複数コア/FPGA チップ間の通信構造に配慮したアルゴリズムの自動合成

宮坂幸雄[†1]　藤田昌宏[†2]

概要：複数コアまたは複数 FPGA チップからなるシステムを使用するためには、プログラムや回路を分割する必要がある。しかし、通信のオーバーヘッドが並列計算による性能の向上を妨げてしまうことがあり、問題となっている。本論文は、行う計算の入出力の関係とコア間/FPGA チップ間の通信構造から、その通信構造上で効率的に計算を行うアルゴリズムの自動合成を提案する。その問題は Quantified Boolean Formula(QBF)として定式化でき、SAT ソルバーを繰り返し用いることで解くことができる。行列ベクトル積の計算に関して実験を行い、通信が計算時間に影響を与えないようなアルゴリズムを複数得ることができた。そして、それらのアルゴリズムの一般化について考察を行なった。

キーワード：複数コア、複数 FPGA、アルゴリズム合成、部分合成

1. はじめに

近年では、複数コアや複数の FPGA を用いて並列計算を行う機会が増えている。それは、1 コアの処理能力に限界があり、また FPGA チップの LUT の数や RAM の容量が限られているからである。

複数コア/FPGA を使うためには、行うべき計算処理を分割しなければいけない。それに関して多くの研究が行われており、プログラムの分割に関しては、ループ処理を分割する手法[1]、コードを複数のブロックに分割する手法[2]が研究されている。回路の分割に関しては、ネットリストの分割[3]、RTL での分割[4]、高位合成の際に分割された回路を合成する[5]という研究が行われている。

しかし、効率的な並列計算を行うためには、計算アルゴリズムを計算機の構造に適したものに変える必要がある。例えば、行列ベクトル積の計算を行うときに、計算の順序を変えることで効率的に計算が行えるということが提案されている[6]。

我々が提案するのは、計算の入出力の仕様と計算機の構造から、効率的な並列計算を行うアルゴリズムを自動合成するということである。もちろん、入出力の数が増えると自動合成は難しくなるが、計算を一般化し入出力の数を減らしたもので自動合成を行い、それで得られたアルゴリズムを元の計算に使えるように人手で変形するということは可能であると思われる。例えば行列ベクトル積の計算であれば、行列の大きさが 4×4 である場合のアルゴリズムを自動合成し、それを参考に行列の大きさが n×n の場合のアルゴリズムを人が考えるということである。

2. 原理

本論文では部分合成[7]を用いて合成を行なった。部分合成とは、空欄のある回路に対し、その空欄を埋めることで仕様を満たすような回路を合成するという手法である。

提案手法では、空欄はマルチプレクサと LUT (Look Up Table) によって表される。本論文では便宜上、複数の出力を持ったマルチプレクサを扱うが、それはそれぞれの出力に対し通常のマルチプレクサが一つずつ用意されていることを意味している。また、全てのマルチプレクサの入力に定数 0 が暗黙的に含まれていることとする。

空欄を埋めるということは、マルチプレクサの選択信号と LUT の関数に適切な値を与えるということである。これは、すべての入力パターンに対して回路が仕様を満たすような選択信号及び関数の与え方があるか、という QBF (Quantified Boolean Formula) として定式化される。QBF ソルバーは網羅的に解を探すことから、解が得られなかった時は解が存在しないということが証明されたことになる。

3. 問題例

ここでは提案手法の目的を説明するために、4×4 行列に対する行列ベクトル積の計算を扱う。その計算式を(1)に示す。

この式をリング状に接続された 4 つの FPGA チップで計算することを考える。通常であれば図 1 のように、(1)を展開した式である(2)を 1 行ずつそれぞれのチップに割り当て、左側の積から計算すると思われる。しかし、この方法では Is1-Is4 をブロードキャストする必要があるが、その通信はこの構造では 2 サイクル以上を必要とするため、計算のオーバーヘッドとなり得る。

図 2 で示される計算方法[6]では、通信が隣接するチップ間に限られているため、図 1 の計算方法よりオーバーヘッドになりにくい。それぞれの積のチップへの割り当て方、またその積を計算する順番を変えることによって、チップ間

