Vol.2021-SLDM-196 No.11
2021/12/1

ループ平坦化におけるループ回数の 2 のべき乗化による回路最適化

伊澤　昇平†　　瀬戸 謙修

東京都市大学工学研究科 〒158-8557 東京都世田谷区玉堤 1-28-1

E-mail:　†g2081205@tcu.ac.jp

要約　自動運転用 LSI など高性能なハードウェアは設計期間が増加する問題があり、設計期間の短縮が求められている。そこで、C 言語からハードウェアを自動生成する高位合成が注目されている。現在の高位合成技術では、高性能なハードウェアを設計するために、ソースコード最適化が必要である。コード最適化の 1 つであるループ平坦化は、多重ループを単一ループにする最適化であり、ループ処理の実行時間を短縮できる。しかし、既存のループ平坦化では、元々のループ変数を復元する処理に除算や剰余算が使用されるため、回路面積と消費電力が増加する問題点がある。本論文では、この問題に対処するため、多重ループの内側ループの反復回数を 2 のべき乗に変更することでループ変数復元処理を簡略化する手法を提案する。実験の結果、最適化前と比べ、平均で、サイクル数、総実行時間、消費エネルギーをそれぞれ 8%、20%、12%削減し、回路面積は 4%、消費電力は 9%増加した。３つの例題において、既存のループ平坦化と比較した結果、提案手法は同じサイクル数削減率を達成し、平均で、総実行時間を 18%、回路面積を 31%、消費電力を 64%、消費エネルギーを 84%削減することができた。よって本論文の提案手法は、高位合成向けループ平坦化手法の 1 つとして有効であることが分かった。

キーワード　ループ平坦化、高位合成

1. はじめに

C 言語等の高級言語から HDL(Hardware description language)を自動生成する高位合成では、ハードウェアの要求性能を満たすために高位合成前にソースコード最適化を行う必要があることが多い。高位合成向けの最適化には、ループ最適化[1][2][3][4][5]やメモリアクセス最適化 [6][7][8]などが存在する。ループ最適化は、for、while ループに着目し、反復処理の処理時間を削減する。メモリアクセス最適化は、配列のアクセス数を削減し、ループの並列性を向上する。本論文では、ループ最適化の一つであるループ平坦化[1][2][3]に着目する。ループ平坦化のサンプルコードと実行イメージを図 1 と図 2 に示す。

```
1.  for(i=0; i<2; i++){
2.      for(j=0; j<3; j++){
3.          //3サイクル処理(ループパイプライン化実施)
4.  }}
```

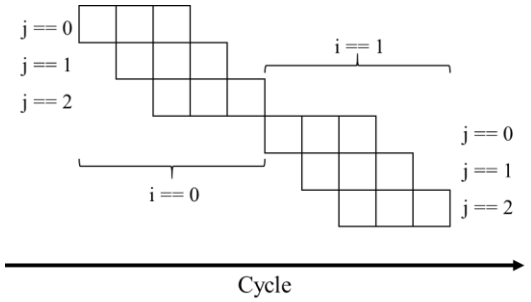

図 1.(上)多重ループ(平坦化前)
(下)実行イメージ

```
1.  for(n=0; n<(2*3); n++){
2.  //3サイクル処理(ループパイプライン化実施)
3.  }
```

