6J-07

複雑な添え字を持つ配列アクセス向けスカラリプレイス技術

外山 知人　　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

高位合成技術[1]はループパイプライン化によりループ記述に対する高性能なRTL記述をCプログラムから自動生成できる。しかし配列アクセス回数が多いループ記述では多数のメモリアクセスのためパイプライン化による並列性が悪化し性能制約を満足するRTL記述を生成できない場合が多く通常Cプログラムの最適化が必要となる。ループパイプライン化による並列性向上のためのメモリアクセス最適化としてループ不変式の移動技術、スカラリプレイス技術[2]、メモリ分割技術[3]があり、本稿ではループ不変式の移動、スカラリプレイスを組み合わせて使用する。ループ不変式の移動は配列アクセスの添字に最内ループのループ変数が含まれていない場合、その配列アクセスを外側のループに出すことで最内ループでの配列アクセス数を削減できる。スカラリプレイスはシフトレジスタを用いてデータを再利用し配列アクセスを削減できる。その際に配列アクセス間で各次元の添字を引き算し再利用ベクトル(A[i][j],A[i][j-1]⇒<0,1>)を求める。さらに再利用ベクトルとループ回数を用いて最内ループ換算で何回実行された後にデータ再利用を行うかを表す再利用距離[2]を求める必要がある。しかしスカラリプレイスには再利用距離にループ変数を含む場合適用できないという問題がある。本稿は再利用距離にループ変数が含まれる場合に配列アクセスを複数の再利用グループに分けることでこれを解決しループ不変式の移動と併せて配列アクセスを削減した。

2　スカラリプレイス技術の問題点

スカラリプレイスは再利用距離にループ変数を含む場合適用できないという問題がある。再利用距離にループ変数が含まれている場合、最内ループが何回実行された後にデータを再利用するのか判断できないからである。例えばA[i][j]とA[i][k]という配列アクセスが存在する場合に再利用ベクトルを求めると<0,j-k>となり、再利用距離にループ変数が含まれることが分かる。またループ回数が外側のループ変数に依存する場合for(j=0;j<i;j++))のB[i][j]とB[i-1][j]の再利用ベクトルは<1,0>となり、内側ループ換算の再利用距離は 1×i = iとなる。このような場合も再利用距離にループ変数を含んでしまう。３節でこの問題に対する解決策を提案する。

3　グループ分けによるスカラリプレイス

提案手法では、再利用距離にループ変数が含まれないように配列アクセスのグループ分けを行い、それぞれのグループにスカラリプレイスを適用することで、配列アクセス数を削減する。まずグループ分けを行う手順を図１に示す。また図１の手順で提案するスカラリプレイス技術を適用したソースコード 1,2 を図２に示す。まず配列アクセスの各次元に含まれる変数が異なるかを判定する。ソースコード１の B 配列では１次元目の添え字に[i]を含んでいるが、２次元目では[j]を含んでいる配列アクセスと[k]を含んでいる配列アクセスが存在するので、true となる。配列アクセスの添字に含まれている変数パターン(１次元目 i、２次元目 j のパターンと１次元目 i、２次元目 k の２パターン)(B[i][j-1],B[i][j]),(B[i][k],B[i][k-1])にグループが分けられる。ソースコード２ではすべての配列アクセスの１次元目に[i]２次元目に[j]が含まれているので、1 つだけグループを作成する。次に再利用ベクトルより内側ループのループ回数に変数が含まれているか判定する。ソースコード１ではループ回数がすべて定数になっているので else。ソースコード２では B[i][j]、B[i-1][j]との間に発生する再利用ベクトルが<1,0>となり、さらに i ループの内側に存在する j ループがループ回数 i となっているので、true となる。i ループを含んでいる次元の添字パターンだけグループ分け([i]と[i-1])を行うと(B[i][j],B[i][j],B[i][j])(B[i-1][j])とグループが分けられる。

```
start
 if(配列アクセスの各次元に含まれるループ変数が異なる
 (A[i][j],A[i][k]⇒true))
  配列アクセスの各次元の添字に含まれている変数パターンだけ
  再利用グループ作成
  (A[i][j],A[i][k],A[i][k+1]⇒[i][j],[i][k]の2グループ作成)
 else
  1つだけ再利用グループ作成
  (A[i][j],A[i+1][j],A[i-1][j]⇒[i][j]の1グループ作成)
 if(再利用ベクトルより内側ループのループ回数が
 変数(<1,0>,内側ループ回数=i⇒true))
  外側ループを含んでいる次元の添字パターンだけグループ分け
  (A[i][j],A[i+1][j],A[i-1][j]⇒[i],[i+1],[i-1]の3グループ作成)
 else
  nop
end
```

