伝播する。出力先のゲートのすべての入力についてレベルが定まっていれば、そのゲートのレベルが決まる。レベルの決定したゲートはレベル付けキューに加える。

また、上記のレベルに基づく順序を逆に用いれば許容関数の計算ができるが、これでは十分な並列度が得られないので、並列度を高めるため、許容関数の計算では、以下に定義されるような出力からのレベルを用いる。外部入力からのゲート g のレベルを $level\_in(g)$、外部出力からのゲート g のレベルを $level\_out(g)$ と記述する。

$$level\_out(g) = \begin{cases} 0 & g \text{ が外部出力のみ結合の時} \\ (\text{出力に結合されているゲートのレベルの最大値}) + 1 & \text{上記以外} \end{cases}$$

出力側からのレベル付けの場合、

1. 外部出力とのみ接続されているゲートのレベルを 0 とし、レベル付けキューに加える。

2. レベル付けキューからゲートをとりだし、入力側のゲートへレベルを伝播する。入力側のゲートのすべてのファンアウト先のレベルが定まっていれば、そのゲートのレベルが決まる。レベルの決定したゲートはレベル付けキューに加える。

各プロセッサは、キューからゲートをとりだし、処理を行なう。レベル付けの終了はキューが空になり、かつレベル付け処理中のプロセッサがなくなったときである。

4.2 許容関数の計算の並列化

許容関数の計算は外部出力側から行なわれ、外部入力側へと進んでいく。ゲートの出力の許容関数が求まらねばそのゲートの入力の許容関数が計算できないため、出力の許容関数があらかじめ計算されていることが必要である。

許容関数の計算を並列に行なうという点からは、並列に実行できる計算を増やすため、回路の外部入力からのレベル *level_in* を用いるのではなく、回路の外部出力側からのレベル *level_out* を用いる。

論理関数の計算と同様、レベルの小さい順にゲートの順番を決める。同じレベルの場合にはどのように決めても良い。外部出力に対する許容関数は論理関数そのものであるから、そのままコピーする。レベルが 0 でないものは、上記で決めた順番にゲートに対応する許容関数の計算を行なえば、各ゲートの入力に対応する許容関数が求められる。

ゲートの出力が複数のゲートにファンアウトしている場合には、ファンアウト側のゲートの入力の許容関数の AND を取ればよい。すなわち、出力の許容関数の初期状態をすべてドントケアとしておき、つながっている入力の許容関数がすべて求まった時点で AND 演算を行なうことで許容関数が求められる。この処理は排他制御ができていれば並列実行しても問題はない。

上記の処理は外部出力からのレベルを用いることを除いて、論理関数の処理と同じであり、並列 BDD の処理手法が応用できる。

4.3 簡単化手法の並列処理

4.3.1 枝刈り

枝刈りできるかどうかは、許容関数を計算した時点で判明している。したがって、実際に必要な処理は、削除の処理である。また、削除の結果としてゲートが不要となることがあるので、その場合の処理も行なう必要がある。

すなわち、ある正論理の二入力ゲート A の片方の結線が削除可能なとき、もう一方の入力 i を直接出力につなぐことができる。この結果、このゲートの出力があるゲートの入力につながっているとき、その入力には i を直接つなぐことができる。

枝刈り処理では、ゲートの削除の結果が出力側へ伝搬されるので、この処理は入力側からのレベル付けに基づいて行なう。以下、二入力ゲートを例として述べる。

ある二入力ゲート G の入力を i_1, i_2、出力を o とする。i_2が削除可能なとき、o に i_1 を直接つないでおく。入力側からのレベル付けに基づく順序で行なっているため、この処理を行なう時点では、o を入力に持つゲートはまだ処理されていない。後に o を入力に持つゲートが現れたとき、o に i_1 が直接つながっていることを確認し、o のかわりに i_1 を入力とする。回路すべてについて処理が終了すれば、o を入力に持つゲートは存在せず、ゲート G が削除可能となる。

上記の処理は、各レベル毎で並列に処理を行なうことができる。削除されるゲートについては o に i_1 を直接つなぐときにゲートにマークしておき、枝刈りの最後の処理として取り除く。