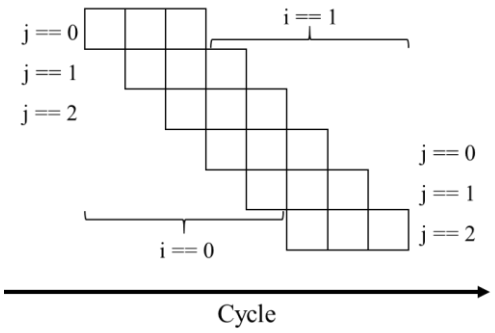

図 2.(上)単一ループ(平坦化後)
(下)実行イメージ

ループ平坦化は図 1(上)の多重ループを図 2(上)の単一ループに変換する最適化である。図 1、図 2 では、共に開始間隔 1 でループパイプライン化している。図 1(上)のような多重ループの場合、内側ループの処理が完了した後、外側ループのインクリメントが行われるため、内側ループの処理終了を待つ待機時間が発生する(図 1(下))。ループ平坦化は、この待機時間を削減することができるため、ループ処理の並列性が向上する(図 2(下))。しかし、ループ平坦化を適用すると、ループ変数 i、j などから n に変更される。図 1(上)の 3 行目や図 2(上)の 2 行目の処理において配列アクセスなどでループ変数を用いる場合、元のループ変数を復元して実行文の整合性を保つ必要がある。従来のループ変数の復元方法として、ループ平坦化適用後のループ変数に対して除算、剰余算を用いることにより復元で

きる(図 4 の 2 行目から 4 行目)が、除算や剰余算は回路規模が大きいため、高位合成において回路面積が増加するという問題点がある。先行研究では、剰余算を減算と乗算に置き換える Modulo Free Flatten(MFF)手法が提案された[2]。

本論文で提案するループ平坦化手法では、多重ループの内側ループの反復回数を 2 のべき乗値に拡張した後にループ平坦化を適用する。反復回数を 2 のべき乗にすることにより、元のループ変数復元処理に 2 のべき乗の値が用いられる(図 7)。2 のべき乗値を用いた除算や剰余算はビットシフト演算と論理積に置き換えることが可能になるため(図 8)、演算処理が簡略化され、回路の遅延時間が短縮できると考える。また、高位合成時の演算器が削減できることから回路面積が削減でき、それに伴い消費電力も削減できることが期待される。

よって、本論文では、従来のループ平坦化適用後の総実行時間、回路面積及び消費電力の改善を目指し、多重ループの内側ループの反復回数を 2 のべき乗に拡張したループ平坦化の手法を提案する。そして、既存の平坦化手法と提案手法をそれぞれ使用した高位合成を行う。既存の平坦化手法[1][2]と提案手法を比較することにより、提案手法がループ平坦化の 1 つの手法としての有効性を示すことを目的とする。

2. 先行研究

本節では、2.1 節に従来のループ平坦化、2.2 節に先行研究で提案された平坦化手法(MFF 手法)について述べる。

2.1. 従来の平坦化手法[1]

図 3 に元のソースコード、図 4 に従来のループ平坦化[1]を適用したコードをそれぞれ示す。

```
1: for(i=0; i<7; i++){
2:     for(j=0; j<7; j++){
3:         for(k=0; k<7; k++){
4:             A[i][j] = B[i][k] * C[k][j];
5: }}}
```

図 3. ソースコード

```
1: for(n=0; n<(7 * 7 * 7); n++){
2:     i = n / (7 * 7);
3:     j = n / 7 % 7;
4:     k = n % (7 * 7) % 7;
5:     A[i][j] = B[i][k] * C[k][j];
6: }
```

図 4. 従来のループ平坦化適用コード

図 3 は、行列積を想定した各ループ回数が 7 回で統一された 3 重ループである。図 4 は、図 3 に従来のループ平坦化を適用して単一ループに平坦化したコードである。図 4 の 2 行目から 4 行目は、元のループ変数を復元する処理であり、ループ平坦化適用により新たに生成されたループ変数 n と内側ループの反復回数 7、除算と剰余算、乗算を用いて表現される。図 4 で使用される除算や剰余算は、回路規模が大きいため、高位合成において回路面積が増加する。さらに、回路面積が増加することで消費電力も比例して増加する問題点がある。ループ平坦化のデメリットである回路面積増加を改善するために提案された手法が、2.2 節で述べる Modulo Free Flatten(MFF)手法である。

