6J-2

# 高位合成向け動作記述に対するメモリアクセス削減ツール

式部　剛広　　瀬戸　謙修　　丸泉　琢也
東京都市大学 工学部 電気電子工学科

## 1　はじめに

近年、半導体技術の発展により LSI の多機能化、回路規模の増大が起こっている。よって、設計生産性の向上を目的として、動作記述からハードウェア記述を生成する高位合成 [1] と呼ばれる技術が注目されている。動作記述とは C 言語などの高級プログラミング言語で書かれた記述であり、高位合成によって verilog などで書かれたハードウェア記述を自動生成できる。しかし、高位合成の現状として、与えられた C 記述に対してそのまま高位合成を適用しても、性能制約を満足する RTL を生成できない場合も多く、通常 C 言語の最適化が必要となる事が多い。

## 2　ハードウェア合成向け C 言語の最適化

ハードウェア合成向け最適化にはループ最適化とメモリアクセス最適化が含まれる [2]。これらの最適化により高性能なパイプラインを生成することが可能になる。ループ最適化の１つとして複数に分けられているループ文を１つにまとめ、並列処理の機会を増やすループ融合があげられる。また、メモリアクセス最適化は並列処理にてボトルネックになる事が多いメモリポート数の同時アクセス制限に対処するため余分なメモリアクセスをできるだけレジスタ等を用いて減らすものである。本稿ではメモリアクセス最適化を自動で行うことにより高性能なパイプラインの自動生成を目指した。ループパイプラインとはあるループのイタレーションが終わる前に、次のイタレーションの実行を開始する技術である [3]。性能を測る上で、連続するループを実行する実行サイクルの間隔である開始間隔（II）と実行サイクル数が重要となる。本研究ではこれらを少なくすることが目標である。

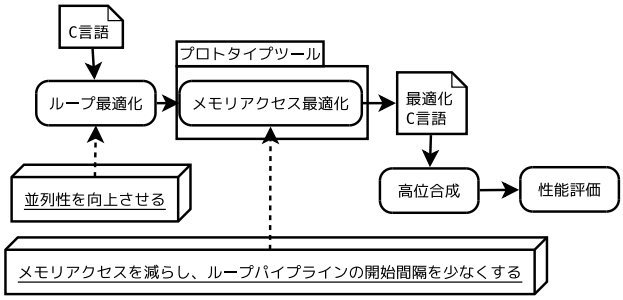



図 1: ハードウェア合成向け C 言語の最適化

## 3　メモリアクセスの最適化

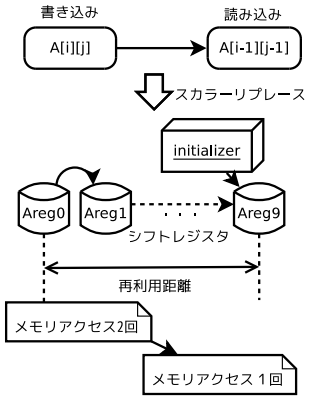
メモリアクセスの最適化にはスカラーリプレイス [4] という手法を用いる。スカラーリプレイスとはレジスタを有効利用することによってメモリアクセスを極力少なくする手法である。通常高位合成では配列はメモリに、変数はレジスタに置き換えられる。つまり配列へのアクセスの数だけメモリにアクセスしていることになり、配列アクセスを減らすことで性能の向上を望めるといえる。手順としては、まず依存解析を行い再利用距離を算出し、次に依存を分類する。さらに、依存の各分類に応じて再利用のためのレジスタ（ローテーションレジスタまたはシフトレジスタ）を用意する。これにより将来再利用されるデータをレジスタに一時保存することで無駄なメモリ再読み出しをなくすことができる。ソースコード 1 とソースコード 2 をみても分

ソースコード 1: スカラーリプレイス前

```
1 for(i=1;i<10;i++){
2  for(j=1;j<9;j++){
3     A[i][j]=C[i][j]+...; //S0
4     B[i][j]=10*A[i−1][j−1];//S1
5  }
6 }
```

ソースコード 2: スカラーリプレイス後

```
1 for(i=1;i<10;i++){
2  for(j=1;j<9;j++){
3     if(i==1||j==1)
4        Areg9=A[i−1][j−1];
5     //initializer
6     Areg0=C[i][j]+...; //S0
7     B[i][j]=10*Areg9; //S1
8     shift_register(Areg0,..);
9  }
10 }
```




