6J-06

ループタイリング後のスカラリプレイスを可能にするコード変換技術

永澤　圭祐　　瀬戸　謙修

東京都市大学　工学部　電気電子工学科

1　はじめに

設計生産性の向上を目的として、C 言語プログラムから自動でハードウェアを生成する高位合成技術が注目されている。しかし、現状として、高位合成によって高性能なハードウェアを生成するためには高位合成前に人手での最適化が必要となる。そのような最適化としてメモリアクセス最適化技術であるスカラリプレイス[1][3]、またスカラリプレイスによる回路面積増大を抑制するための前処理であるループタイリング[2]がある。しかし、現状では多くの例題でループタイリング後にスカラリプレイスを適用できない問題が発生する。そこで本稿ではその問題を解決する手法を提案する。

2　メモリアクセス最適化技術

高位合成による高性能なハードウェア生成を妨げる要因としてメモリアクセスの競合がある。コード中に同じ配列へのアクセスが複数あるとメモリアクセスに競合が起こり、並列処理を妨げる。よってメモリアクセス最適化は高位合成における重要な課題である。メモリアクセス最適化技術として、スカラリプレイスと呼ばれる技術がある。これはコード中の配列アクセスを一時変数へ置き換え、シフトレジスタを用意し、メモリ内のデータを最初にアクセスする配列アクセス（これをジェネレータと呼ぶ）のデータを再利用することでメモリへのアクセス競合を減らす技術である。例えば、外側ループ変数が i.e.、内側ループ変数が j のループ内に A[i][j]、A[i][j-1]という二つの配列アクセスが存在するとき、A[i][j-1]は A[i][j]が読み出したデータと同じデータを j ループが 1 回実行された後再度読み出す。そこで A[i][j]から読み出したデータを再利用することで A[i][j-1]がメモリにアクセスする必要がなくなるので、効率よくループパイプライン化できる。しかし、スカラリプレイスをそのまま元のコードに適用すると、シフトレジスタが大量に発生し、回路面積が増大してしまう場合がある。そこで、ループタイリングと呼ばれる技術を適用することでシフトレジスタを削減できることが知られている[3]。

3　ループタイリングによる再利用距離削減

再利用距離とは再利用するデータをシフトレジスタに保存後、そのデータを使用するまでに内側ループ換算でループが実行される回数であり、再利用距離が長いほどシフトレジスタが大量に発生してしまう。再利用距離を縮める技術としてループタイリングがある。ループタイリングとはループ構造をタイル状に切り分けるようにループの実行順序を変更し、再利用距離を削減する技術である。図１のような２次元ループを 3 つに分割する例で手順を説明する。まず初期化文、継続条件は内側ループと同一で、1 回実行される毎に（内側ループ回数/3)+1 だけ増える for 文（タイル選択ループ）を最も外側に追加する(図２の１行目)。次に内側ループの初期化文をタイル選択ループのループ変数、継続条件をタイル選択ループのループ変数＋（内側ループ/3）に設定する（図２の３行目）。こうすることによって図 2 のコードになり、図３に示すように再利用距離を 1/3 に縮めることができる。しかし現状では、ループタイリング後にスカラリプレイスを適用するとシフトレジスタに正しいデータが保存されない問題が発生し、適用できる例題が限られている。本稿ではその問題を解決する手法を提案し、ループタイリングの適用範囲を拡張することを目的とする。

```
for(i=0;i<M;i++){
  for(j=0;j<N;j++){
    B[i][j]=A[i][j]+A[i-1][j];
}}
```

図 1:元コード

```
for(ti1=0;ti1<N;ti1+=N/3+1)
  for(i=0;i<M;i++){
    for(i=ti1;i<ti1+N/3;i++){
      B[i][j]=A[i][j]+A[i-1][j];
}}}
```

図 2:ループタイリング適用後コード

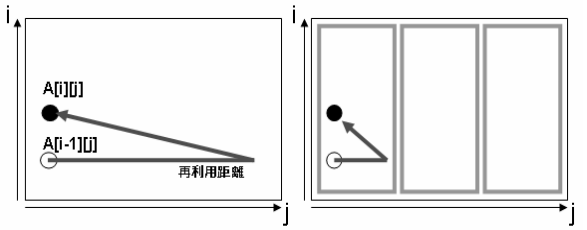


図 3:ループタイリングによる再利用距離縮小

4　タイル拡張

ループタイリング後にスカラリプレイスを適用すると、タイルの境界で再利用するデータをジェネレータが読み込めず、正しいデータがシフトレジスタに保存されない場合がある。

```
for(i=1;i<5;i++){
  for(j=1;j<5;j++){
    B[i][j]=A[i][j]+A[i][j-1];
}}}
```