†1 東京大学大学院工学系研究科電気系工学専攻
†2 東京大学大規模集積システム設計教育研究センター

ⓒ2018 Information Processing Society of Japan

$$\begin{pmatrix} I_{stim}1 \\ I_{stim}2 \\ I_{stim}3 \\ I_{stim}4 \end{pmatrix} = \begin{pmatrix} w11 & w12 & w13 & w14 \\ w21 & w22 & w23 & w24 \\ w31 & w32 & w33 & w34 \\ w41 & w42 & w43 & w44 \end{pmatrix} \cdot \begin{pmatrix} I_s1 \\ I_s2 \\ I_s3 \\ I_s4 \end{pmatrix} \quad (1)$$

$$\begin{cases} \mathrm{I_{stim}}1 = w11 \cdot I_s1 + w12 \cdot I_s2 + w13 \cdot I_s3 + w14 \cdot I_s4 \\ \mathrm{I_{stim}}2 = w21 \cdot I_s1 + w22 \cdot I_s2 + w23 \cdot I_s3 + w24 \cdot I_s4 \\ \mathrm{I_{stim}}3 = w31 \cdot I_s1 + w32 \cdot I_s2 + w33 \cdot I_s3 + w34 \cdot I_s4 \\ \mathrm{I_{stim}}4 = w41 \cdot I_s1 + w42 \cdot I_s2 + w43 \cdot I_s3 + w44 \cdot I_s4 \end{cases} \quad (2)$$

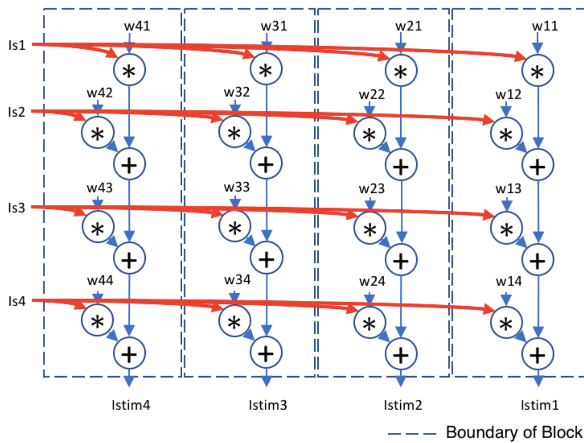

図 1 通常の分割

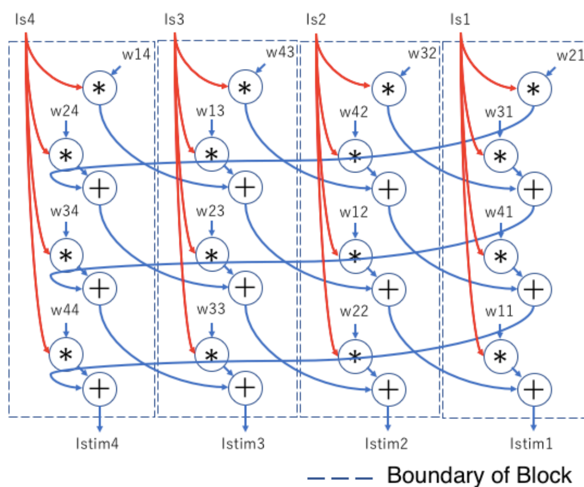

図 2 特殊な分割

の通信を隣接するチップ間に制限している。

4. 提案手法

4.1 提案手法の流れ

提案手法の流れを図 3 に示す。ここでは、与えられた通信構造に適したアルゴリズムで計算を行う回路を合成する。

合成を簡単にするために、計算で扱う値のビット幅は 1bit とした。正しい回路が合成できない可能性もあるが、この場合は経験的に正しい回路が合成できるとわかっている。なお、1bit なので加算と減算の区別ができないことなどは注意されたい。