図スカラーリプレース

かるとおり配列 *A*[][] に関して再利用が行われていることが分かるこれにより *A*[][] 配列の同時アクセスが 2 から 1 に減っていることは明らかである。本研究ではこの最適化をプロトタイプツールとして実装したので、その実装方法を説明するとともに、例題による評価結果に基づき、有効性を示す。

## 4　プロトタイプツールの概要

図 2 にプロトタイプツールの処理フローを示す。入力は完全ネストループでかつ、for ループの境界条件、if 文の条件式、配列の添字内にループ変数のアフィン式 (線形式）のみを使用して表現されているプログラムである。また、再利用元は 1 つになっているものを対象とする。まず依存解析を行った後、再利用できるデータを表す再利用グラフを作成する。次に、レジスタ置き換えに使用するレジスタ番号を算出するための再利用距離を算出する。再利用距離の算出は n 次元のループイタレーション空間での距離の差を表す配列の依存ベクトルを $d =< d_1, d_2, ..., d_n >$ とすると、

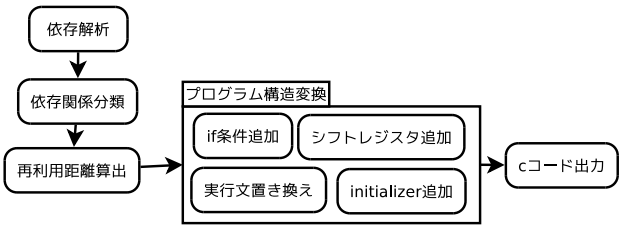



図 2: プロトタイプツールフロー

$$\epsilon(d,n) = \sum_{l=1}^{n-1} \left\{ \left( \prod_{k=l+1}^{n} I_k \right) \times d_l \right\} + d_n,$$

として表せる [4]。$I_k$ は n 次元目のループ回数を表す。再利用距離に基づきシフトレジスタを挿入しデータの満たされていないレジスタをチェックし initializer を挿入する。最後に置き換えを行い、ソースコードを出力する。

## 5 人手最適化が必要な部分

現在のツールは initializer 挿入部分まで実装されているが、initializer もメモリアクセスを含むためループパイプラインの開始間隔が縮められない可能性がある。よって loop peeling と呼ばれる手法を適用し、initializer などを一番内側のループから外に出す事で更なる性能向上を図る必要がある。その例をソースコード 3,4 に表す。

ソースコード 3: peeling 前

```
1 for(t1=1;t1<25;t1++){
2   for(t2=1;t2<25;t2++){
3     for(t3=0;t3<25;t3++){
4       PIPELINE;
5       //start_initializer
6       if(t3==0){
7         a_reg0=tmp1[t1][t2];
8         if(t1==1||t2==1){
9           a_reg25=tmp1[t1−1][t2−1];
10        }
11      }//end_initializer
12          .
13          .
14          .
```

ソースコード 4: peeling 後

```
1 for(t1=1;t1<25;t1++){
2   for(t2=1;t2<25;t2++){
3     //start_initializer
4     a_reg0=tmp1[t1][t2];
5     if(t1==1||t2==1){
6       a_reg25=tmp1[t1−1][t2−1];
7     }
8     //end_initializer
9     for(t3=0;t3<25;t3++){
10      PIPELINE;
11          .
12          .
13          .
```

## 6 実験

今回、４つのループから成る画像処理プログラムをハードウェア化し、効果を検証した。実験フローは、図 1 の通りで、商用高位合成ツールを使用し、配列は外部メモリに割り当て、内側のループはすべてループパイプライン実装を行った。結果を表 1 に表す。cycle はクロックサイクル数、II は開始間隔、area はハードウェア面積（ゲート数）、SR はスカラーリプレイスを適用したかどうかを表す。また RG はレジスタ数、MA はメモリアクセス数を表す。 この結果を見ると最適化前

表 1: 画像処理例題：動作結果

| | cycle | II | area | SR | RG | MA |
|---|---|---|---|---|---|---|
| 最適化前 | 4.1k | 14 | 18.8k | × | 0 | 3708 |
| 最適化後 | 0.3k | 1 | 43.7k | ○ | 132 | 576 |