4.3.2 Connectable/Disconnectable 法

Connectable/Disconnectable 法では、あるゲートの入力の許容関数とその他のゲートの出力の論理関数の比較という操作が繰り返される。そこで、一つのゲートごとに一つのプロセッサが処理を担当し、そのゲートの入力の許容関数とその他のゲートの出力の論理関数とを比較する。通常の場合ゲート数の方がプロセッサ数よりも多いため、プロセッサは一つのゲートの処理が終われば、まだ

処理されていない別のゲートの処理に移る。ゲートの評価順序は、外部出力からのレベルに基づくものを用いる。

CSPF を用いているため、あるゲートで接続の変更があった場合、その変更の影響でそれ以外の変更が許されなくなる可能性のあるゲートは新たに接続された、あるいは接続を切られたゲートから入力側へたどったときに含まれるゲートのみである。

それ以外のゲートについてはその接続の変更と同時に変更が行なわれても全体としての出力結果は保証される。したがって、以下のアルゴリズムで、複数箇所の同時変更が可能である。

1. 各プロセッサで処理すべきゲートを取得する。
2. 割り当てられたゲートの入力に他のゲートの出力を接続できるかどうか調べる。
3. 接続の変更があった場合、その接続で影響を受けるゲートについては計算を止める。
4. 割り当てられたゲートの処理が終わった、あるいは 3. で計算を止めたプロセッサにはまだ処理されていないゲートを割り当て、2. へ戻る。

上記の回路変換の結果として不要となったゲートの削除も行なわれる。すなわち、接続を切られたゲートが結果的に外部出力への経路を持たなくなった場合、そのゲートは取り除かれる。このようなゲートは接続を切られたときにマークされ、以降は接続変更の比較の対象から除かれる。その後すべてのゲートの処理が終わったときに実際に削除される。

4.4 並列トランスダクションアルゴリズム

以上の並列化手法をまとめると、以下のようなアルゴリズムになる。

1. 外部入力からのレベルおよび外部出力からのレベルを並列計算する。
2. 回路の論理関数を並列 BDD を用いて計算する。
3. 許容関数を並列計算する。許容関数も BDD で表すこととし、計算には並列 BDD 処理を用いる。
4. 枝刈りを行なう。枝刈りが行なわれた場合は 1. へ行く。
5. Connectable/Disconnectable 法を適用する。回路の変更が行なわれた場合は 1. へ行く。

5 おわりに

本稿ではトランスダクション法の並列化について考察を行なった。本手法は、BDD を用いた論理関数処理に基づいており、並列 BDD 処理手法をベースにしている。

論理関数および許容関数の計算は、処理を外部入力あるいは外部出力からのレベルに基づく順序で並べておくことで、並列 BDD の処理手法を用いて行なうことができる。一方、枝刈りも、枝刈り処理を行なう枝を外部入力からのレベルを基にした順序で並べておいて、並列処理する。最後に Connectable/Disconnectable については、複数のゲートについて、そのゲートを中心とした Connectable/Disconnectable の処理を並列に行なうこととし、処理間で不整合があった時にいずれか最初に処理を終了したものが他の関係する処理を停止することとした。

今後はアルゴリズムの実装を行なって評価するとともに、より高速なアルゴリズムの開発を行ないたいと考えている。

謝辞

日頃から御討論いただく、本学國島 丈生助手、高木 一義助手はじめ渡邊研究室の皆様に感謝します。本研究は一部文部省科学研究費補助金による。

参考文献

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

[2] 藤田昌宏. トランスダクション法に基づく多段論理回路簡単化機能をもつ論理合成システムとその評価. 情報処理学会論文誌, Vol. 30, No. 5, pp. 613–623, May 1989.

[3] 藤田昌宏. BDD の CAD への応用. 情報処理, Vol. 34, No. 5, pp. 609–616, May 1993.

[4] P. Banerjee. *PARALLEL ALGORITHMS FOR VLSI COMPUTER-AIDED DESIGN.* Prentice Hall, 1994.

[5] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. on Comput.*, Vol. C-35, No. 8, pp. 677–691, Aug. 1986.

[6] S. Kimura, T. Igaki, and H. Haneda. Parallel Binary Decision Diagram Manipulation. *IEICE Transactions on Fundamentals*, Vol. E75-A, No. 10, pp. 1255–1262, Oct. 1992.