2.2. Modulo Free Flatten(MFF)手法[2]

MFF 手法は、ループ平坦化後に必要になる元のループ変数復元処理に使用される剰余算を乗算と減算に置き換えるループ平坦化手法である。MFF 手法を適用することにより、剰余算による回路面積増加を抑えることができる。図 3 のソースコードに対して MFF 手法を適用したコードを図 5 に示す。

```
1: for(n=0; n<(7 * 7 * 7); n++){
2:     i = n / (7 * 7);
3:     j = n / 7 – i * 7;
4:     k = n – i * (7 * 7) – j * 7;
5:     A[i][j] = B[i][k] * C[k][j];
6: }
```

図 5.MFF 手法

図 5 は、図 3 に MFF 手法を適用したコードであり、図 4 で使用されていた剰余算が乗算と減算で表現される(3,4 行目)。MFF 手法の適用により、剰余算が使用されないため、高位合成時の回路面積増加を従来の平坦化(図 4)よりも抑えることができる。しかし、MFF 手法では、除算を用いているため、回路面積の増加を抑えることを十分に達成できない。また、用いる演算器が増加することにより、処理に時間がかかるという問題点がある。

本論文では、ループ平坦化による回路面積の増加をさらに抑える手法の 1 つとして、第 3 節で述べる手法を提案する。

3. 提案手法(べき乗平坦化)

本節では、前節で述べた既存のループ平坦化手法[1][2]の問題点である回路面積増加を改善する 1 つの手法として、べき乗平坦化手法を提案する。べき乗平坦化手法は、図 3 のような多重ループの内側ループの反復回数を 2 のべき乗に拡張させた後に、従来のループ平坦化を適用する手法である。反復回数を拡張させていることから、元のコードと比べて実行回数が多くなるため、if 文による制御文を追加することにより、実行回数を同じに保つ。本節では、3.1 節で提案手法のコード変換手順を示し、3.2 節で整合性を保つための制御文について述べる。

3.1. べき乗平坦化

本節では、図 3 をソースコードとして、図 6 に図 3 の内側ループを 2 のべき乗に拡張したコード、図 7 に図 6 のコードに対して従来のループ平坦化[5]を適用したコード、図 8 に、図 7 の 2 行目から 4 行目にあるループ変数復元処理の除算と剰余算をビットシフト演算と論理積に置き換えたコードを示す。

```
1: for(i=0; i<7; i++){
2:     for(j=0; j<8; j++){
3:         for(k=0; k<8; k++){
4:             A[i][j] = B[i][k] * C[k][j];
5: }}}
```

図 6. ソースコードの内側ループ回数を 2 のべき乗値に拡張

```
1: for(n=0; n<7*8*8; n++){
2:     i = n / (8 * 8);
3:     j = n / 8 % 8;
4:     k = n % (8 * 8) % 8;
5:     A[i][j] = B[i][k] * C[k][j];
6: }
```

図 7. 図 6 に従来のループ平坦化を適用

```
1: for(n=0; n<7 * 8 * 8; n++){
2:     i = n >> 6;
3:     j = (n >> 3) & 7;
4:     k = n & 7;
5:     A[i][j] = B[i][k] * C[k][j];
6: }
```

図 8. ビットシフト演算と論理積に置換

図 6 のコードに対して従来のループ平坦化を適用することで、図 7 のようにループ変数復元処理(2 行目から 4 行目)の計算に 2 のべき乗の値が使用される。2 のべき乗値を用いた除算や剰余算は、図 8 のようにビットシフト演算と論理積に置換できる。ハードウェアにおいて、2 のべき乗での処理は、計算の高速化や演算器数を削減できるため、回路の実行時間の短縮と回路面積及び消費電力の削減が期待できる。

しかし、図 7 のコードは、ループ回数を拡張しているため元のコードと動作が異なる問題点がある。この問題の解決方法は、3.2 節で述べる。

3.2. if 文を用いたループ実行制御部文

本節では、上記 3.1 節で述べた提案手法の問題点を解決するための制御文について述べる。図 3 において、ループ変数 i、j、k は、すべて 0 から 6 までの値をとるが、べき乗値に拡張した図 6 のループ変数 j、k は 0 から 7 の値になる。このままでは、定義されていない配列にアクセスするため、コンパイルエラーが発生するなど、元のコードと動作が異なる。このエラーを回避するために if 文を用いた制御文を追加することにより、べき乗に拡張したループを元のループと同じ動作にする必要がある。図 9 に図 8 の 5 行目と 6 行目に追加する制御文を示す。

```
1: if(k == 6){
2:     n += 1;
3:     if(j == 6){
4:         n += 8;
5: }}
```