計算機の通信構造は行列として表現する。本論文では通信するコア/チップのことをブロックと表記することにする。行列は、(i, j)成分の値がブロック i からブロック j へ 1 サイクルで通信できるデータの数を表す。0 である場合は

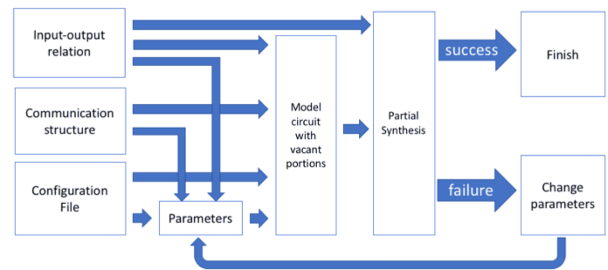

図 3 合成の流れ

その間に接続がないことになる。

まず、パラメータの値が決められる。パラメータは 5 つある。なお、1 つのブロックで 1 サイクルにつき 1 回だけ演算が行われるとする。

1. 計算にかかるサイクル数
2. 1 ブロック内の記憶素子（レジスタ）の数
3. 1 ブロックあたりの外部出力の候補の数
4. ブロック、サイクル間での演算の共有の仕方
5. 1 演算あたりの被演算子の数

演算の共有の仕方とは、例えばブロック 1 のサイクル 1 とブロック 1 のサイクル 2 が同じ演算をすることである。

次に、それらのパラメータを用いて空欄のある回路を生成する。図 4 にその回路の全体を示す。これはレジスタなどの記憶素子を含むが、ループのない組み合わせ回路であり、順序回路を時間軸で展開したものとなっている。

ブロック内の回路を図 5 に示す。組み合わせ回路であるため、レジスタは実際にはただのワイヤである。演算は LUT で定義される。

通信は通信構造に則ってワイヤを配置することで表現されている。通信されるデータはブロック内のデータから選ばれる。 LUT で任意の関数が表せるため、LUT に入力されるレジスタ同士は区別することができない。また、LUT に入力されないレジスタ同士も区別できない。よって、1 番目の通信には「LUT に入力されるレジスタ」と「LUT に入力されないレジスタ」からそれぞれ 1 個ずつを候補とし、n 番目にはそれぞれ n 個ずつを候補とした。足りない場合は足りない分を無視する。これによって QBF 問題を解く時の探索空間を狭めることができる。

最初のサイクルのブロックの入力は外部入力から選ばれる。他のサイクルのブロックの入力は、直前のサイクルの同ブロックにあるデータと、通信で受け取ったデータから選ばれる。最後のサイクルでは、他のサイクルで通信されるデータを選んだ時と同じ方法で、外部出力の候補を選ぶ。外部出力はその候補の中から選ばれることになる。

最後に、二章で説明した部分合成が行われる。解が得られた場合は、その解を用いて回路を作成する。解が得られなかった場合は、そのパラメータでは回路が合成できないことが証明されたことになるので、パラメータを変更し合成をやり直す。