に比べ合計サイクル数が 13 分の１になっていることが分かる。これはメモリアクセス数の減少によりメモリポートのボトルネックが解消され、並列処理が促進され、開始間隔を１に出来たからである。そしてハードウェア面積が大きくなってしまったのは、レジスタを多く用意したことと、パイプラインによるものである。同様に他の６つの例題を合成し、画像処理例題 (gra) と共にグラフ化した。ここで注目されるのが、int 例題である、この例題はイニシャライザーがボトルネックとなり開始間隔が縮まらなかったために、サイクル数が変わらないという問題が起きている。そこで先の節にかかれている loop peeling を人手で適用した。それにより、面積やメモリアクセス数をスカラーリプレイス後から変化させずに実行サイクル数を 52%まで落とすことができた。結果、プロトタイプツール適用によって短期間で、高性能なハードウェアを生成することができた。メモリアクセス数は 25%以下に抑えられ、実行サイクル数は６０%以下に抑えられている。しかし、ハードウェアコストはレジスタを多く消費してしまう

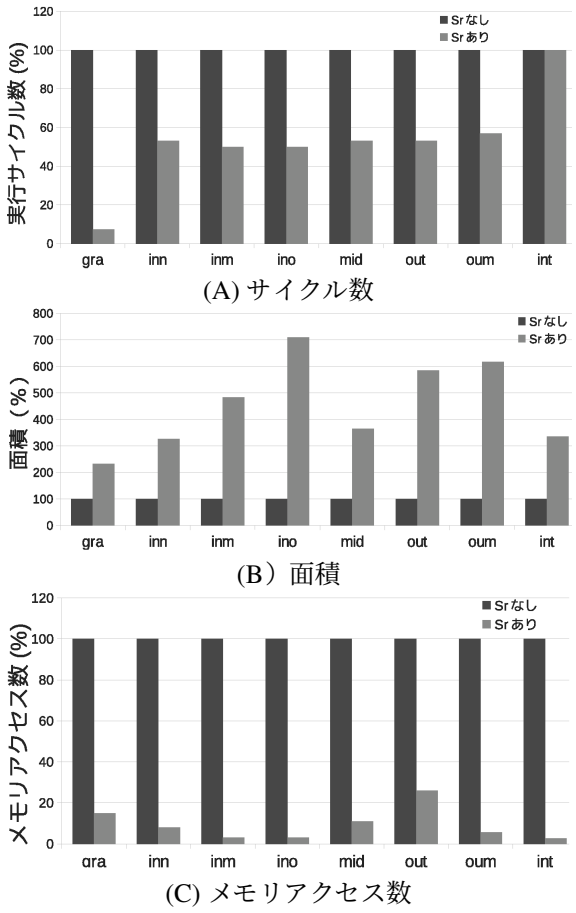



図 3: 合成結果

ため、２～７倍の面積になってしまう。今後はハードウェア面積を抑えつつスカラーリプレイスを行う手法を模索していく必要がある。

## 7 結論

本稿では、高位合成向けメモリアクセス最適化ツールを実装した概要を報告した。またそれを画像処理プログラムに適用し、効果を確認した。性能としてサイクル数を評価した結果パイプライン効率を向上させ開始間隔を１にし 13 倍の処理性能を実現することが出来た。また他の例題においても性能向上を確認した。initializer のボトルネックに対しては人手での loop peeling によって対処した。これにより、すべての例題において実行サイクル数の 40%以上の削減を果たし、メモリアクセス最適化ツールの有効性を示した。

## 参考文献

[1] D.D.Gajski, N.D.Dutt, A.C.-H.Wu and S.Y.-L.Lin, “High-Level Synthesis Introduction to Chip and System Design,” Kulwer Academic Publishers, 1992.

[2] 瀬戸謙修, 田中輝明, 佐々木俊介, 小松聡, 藤田昌宏, “ ループ融合を利用した複数の for ループからのパイプラインハードウェア合成, ”情報処理学会研究報告 2010-SLDM-146(4), 2010

[3] Aho,Alfred,V.Lam,Monica,S.Sethi,Ravi.Ullman,Jeffrey D, ”Compilers Principles, Techniques, And Tools” Pearson Education,Inc 2007

[4] Byoungro So and Mary Hall, “Increasing the Applicability of Scalar Replacement,” 13th International Conference on Compiler Construction , CC 2004.