図 9. if 文制御文

図 9 の n は、ループ平坦化により新たに生成されたループ変数である。多重ループは、最内側ループからインクリメントされるため、if 文でのループ回数制御も最内側ループのループ変数 k から行う。その後、ループ変数 j を制御することで、べき乗拡張後のループ回数を拡張前のループ回数と同じにする。ループ変数 i はべき乗に拡張しないため、ループ変数 i については制御文に含めない。

べき乗拡張前(図 3)のループ変数 k、j の範囲は 0 から 6 までであり、図 9 の 1 行目と 3 行目には、最大アクセスの 6 を記述する。図 3 では、ループ変数 k の反復回数は 7 回だが、図 6 のループ変数 k の反復回数は 8 である。その差が 1 であるため、図 9 の 2 行目では、ループ回数 n に 1 を加え、不要なループ実行を回避する。4 行目では、ループ変数 n に 8 を加えている。これは、ループ j が 7 になった時のループ k の実行は全て不必要となるため、この不必要なループを回避する目的で加えられている。この制御文と図 8 を組み合わせたコードを図 10 に示す。

```
1: for(n=0; n<7 * 8 * 8; n++){
2:     i = n >> 6;
3:     j = (n >> 3) & 7;
4:     k = n & 7;
5:     A[i][j] = B[i][k] * C[k][j];
6:     if(k == 6){
7:         n += 1;
8:         if(j == 6){
9:             n += 8;
10:    }}}
```

図 10.べき乗平坦化と制御文

図 10 コードでは、制御文により、定義されていない配列アクセスを回避することができるため、コンパイル時にエラーが発生しない。そして、出力の値も最適化前後で一致することを確認した。

図 3 から図 10 までに挙げたコードは、ループ回数（この場合、7）より大きく、最も近い 2 のべき乗値（この場合、8）との差が 1 である場合を想定したが、

この差は必ずしも 1 ではない。例えば、元の各ループ回数が 33 回である場合、33 より大きい 2 のべき乗は 64 であり、その差は 31 になる。各ループ回数に対して、図 9 のように n に加算する数値（1 や 8）を求め、実験を行うことは非効率である。本実験では、図 9 の制御文をループ回数と 2 のべき乗値を用いた一般的なコード表現にして実験を行った。その一般的な表現を図 11 に示す。

```
1: if(k == (N - 1)){
2:     n += (E - N);
3:     if(j == (N - 1)){
4:         n += (E * (E - N));
5: }}
```

図 11. 一般化した制御文

ここで、N は 2 のべき乗値ではないループ回数、E は N より大きく N に最も近い 2 のべき乗値である。例えば、N が 5 から 7 の場合は、E は 8 となり、N が 9 から 15 の場合、E は 16 となる。本論文では、ループ回数 N を 3 から 64 回まで変化させ、図 10 のコードをそれぞれ作成して高位合成を行った。

4. 実験準備

本節では、実験に用いたツールや例題、評価方法について述べる。

本実験では、商用の高位合成ツール、論理合成ツールを用いた。高位合成時は、回路の並列性を高めるループパイプラインと 1 回の反復処理で配列の読み取り、書き込みを安全に行うオプションを適用した。動作周期は 5.0ns に設定し、テクノロジライブラリはプロセスルールが 45nm であるものを使用した。

用いた例題は、Polybench[8]に含まれている mms、2mm、3mm の 3 つを用いた。mms は行列積を 1 回行うコード、2mm は行列積を 2 回行うコード、3mm は行列積を 3 回行うコードである。これらのコードは、3 重ループが 2 つ以上存在するコードであり、第 1 節で述べたループ間の待機時間が存在するものを取り上げた。実験において、ループ平坦化の効果を最大限に高めるために、これらの例題に対して開始間隔が 1 で動作するようにコードを変換した。