ⓒ2018 Information Processing Society of Japan

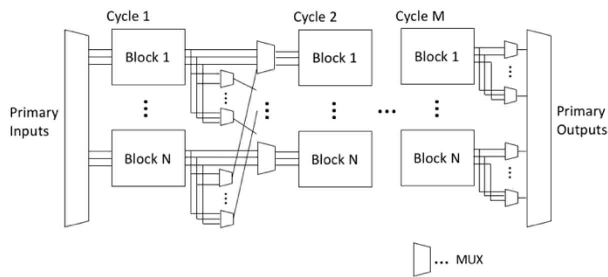

図 4 空欄のある回路の全体

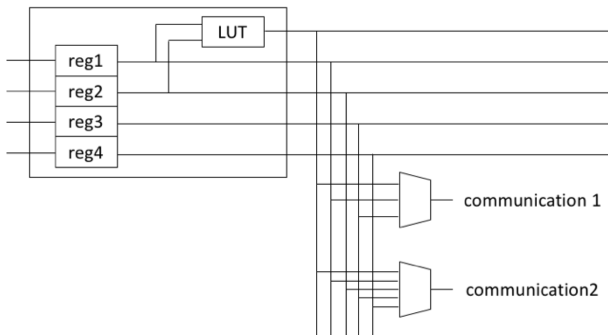

図 5 各ブロックの内部および通信ワイヤ

パラメータは最初、最も厳しい値に設定される。サイクル数は 1、レジスタ数は外部入力の数をブロック数で割った数、外部出力の候補数は外部出力の数をブロック数で割った数、演算の共有は全てのサイクル、ブロックで共有、被演算子の数は 2 である。ただし、必要であれば設定ファイルで異なる値に指定することもできる。

パラメータは図 6 で示される方法に沿って変更される。入れ子状のループであり、パラメータの組み合わせが一つ一つ試されることになる。各パラメータの上限は、設定ファイルに初期値から上限までの差を記述することによって指定する。演算の共有については以下の 4 パターンを試すことにした。(1) 全てのサイクル、ブロックで共有 (2) 全てのブロックで共有、サイクル毎に異なっても良い (3) 全てのサイクルで共有、ブロック毎に異なっても良い (4) 共有を行わない。

4.2 オプション

探索空間を狭めるために、10 個のオプションが用意されている。設定ファイルで指定することで使用することができる。

(1) 外部入力の複数レジスタへの入力禁止

最初のサイクルのレジスタに接続する外部入力について、それぞれの外部入力が接続するレジスタの数を 1 個以下に制限する。

(2) レジスタに接続する外部入力の候補をブロック毎に制限する

通常では、最初のサイクルのレジスタに接続する信号の候補は全ての外部入力となっている。しかし、それでは外部入力の数が大きくなると合成が終了しなくなる。

```
for (increase number of cycles) {
  for (increase number of registers) {
    for (increase number of candidates of primary outputs) {
      for (change the way to share the operation) {
        for (increase number of operands) {
          Try to synthesize circuit. If it succeeds, exit all loops.
}}}}}
```

図 6 パラメータの変更方法

このオプションでは、外部入力をブロックの数と同じ数のグループに分割する。そして、各ブロックのレジスタは、対応するグループの外部入力のみを候補とすることにした。

(3) 各ブロックの外部入力選択の共通化

このオプションはオプション 2 を使用した時のみ使うことができる。このオプションを使うと、ブロック間で何番目の外部入力を何番目のレジスタに接続するかということが共通化される。

例えば、ブロック 1 のレジスタ 1 に 2 番目の外部入力が接続されるとした時、他のブロックのレジスタ 1 も 2 番目の外部入力を使用することになる。なお、2 番目の外部入力とは、オプション 2 によってそのブロックに割り当てられた外部入力のうちの 2 番目という意味である。

(4) ブロック間でのデータフローの共通化

ここでいうデータフローは、どのデータを通信するか、最初のサイクル以外においてどのデータがレジスタに入力されるかということである。共通化するということを例を挙げて説明する。

通信に関しては、サイクル 1 におけるブロック 1 でレジスタ 2 を通信 1 に接続するとした時に、サイクル 1 における他のブロックでもレジスタ 2 を通信 1 に接続するということである。

レジスタに関しては、サイクル 2 におけるブロック 1 のレジスタ 1 にサイクル 1 におけるブロック 1 のレジスタ 3 が接続されるとした時に、サイクル 2 における他のブロックのレジスタ 1 に、それぞれのブロックのレジスタ 3 が接続されるということである。

(5) サイクル間でのデータフローの共通化

オプション 4 と同様の共通化をサイクル間で行う。ただし、オプション 5 ではどのサイクル間で共通化を行うかということを指定できる。例えば、サイクル 2 と 3 で共通化を行い、それとは別にサイクル 4 と 5 で共通化を行うというように指定できる。