図１：グループ分けフローチャート

次にグループ毎にスカラリプレイスを行う。スカラリプレイスは１節で説明した手順で行う。スカラリプレイスを行った配列アクセスに対してループ不変式の移動を適用できる場合、不変式の移動を行う。ソースコード１のスカラリプレイス後では内側ループで配列アクセス数を４から２に削減できていることが分かる。しかし手法適用により図３に示すようにデータの一貫性が損なわれてしまう。例えば i=3,k=3 について考えると、ソースコード１のスカラリプレイス後３行目より B1_0 は B[3][3]のデータを取得する。次に j=4 となったときにソースコード１のスカラリプレイス後７行目より B0_1 は B0_0+temp の値を取得する。本来であれば B[3][3]が書き換えられ、B1_0 で使用するデータが書き換えられる必要がある。しかし、B1_0 を書き換える記述がないため、スカラリプレイス後ではデータの一貫性が損なわれ、スカラリプレイス前と結果が異なってしまう。このデータ一貫性の喪失を回避するために書き込みアクセスがあるグループとその他のグループにある配列アクセスでデータの整合性を取る必要がある。図３よりデータ一貫性の喪失は同一アドレスを参照する際に発生することが分かる。よって同一アドレスを参照したとき、データ書き込み直後にその他のグループにある配列アクセスにもデータを書き込む

Scalar Replacement for Array Accesses with Complex Subscripts
Toyama Tomohito , Tokyo City Unversity Faculty of Engineering Electrical and Electronic Engineering
Seto Kenshu , Tokyo City Unversity Faculty of Engineering Electrical and Electronic Engineering

ことによりデータ一貫性の喪失を回避することができる。ソースコード１のスカラリプレイス後７行目の B0_1 書き込み直後に図４のような if 文を挿入しデータ一貫性の喪失を回避する。

ソースコード1

```
for(k=0;k<32;k++){
  for (i=0; i <33; i++){
    for (j=0; j <17; j++){
      if(i>=1&&j>=1){
        B[i][j-1]=B[i][j]+temp;
        C[i][j]=B[i][k]+B[i][k-1];
}}}}
```

↑スカラリプレイス前

スカラリプレイス後→

```
for(k=0;k<32;k++){
  for (i=0; i <33; i++){
    B1_0 = B[ i ][ k ];
    for (j=0; j <17; j++){
      B0_0 = B[ i ][ j ];
      if(i>=1 && j>=1){
        B0_1 = B0_0 + temp;
        C[ i ][ j ] = B1_0 + B1_33;
      }
      B[ i ][ j-1 ] = B0_1;
      shift_registerB0;
    }
    shift_registerB1;
}}
```

ソースコード2

```
for (i=0; i <33; i++){
  for (j=0; j < i; j++){
    if(i>=1&&j>=1){
      B[i][j]=B[i][j]+temp;
      C[i][j]=B[i-1][j]+B[i][j];
}}}
```

↑スカラリプレイス前

スカラリプレイス後→

```
for (i=0; i <33; i++){
  for (j=0; j < i; j++){
    B0_0 = B[i][j];
    B1_0 = B[i-1][j];
    if(i>=1&&j>=1){
      B0_0=B0_0+temp;
      C[i][j]=B1_0+B0_0;
      B[i][j]=B0_0;
}}}
```