比較対象は、最適化前、MFF による平坦化(既存手法)、べき乗平坦化(提案手法)とした。ループ回数による回路性能の変化を考慮するため、各ループのループ回数を 3 回から 64 回まで変化させ、計 61 回実験を行った。このとき、多重ループの各ループの反復回数は統一した。

評価方法は、設定した動作周期からスラックを引いた差(回路遅延)とサイクル数の積で表される総実行時間と論理合成により算出された回路面積と消費電力を評価した。

5. 実験結果

本節では、用いた例題の中で最も単純なプログラムである mms を一例として実験結果を示す。最適化前のオリジナルコード、既存手法 MFF を適用したコード、提案手法べき乗平坦化を適用したコードをそれぞれ高位合成する。すべての手法において、ループの開始間隔は 1 で正常に動作し、実験前後で出力値が一致することを確認した。各手法における回路の総実行時間、回路面積、消費電力を比較し、提案手法の有効性を示す。

表 1 に mms の高位合成結果を示す。表 1 の数値は、最適化前の値を 100%としたときの百分率を示す。表のループ回数は、高位合成により出力される回路面積の値が最小値になる場合(N=3)、最大値になる場合(N=48)、ループ回数がべき乗値 64 から最も離れる場合(N=33)を示す。

サイクル数は、高位合成により出力された値と、各手法の比率を示す。遅延時間は、設定した動作周期 5.0ns とスラックの値の差である。総実行時間は、サイクル数と遅延時間を乗算した値を用いた。回路面積と消費電力は、論理合成により出力された値を用いた。消費エネルギーは、総実行時間と消費電力を乗算した値を用いた。

サイクル数において、既存手法、提案手法ともに最適化前より削減できることから、ループ平坦化のメリットである並列処理向上の効果が表れることを確認した。

ループ平坦化は、回路の並列性を向上し、実行時間を短縮することが期待されるが、既存手法では、一部のループ回数において遅延時間が最適化前より増加することを確認した。これは、除算などの演算による遅延増加が原因と推測する。提案手法では、最適化前より遅延時間が増加することはないことを確認した。また、多くのループ回数において提案手法は既存手法より遅延時間が小さくなることを確認した。これは、べき乗化により、ループ変数復元処理がビットシフト演算と論理積に置き換わることで、処理が高速化したと考える。提案手法では、ループ回数を制御する if 文を追加するが、制御文の処理時間の増加よりも回路全体の処理時間の短縮が上回ったと考える。この結果から、提案手法は、既存手法よりも総実行時間の短縮が期待できるため、ループ平坦化の 1 つの手法として有効であると考える。

ループ平坦化を適用する場合、回路面積や消費電力が増加する問題点がある。既存手法、提案手法共にループ回数が少ない場合、最適化前よりも回路面積は削減したが、ループ回数が多い場合、回路面積は増加した。これは、ループ回数が少ない場合、演算器の規模

表1. mmsの高位合成結果

ループ回数N[回]	3（面積出力最小値）			33（べき乗値64から最も離れた点）			48（面積出力最大値）		
手法	最適化前	既存	提案	最適化前	既存	提案	最適化前	既存	提案
サイクル数[kCycle]	0.105 (100%)	0.070 (92%)	0.073 (92%)	77.39 (100%)	73.04 (94%)	73.03 (94%)	232.8 (100%)	223.6 (96%)	223.6 (96%)
遅延時間[ns]	1.182	1.268	1.034	3.238	4.366	3.315	3.665	3.575	3.782
総実行時間[%]	100	75	61	100	127	97	100	92	97
回路面積[%]	100	89	86	100	104	104	100	138	110
消費電力[%]	100	104	95	100	189	109	100	159	115
消費エネルギー[%]	100	78	58	100	241	105	100	147	111