(6) 通信されるデータを LUT の出力に限定

通常では通信されるデータはブロック内のすべてのデータから選ばれるが、それを LUT の出力に限定してしまう。

(7) 外部出力選択の固定

通常では、外部出力は外部出力の候補から選ばれる。このオプションではその候補の 1 つを以下のように外部出力

に接続してしまう。

1 番目の外部出力にはブロック 1 の外部出力の候補 1 を接続し、2 番目の外部出力にはブロック 2 の外部出力の候補 1 を接続する。同様に繰り返し、すべてブロックの外部出力の候補 1 が接続されたら、ブロック 1 の外部出力の候補 2 から再び繰り返す。

(8) LUT 関数の固定

このオプションでは LUT の関数を指定したものに固定してしまう。この固定ではすべての LUT が対象となり、例えば 2 入力 XOR を指定したら、すべての LUT は 2 入力 XOR となる。このオプションを使用した際は、パラメータの関数の共有及び被演算子の数が無視されることに注意されたい。

(9) レジスタの候補の限定

通常では、レジスタは直前のサイクルにおける同じブロックのレジスタすべてを、そのレジスタに接続する候補としている。このオプションでは、図 7 のようにその候補を限定してしまう。

まず、LUT の出力と通信で受け取ったデータをその候補に含む。これは通常と同じである。そして、前のサイクルにおける同じ番号のレジスタ (青い線) ともう一つのレジスタ (赤い線) を候補とする。赤い線は、LUT に入力されているレジスタ (reg1, reg2) に対しては、前のサイクルにおいて LUT に入力されていないレジスタを小さい番号から順番に候補としている。LUT に入力されていないレジスタ (reg3, reg4) は自分の番号より 1 つ後ろのレジスタを候補としている。ただし、対応するレジスタが存在しない場合は、その候補は無視される。

(10) 指定したレジスタの候補を 1 つ後ろのレジスタに限定する

reg3 と reg4 を指定した場合を図 8 に示す。reg3 は reg4 を候補とし、reg4 は reg5 を候補とする。しかし、reg5 が存在しないため、reg4 のその候補は無視される。この場合、2 章で説明したことから考えて、reg4 は定数 0 のみを候補とする。レジスタの指定は、後ろからの番号で指定するとする。レジスタ 3 は後ろから 2 番目、レジスタ 4 は後ろから 1 番目である。

5. 実装と実験準備

提案手法を C++のプログラムとして実装した。QBF ソルバーとしては ABC[8]の qbf コマンドを用いた。

実験は次の 4 つについて行った。

1. 相互に接続している 2 ブロックにおける 2×2 行列ベクトル積
2. 相互に接続している 2 ブロックにおける 3×3 行列ベクトル積

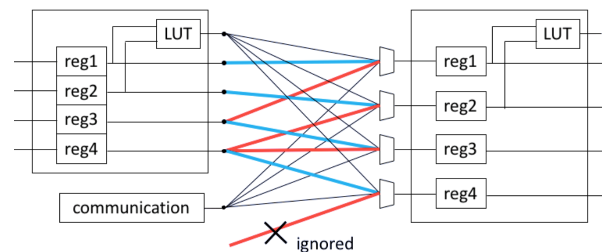

図 7 レジスタの候補の限定 (オプション 9)

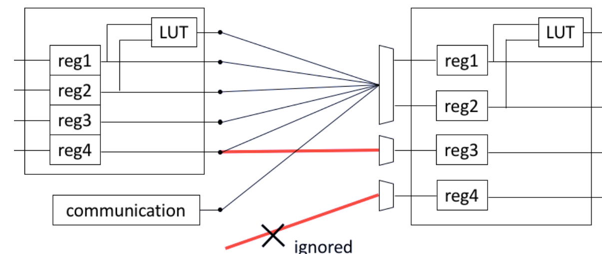

図 8 レジスタの候補の固定 (オプション 10)

3. 片方向リング状に接続している 4 ブロックにおける 4×4 行列ベクトル積
4. 片方向リング状に接続している 4 ブロックにおける 8×8 行列ベクトル積