図２：ソースコード１，２に手法を適用した結果

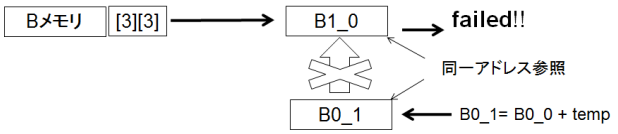

図３：データ一貫性の喪失の様子

```
if( i == i && k == j-1){    //[ i ][ k ]==[ i ][ j-1 ]
  B1_0 = B0_1;
}
```

図４：データ一貫性の喪失を回避するための実行文

この例題ではグループ分けスカラリプレイス、ループ不変式の移動、データ一貫性の喪失回避、を行うことにより、開始間隔を４から２に削減することができた。

4　実験

今回再利用距離に変数を含んでいるため既存スカラリプレイス[2]が適用できない５つのベンチマークプログラムに対して提案手法を適用し効果を検証した。商用高位合成ツールを使用し、配列は内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化を行った。もとのコードを最適化なし、提案手法を適用したコードを手法適用後としたときの結果を表１及び図５に示す。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）を表していて、クロック制約は 10.0ns とした。ludcmp は他の例題に比べて配列アクセス数が多いためメモリをデュアルポートで実装し、その他の例題ではシングルポートで実装した。なお手法適用前後でメモリ面積に変化はないので、ハードウェア面積にメモリ面積は含めていない。図５より手法適用後は最適化なしに比べて、平均で面積 31%増加に対してサイクル数が 49%削減されていることがわかる。面積増加理由はシフトレジスタの追加、if 文によるマルチプレクサの増加、開始間隔削減によるリソースの増加などによるものである。サイクル数の削減理由は配列アクセスの削減により、開始間隔が削減され、ループパイプライン化の効果が増大したことによる。

表 1：提案手法適用前後での論理合成結果

		Ⅱ	cycle (kcycles)	area (kgate)
ludcmp	最適化なし	7	1210	6.93
	手法適用後	2	370	11.8
floyd warshall	最適化なし	4	4.49	1.20
	手法適用後	3	3.10	1.12
3mm	最適化なし	4	2.88	10.1
	手法適用後	3	1.72	14.1
cholesky	最適化なし	4	24.5	2.87
	手法適用後	2	13.6	3.99
trisolve	最適化なし	3	51.7	3.96
	手法適用後	1	19.0	4.41

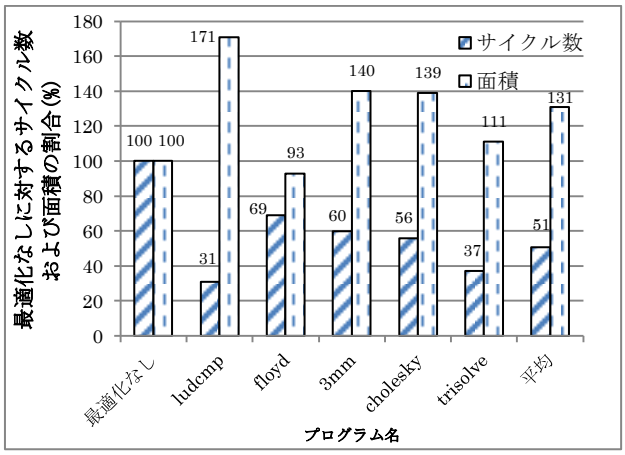

図５：提案手法適用前後での論理合成結果

5　結論

本稿では、複雑な添え字を持つ配列アクセス向けスカラリプレイス技術を提案した。再利用距離がループ変数を含む場合、配列アクセスをグループ分けすることでスカラリプレイスを実現し、ループ不変式の移動と併せて配列アクセス数を削減した。さらにグループ分けにより発生するデータ一貫性の喪失を if 文により回避した。既存スカラリプレイスが適用できない 5 つのベンチマークプログラムに手法を適用し平均で面積増加 31%に対して 49%サイクル数削減を果たし、提案手法の有効性を示した。

参考文献

[1] Daniel D. Gajski, "High Level Synthesis: AnIntroduction to Chip and System Design," Kluwer Academic Publishers, 1992

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

[3] Yuxin Wang, Peng Li, Peng Zhang, Chen Zhang, Jason Cong, "Memory partitioning for multidimensional arrays in high-level synthesis," Proceedings of the 50th Annual Design Automation Conference