表2. 各例題におけるループ回数3回から64回の高位合成結果の平均値

例題	mms			2mm			3mm			既存手法の	提案手法の
手法	最適化前	既存	提案	最適化前	既存	提案	最適化前	既存	提案	平均値	平均値
サイクル数[kCycle]	146.8 (100%)	141.1 (92%)	141.1 (92%)	146.8 (100%)	141.1 (92%)	141.1 (92%)	219.5 (100%)	210.9 (92%)	210.9 (92%)	92	92
遅延時間[ns]	3.073	3.292	3.067	3.237	3.340	3.105	2.899	3.144	1.798	3.259	2.657
総実行時間[%]	100	98	91	100	95	89	100	102	60	98	80
回路面積[%]	100	137	106	100	133	101	100	135	105	135	104
消費電力[%]	100	169	115	100	167	112	100	182	99	173	109
消費エネルギー[%]	100	168	105	100	161	100	100	186	60	172	88

が小さく、並列処理により効率的に実行できたためと考える。提案手法において、べき乗から離れているループ回数 33 回のときは、既存手法と同一の結果になった。その他のループ回数では、既存手法よりも回路面積増加を抑えることが分かった。この理由として、既存手法は、除算などの演算を多く行うため、演算器面積が増加するが、提案手法は、べき乗化によりループ変数復元処理にビットシフト演算と論理積が用いられることで必要最小限の演算器で実装されるためである。また、制御文追加により回路面積が増加するが、提案手法の面積増加は、既存手法の面積増加よりも小さいことが分かった。よって、提案手法は、ループ平坦化のデメリットである回路面積増加を既存手法より抑えることから、回路面積の観点からもループ平坦化として有効であると考える。

消費電力において、既存手法は、ループ平坦化の際、ループ変数復元処理に多くの演算が必要になる。ループ回数が少ない場合は、用いられる演算器の規模が小さいことから、消費電力が小さくなったと考える。ループ回数が多い場合、演算器の規模が大きくなること、演算処理に多くのサイクルがかかることにより消費電力が大きくなると考える。提案手法は、ループ変数復元処理にビットシフト演算と論理積を用いており、既存手法よりも少ない演算器で処理できるが、制御文を追加するため、消費電力が増加すると考える。すべてのループ回数のとき、提案手法は既存手法より消費電力の増加を抑えることができた。よって、提案手法は、消費電力の観点からも既存のループ平坦化と比べて優れていると考える

回路全体の消費エネルギーにおいて、提案手法は多くのループ回数で、既存手法よりも総実行時間が小さくなった。全てのループ回数で既存手法よりも消費電力が小さくなるため、消費エネルギーも既存手法より小さくなることを確認した。よって、提案手法は、消費エネルギーの観点からもループ平坦化の 1 つの手法として有効であると考える。

ループ平坦化を適用したことにより、既存手法、提案手法ともに、サイクル数を最適化前より削減できたことを確認し、多くのループ回数において回路面積と消費電力が最適化前よりも増加することを確認した。提案手法は、多くのループ回数において、既存手法よりも遅延時間が小さくなり、総実行時間が削減した。また、提案手法の回路面積と消費電力の増加率は、既存手法よりも小さく抑えることができることが分かった。

次に、用いた例題 mms、2mm、3mm において、ループ回数を 3 回から 64 回まで変えて高位合成し、出力されたサイクル数、総実行時間、回路面積、消費電力、消費エネルギーの各 61 個の平均値を表 2 に示す。ここで、表 2 の各項目の詳細は、表 1 と同様である。表 2 の最右側の既存手法の平均値、提案手法の平均値は、表 2 の既存手法、提案手法の各項目のそれぞれ 3 つの例題に対する値の平均値である。表 2 より、mms、2mm、3mm の各例題においてもループ平坦化によるサイクル数の削減ができたことを確認した。mms と 2mm では、回路面積と消費電力が増加することを確認した。3mm では、回路面積は増加したが、消費電力は最適化前と変わらなかった。提案手法は、最適化前、既存手法よりも遅延時間が小さく、回路の総実行時間を短縮できることが分かった。3mm では、最適化前と比べて、