実験 1 のパラメータは、関数の共有についてはすべて試すこととし、そのほかについては上限値までの差分を 1 とした。オプションは使用しない。

実験 2 では、すべてのパラメータで上限値までの差分を 0 とした。また、レジスタ数の初期値を 7 に固定した。これは、レジスタ数が 6 の時の合成が終了せしなかったため、最初のサイクルで reg3 が 0 でなければいけないこと、そのほかの外部入力を保持する必要があることから予測して決めた。オプションは 1, 2, 8, 10 を使用した。オプション 2 では表 1 のように外部入力がブロックに割り振られた。オプション 8 では関数を(3)に固定した。

実験 3 では、レジスタ数の上限値までの差分を 1、他は 0 とした。オプションは 1, 2, 8 に加えて、計算が対称的であることから 4, 5, 7 を使用した。オプション 2 では表 2 のような割り当てが行われた。オプション 5 ではサイクル 2, 3, 4 において共通化を行なった。オプション 8 では関数を(3)に固定した。

実験 4 では、レジスタ数の上限値までの差分を 2、他は 0 とした。オプションはすべて使用した。オプション 3 を使用するために、外部入力が表 3 のように対称に割り当てられるように注意した。オプション 5 では、データが 1 周するサイクル 5, 9, 13 以外において共通化を行い、またサイクル 5 と 13 を別途共通化し

ⓒ2018 Information Processing Society of Japan

表 1 実験 2 における外部入力の割り当て

ブロック	外部入力
1	Is1, w11, w12, w13, Is2, w21
2	w22, w23, Is3, w33, w32, w33

表 2 実験 3 における外部入力の割り当て

ブロック	外部入力
1	Is1, w11, w21, w31, w41
2	Is2, w12, w22, w32, w42
3	Is3, w13, w23, w33, w43
4	Is4, w14, w24, w34, w44

表 3 実験 4 における外部入力の割り当て

ブロック	外部入力
1	Is1, w11, w21, w31, w41, w51, w61, w71, w81, Is5, w55, w65, w75, w85, w15, w25, w35, w45
2	Is2, w22, w32, w42, w12, w62, w72, w82, w52, Is6, w66, w76, w86, w56, w26, w36, w46, w16
3	Is3, w33, w43, w13, w23, w73, w83, w53, w63, Is7, w77, w87, w57, w67, w37, w47, w17, w27
4	Is4, w44, w14, w24, w34, w84, w54, w64, w74, Is8, w88, w58, w68, w78, w48, w18, w28, w38

た。オプション 8 では関数を(4)に固定した。オプション 10 では 1 番目から 15 番目までを指定した。

$$(reg1 \cdot reg2) \oplus reg3 \quad (3)$$

$$reg1 \oplus (reg2 \cdot reg3) \quad (4)$$

6. 実験結果

最後のパラメータの値と QBF 問題の特性を表 4 に示す。

成分の積の計算が 1 ブロックで 1 サイクルにつき 1 回行われるとすると、これらのサイクル数は最適である。例えば、実験 1 では 2 ブロックで 4 回の掛け算をしなければならないから、少なくとも $4 \div 2 = 2$ サイクル必要となる。

また、全称量化された変数の数が計算時間に主たる影響を与えていることがわかる。

(1) 実験 1

合成された回路を図 9 に示す。LUT の関数は(5)の通りであった。

この結果では、通信されるデータが 3 章で取り上げた例とは異なっている。例では LUT の出力である成分の積が通信されているが、これではベクトルの成分が通信されている。

1bit の計算であったため、LUT の論理関数に否定が現れているが、単純に否定を無視すれば正しい計算を行える。

表 4 最後のパラメータの値と QBF 問題の特性

実験番号	1	2	3	4
サイクル数	2	5	4	16
レジスタ数	3	7	6	20
演算の共有方式	2	1	1	1
被演算子数	3	3	3	3
全称量化された入力の数	6	12	20	72
存在量化された入力の数	94	492	207	428
変数の数	302	1,984	1,481	6,852
節の数	947	7,097	6,157	29,467
リテラルの数	2,810	20,382	18,001	82,468
計算時間 (s)	2	39,083	3	4,452