図 4:元コード

```
for(ti1=1;ti1<5;ti1+=3){
 for(i=1;i<5;i++){
  for(j=ti1;j<ti1+2;j++){
    B[i][j]=A[i][j]+A[i][j-1];
}}}
```

図 5:ループタイリング適用後コード

```
for(ti1=1;ti1<5;ti1+=3){
 for(i=1;i<5;i++){
  for(j=ti1;j<ti1+2;j++){
    A0=A[i][j];
    B[i][j]=A0+A1;
    A1=A0;
}}}
```

図 6:ループタイリング&スカラリプレイス適用後コード

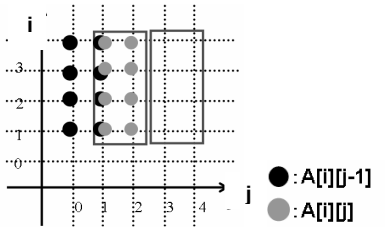


図 7:図 5 におけるタイル内での A[i][j],A[i][j-1]のアクセスパターン

Code Transfomation for Applying Scalar Replacement after Loop Tiling

Keisuke Nagasawa , Tokyo City University Faculty of Engineering Electrical and Electronic Engineering

Kenshu Seto , Tokyo City University Faculty of Engineering Electrical and Electronic Engineering

例として、図４のコードにループタイリングを適用すると図５のコードになり、図５にそのままスカラリプレイスを適用すると図６のコードになる。この時、図７を見るとジェネレータ(A[i][j])が読み込む範囲と A[i][j-1]の読み込む範囲を比較すると A[1][0],A[2][0],…A[4][0]を A[i][j]は読み込んでおらず、A[i][j-1]が読み込むデータの一部がシフトレジスタに保存されない。よって、図６で A[i][j-1]を一時変数に置き換えた際に正しいデータが再利用されず正しい計算結果が出力されない。この問題は例えば A[i][j-1]、A[i][j]や B[i][j+2],B[i][j-1]のような同じ配列への read アクセスで内側ループ変数を使用した添え字の定数部分が異なる場合に必ず発生する(A[i+1][j]、A[i][j]のように外側ループ変数を使用した添え字の定数部分が異なる場合は既存技術であるスカラリプレイスにおける初期化[1]を適用するだけでよい)。そこで本稿では解決策としてタイル拡張を提案する。これはループの範囲を拡張し、ジェネレータの読み込む範囲を広げることでシフトレジスタに正しいデータを保存できるようにするスカラリプレイスにおける初期化[1]を切り分けたタイル全てに適用する手法である。ジェネレータから見て、同じループ変数を使用した添え字の定数部分の差が一番大きい同配列への read アクセスまでの距離分だけタイルを拡張する（A[i][j]と A[i][j-1]では j 方向に-1 拡張）。また、タイルを拡張したことでタイルの一部が隣と重なってしまう。このままプログラムを実行すると、重なった部分は計算を２回行うことになり、正しい結果が得られない。そこで、計算部分のみを if 文で本来そのタイルで計算を行うべき範囲のみで計算を実行するように制御する。これらを図５に適用した後にスカラリプレイスを適用したコードが図８である。図９を見ると、ジェネレータが A[i][j-1]が読み込むデータを全て読み込んでいることが分かる。

```
for(ti1=1;ti1<5;ti1+=3){
 for(i=1;i<5;i++){
  for(j=ti1-1;j<ti1;j++){
    A0=A[i][j];
    if(j=>t1) B[i][j]=A0+A1;
    A1=A0;
}}}
```