提案手法の総実行時間が 60%となった。これは、3mm は、第 1 節で述べたループ間の待機時間が他 2 つの例題よりも多く、ループ平坦化による待機時間の削減の効果が大きく表れたと考える。表 2 の最右側の既存手法の平均値と提案手法の平均値を比較した結果、提案手法は、既存手法と同じサイクル数削減率を達成し、総実行時間を 18%､回路面積を 31%､消費電力を 64%、消費エネルギーを 84%削減することができた。

以上のことから、提案するべき乗平坦化は、既存手法と比べて､回路の総実行時間を短縮することができ、回路面積と消費電力の増加を抑えることができることから、ループ平坦化によるメリットを大きくし、デメリットを小さく抑えることが分かった。したがって、提案手法は、高位合成において、ループ回数が２のべき乗でないループを平坦化する際に、有効であると結論する。

5. 結論

本論文で提案したループ平坦化手法であるべき乗平坦化は、多重ループにおける内側ループの反復回数が 2 のべき乗でない場合に 2 のべき乗に拡張し、既存のループ平坦化を適用することで、ループ変数復元処理における除算と剰余算をビットシフト演算と論理積に置換できる。演算処理を簡略化することで、既存手法よりも回路の総実行時間と回路面積及び消費電力を削減することを可能にした。多重ループの各ループ回数を同じ値に設定した条件のもとで、べき乗拡張後のループ変数制御文を一般的に表現した。ループ回数を 3 回から 64 回まで変化させて実験を行った結果､最適化前と比べて、平均で、サイクル数、総実行時間をそれぞれ 8%、20%、消費エネルギーを 12%削減し、回路面積は 4%、消費電力は 9%増加した。実験に用いた 3 つの例題に対して、既存手法と比較した結果、提案手法は同じサイクル数削減率を達成し、平均で、総実行時間を 18%、回路面積を 31%、消費電力を 64%、消費エネルギーを 84%削減することができた｡したがって、提案手法べき乗平坦化は、既存のループ平坦化手法よりも回路の総実行時間、回路面積、消費電力、消費エネルギーを削減できることからループ平坦化手法の 1 つとして有効である。

文　献

[1] Jongeun Lee, Seongseok Seo, Hongsik Lee, and Hyeon Uk Sim. ”Flattening-based Mapping of Imperfect Loop Nests for CGRAs” In CODES+ISSS. IEEE, 2014.

[2] Hyeonuk Sim, Atul Rahman, and Jongeun Lee “Efficient High-Level Synthesis for Nested Loops of Nonrectangular Iteration Spaces” IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 24, NO.8, pp2799-2802, Aug. 2016

[3] Kazuaki Kawamura, Masao Yanagisawa, and Nozomu Togawa “A Loop Structure Optimization Targeting High-level Synthesis of Fast Number Theoretic Transform” 19th Int'l Symposium on Quality Electronic Design. IEEE 2018

[4] Yuta Kato, Kenshu Seto, “Loop Fusion with Outer Loop Shifting for High level Synthesis,” IPSJ Transactions on System LSI Design Methodology, Vol. 6, pp.71 75, 2013.

[5] Optimizing supercompilers for supercomputers / Michael Wolfe. MIT Press, 1989.

[6] Kenshu Seto. “Scalar Replacement with Polyhedral Model” IPSJ Transactions on System LSI Design Methodology Vol.11 46-56(Aug. 2018)

[7] Kenshu Seto. “Scalar Replacement with Circular Buffers” IPSJ Transactions on System LSI Design Methodology Vol.12 1–9 (Feb. 2019)

[8] Daewoo Kim, Sugil Lee, Jongeun Lee “On-chip Memory Optimization for High-level Synthesis of Multi-dimensional Data on FPGA” ASPDAC January 2019 Pages 243–248

[9] Louis-Noel Pouchet, Tomofumi Yuki, "PolyBench/C 4.2, " Ohio State University, May.2016