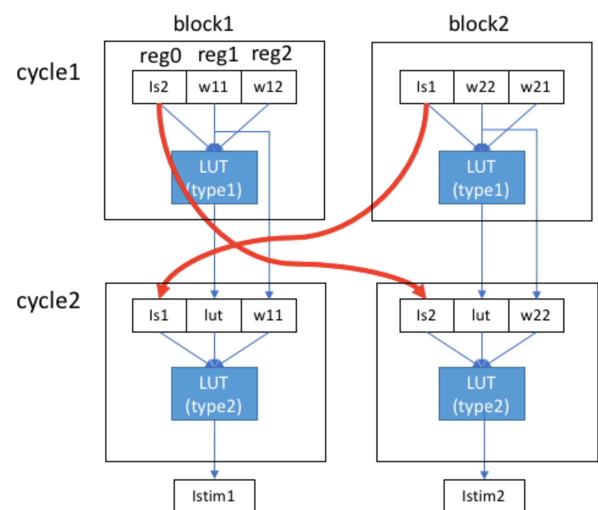

図 9 実験 1 で得られた回路

$$\begin{cases} type1 = \overline{reg2 \cdot reg0} \\ type2 = \overline{reg1} \oplus (reg0 \cdot reg2) \end{cases} \quad (5)$$

(2) 実験 2

得られた回路を図 10 に示す。この図では同じブロック内のデータの移動の矢印は省略した。

通信を行う必要がないサイクルがいくつかあることがわかる。そのことから、1 サイクル目ではブロック 1 からブロック 2 への通信、2 サイクル目ではブロック 2 からブロック 1 への通信というように交互に通信を行う構造でも最適なサイクル数で計算が行えるかもしれない。

(3) 実験 3

得られた回路のデータフローグラフを図 11 に示す。

実験 1 で得られたものと同様に、ベクトルの成分を通信するアルゴリズムを得ることができた。これを一般化すれば n×n の行列ベクトル積を n 個の片方向リング状に接続されたブロックで計算する場合、ベクトルの成分を通信すれば、最適なサイクル数で計算が行えることがわかる。