図 8:図 5 にタイル拡張適用後にスカラリプレイス適用

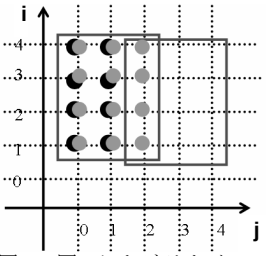


図９　図８におけるタイル内での A[i][j],A[i][j-1]のアクセスパターン

これによって今までループタイリング適用後にスカラリプレイスを適用できなかった例題も最適化できる。

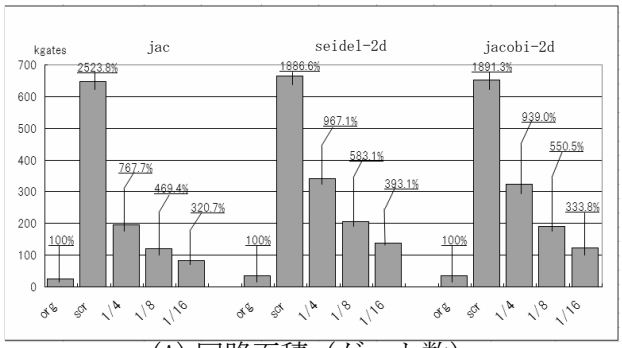
5　実験

ループタイリングツールを作成し、従来ループタイリング後にスカラリプレイスが適用できなかった 3 つの例題に対し図 10 のフローでループタイリング、スカラリプレイスを適用し、商用高位合成ツールでハードウェアを生成し、性能比較を行った。また、ライブラリは 45nm を使用、クロック制約は 10ns に設定した。スカラリプレイスの際に発生するシフトレジスタは内部メモリとして実装した。

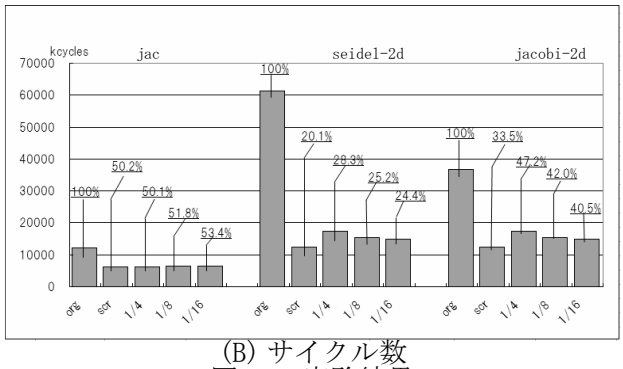
図 10　実験フロー

本実験で、ループサイズは 640×480 とした。また内部メモリは全てシングルポートで実装した。図１１については org がスカラリプレイスなし、scr はスカラリプレイスのみ適用、1/4.1/8,1/16 はループタイリング後にスカラリプレイスを適用した結果であり、数字は元の内側ループを１としたときのタイルサイズである。

図１１を見ると、最も効果が大きかった jacobi-2d は 1/16 の時、org を 100%として回路面積は 333.8%に収まり、サイクル数は org を 100%として 40.5%となりスカラリプレイスのみと比べても 7%の増加に収まった。他の例題も回路面積増加を抑えつつ高速化を実現できている。したがって、提案手法を使用してループタイリング、スカラリプレイスを適用することで、最適化前と比べてサイクル数を削減し、尚且つスカラリプレイスによる面積増加を抑制できることが示された。



(A) 回路面積（ゲート数）



(B) サイクル数

図 11　実験結果

6　結論

本稿では、複数の同配列へのアクセス（read アクセスのみ）が存在し、それらの内側ループ変数を使用した添え字の定数部分がそれぞれ異なる場合、再利用距離削減技術を適用するとメモリアクセス最適化技術が適用出来なくなる問題を解決した。また、その解決手法を使用して例題に適用し、高位合成を行ったところ、16 分割した場合、最も回路面積増加が抑えられたもので回路面積は最適化なしの場合の 333.8%に収まりサイクル数は最適化なしの場合の 40.5%に削減された。これらのことから、提案手法を適用してもそれぞれの効果（サイクル数削減、面積増加抑制）が出ることを確認した。

7　参考文献

[1] 竹鼻宏晃,瀬戸謙修, “配列アクセス実行条件の厳密な解析に基づくスカラリプレイス技術” デザインガイア2012 -VLSI設計の新しい大地- VLD2012-60, pp. 7-12, 2012,11

[2] 望月俊啓, 瀬戸謙修, “高位合成前のループタイリングによる性能制約下でのレジスタ数削減” DAシンポジウム2012 －システムLSI設計技術とDA－ 2012,8

[3] Byoungro So and Mary Hall, ”Increasing the Applicability of Scalar Replacement,” CC 2004, LNCS 2985, 2004, pp. 185-201.