ⓒ2018 Information Processing Society of Japan

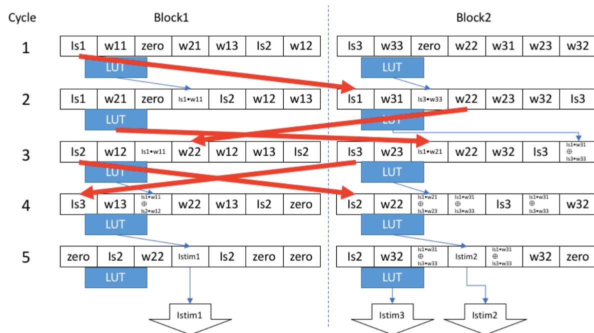

図 10 実験 2 で得られた回路

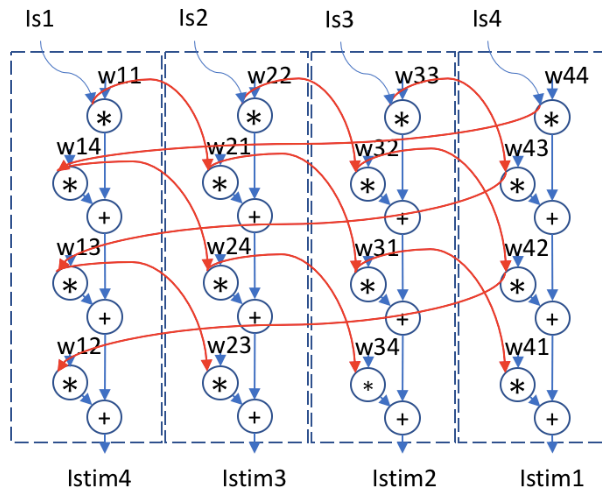

図 11 実験 3 で得られた回路のデータフローグラフ

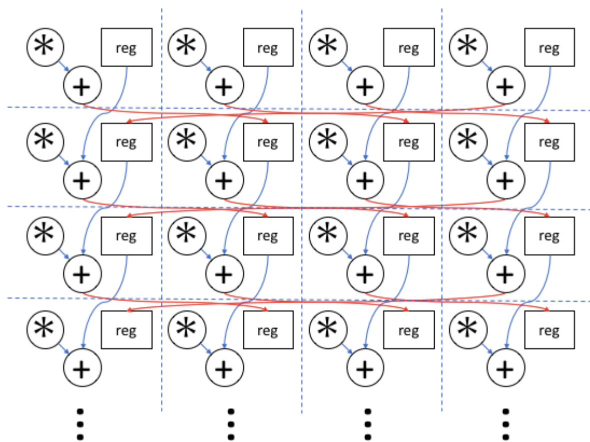

図 12 実験 4 で得られた回路のデータフローグラフの一部

(4) **実験 4**

得られた回路のデータフローグラフの一部を図 12 に示す。3 章で取り上げた例と比較すると、通信されたデータを格納するレジスタが増えていることがわかる。ここから一般化を考えると、4N×4N の行列ベクトル積を 4 つの片方向リング状に接続されたブロックで計算する場合、N-1 個のレジスタを用意すれば最適なサイクル数で計算できることが予想できる。

7. 結論

提案手法は計算機の通信構造を考慮してアルゴリズムを合成するというものであった。

小規模な計算に対しては自動でアルゴリズムを合成することができた。実際には、それを参考にして人間が大規模な計算のアルゴリズムを考えることになるだろう。アルゴリズムの一般化は、6 章の実験結果で書いたように行うことができる。

一般化されたアルゴリズムの検証は、記号シミュレーションと SMT ソルバーによって行うことができる。更に言えば、アルゴリズムの一般化においてテンプレートベースの合成手法を利用できるだろう。それは、一般化したいアルゴリズムを、空欄のあるプログラムとして記述し、そのプログラムをテンプレートとして合成を行うということである。

パラメータとオプションの調節は重要である。いくつかの実験を行い、問題の特性を理解した上でそれらの値を決定する必要があり、一般に簡単ではない。このプロセスを簡略化することは今後の課題である。

また、多くのオプションは計算の対称性を前提としており、非対称な計算を行うアルゴリズムを合成する際には用いることができない。そのため非対称な計算のアルゴリズムを合成することが困難になっていることも今後の課題である。これを解決するためには、空欄のある回路の構造をより簡単なものに変える必要があると思われる。

References

[1] M. Mathews, J. P. Abraham, “Automatic Code Parallelization with OpenMP Task Constructs,” International Conference on Information Science (ICIS), 2016.

[2]P. Mi, Z. Zhao, W. Sheng, W. He, “An Automatic Parallelizer for Coarse-Grained Reconfigurable Processor,” IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), 2016.

[3]N. Ehsan, S. Javeed, H. P. Andre, B. Vaughn, “Multiple Dice Working as One : CAD Flows and Routing Architectures for Silicon Interposer FPGAs,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 24, no. 5, pp. 1821-1834, 2016.

[4]T. Strauch, “Timing driven RTL-to-RTL partitioner for multi-FPGA systems,” Field Programmable Logic and Applications (FPL), 2013 23rd International Conference, 2013.

[5]K. Matsuda, T. Miyoshi, S. Funada, H. Nakajo, “Implementation and Evaluation for Circuit Partitioning with a Java-based High-Level Synthesis Tool,” Information Processing Society of Japan, vol. 56, no. 8, pp. 1582-1592, 2015. In Japanese.

[6]T. Okamoto, T. Kawao, T. Kohno, M. Fujita, “Spiking Neural Network Simulation Accelerator Using Multiple FPGA Chips,” IPSJ Transactions on System LSI Design Methodology, vol. 181, no. 30, pp. 1-6, 2017. In Japanese.

[7]M. Fujita, S. Jo, S. Ono, T. Matsumoto, “Partial Synthesis through sampling with and without specification,” 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2013.

[8]Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 70607. http://www.eecs.berkeley.edu/~alanmi/abc/

ⓒ2018 Information Processing Society of Japan