Vol.2021-SLDM-195 No.11
2021/10/15

高位合成を用いたハードウェア設計における三角ループ向けスカラリプレイス

坂部光 [1, a)]　瀬戸謙修 [1]

概要: 高位合成は高級言語から RTL 記述に自動変換する技術である. しかし, 高位合成により高性能なハードウェアを生成するには, 高位合成前に人手によるコードの最適化が必要となることが多い. そのような最適化法として, メモリアクセス最適化の一つであるスカラリプレイスが存在する. 三角ループに対してスカラリプレイスの適用を試み, 再利用距離が一定と不定の場合に分類されることを確認した. そこで, 本稿では各々の場合についてスカラリプレイスを適用する方法を提案する.

キーワード: 高位合成, メモリアクセス最適化, スカラリプレイス

1. はじめに

高位合成は, C 言語等から RTL（Register Transfer Level）記述を自動生成する技術である. しかし, 高位合成により高性能なハードウェアを生成するには, コーディング時に人手による最適化が必要となることが多い. 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阻む. 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の一つにスカラリプレイス[1]と呼ばれる技術がある. これは配列アクセスを一時変数へのアクセスに置き換え, シフトレジスタを用いて最初にアクセスされる配列データを再利用することによりメモリへのアクセス競合を減らす技術である. 本稿では, 三角ループ[2]にスカラリプレイスを適用する方法を検討する.

2. 三角ループのスカラリプレイス

現状のスカラリプレイス（以下 SR）は, 内側ループのループ回数が外側ループのループ変数に依存しないループ構造のみ対応している. 本稿では特に, 図 1 に示すようなループ構造（以下三角ループと呼ぶ）に SR を適用する手法を検討する. 三角ループは, 内側ループのループ回数が外側ループのループ変数に依存するループ構造である. ループ変数 i がインクリメントされる度に j の反復範囲が変化する.

三角ループの SR では, 再利用距離が一定と不定の場合に分類される. 再利用距離とは, メモリから読み込みしたデータを何回後のループ実行時に再利用するかを意味する. 例えば, 図 1 の配列アクセス A[i][j+1]でアクセスされたデータは一回後のループ実行における配列アクセス A[i][j]で再利用されるため, 再利用距離は 1 となる.

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

図 1　三角ループのコード

2.1 節では一定, 2.2 節では不定の場合について SR の適用方法を述べる.

2.1 再利用距離が一定の場合

図 1 は再利用距離が一定となる三角ループのコードである. 配列アクセス A[i][j+1]でアクセスされたデータが, 最内側ループ j が 1 進むと A[i][j]でアクセスされるため, 再利用距離は常に一定の 1 となる. 再利用距離が一定の場合には, 従来の SR[1]を適用でき, SR を適用したコードを図 2 に示す. 以下では, 図 2 のコードの動作を説明する. 図 2 の 2 行目に示すように, ループ回数を拡張する. 図 1 は, 配列 A へのアクセスを二つ持ち, 一回のループで A[i][j]と A[i][[j+1]の二つのデータを読み込むため, メモリアクセス回数は二回となる. それに対し, 図 2 では, メモリアクセス回数を削減するため, 配列 A を一つにする. よって, 一回のループでのメモリアクセス回数は一回となる. そのため, 図 2 は二回のループで二つのデータを読み込み, 演算を開始する. 例えば図 1 では, 一周目のループは(i,j)=(0,0)となる. この時, 配列 A の添字は A[0][0]と A[0][1]となり, 二つのデータを読み込む. しかし, 図 2 では, メモリアクセス回数を削減するため, 配列 A を一つにした. そのため, 一周目(i,j)=(0,0)では A[0][0]の一つしか読み込みできない. 二周目(0,1)で A[0][1]を読み込む. この時, 演算に必要な二つのデータが揃い, 演算を開始する. つまり, 図 1 は一度のループで, 図 2 は二度のループで演算を実行している. したがって, ループ回数を拡張する必要がある.

```
 1: for(i=0; i<3; i++){
 2:     for(j=0; j<=i+1; j++){
 3:         reg0 = reg1;
 4:         reg1 = A[i][j];
 5:         if(j>0) {
 6:             i1 = i;
 7:             j1 = j-1;
 8:             B[i1][j1] = reg0 + reg1;
 9:         }
10:     }
11: }
```

図 2　再利用距離が一定の場合に SR を適用したコード

1 東京都市大学
a) g2181229@tcu.ac.jp

次にデータの再利用について説明する. 前述したループ回数の拡張は処理時間の増大に繋がるが, シフトレジスタを用いたデータの再利用により処理時間の削減に繋がる. 例えば, 図 1 では二周目(1,0)で A[1][0]と A[1][1]を, 三周目(1,1)で A[1][1]と A[1][2]をメモリから読み込む. 二周目と三周目で A[1][1]を二回読み込むことになる. 図 2 では A[1][1]を一回の読み込みのみで済ます. そのためには, 一度メモリから読み込みした A[1][1]をレジスタに格納し, 次のループで再利用する. 図 2 では 3, 4 行目の変数 reg0 と reg1 がシフトレジスタを表す. 図 2 では, 三周目(1,0)で読み込みした A[1][0]は reg1 に格納される. 次に四周目(1,1)で reg0 に reg1 が, reg1 に A[1][1]が格納される. この時, 演算に必要なデータが二つ揃い, 演算は実行される. 次のループで A[1][1]の再利用が行われる. 五周目(1,2)で reg0 に reg1 が, reg1 に A[1][2]が格納される. この時, A[1][1]は reg0 から取得され, 演算は実行される. したがって, 四周目はメモリから読み込みした A[1][1]で, 五周目はシフトレジスタから取得した A[1][1]で演算を実行する. データの再利用により, メモリアクセスの競合は減少し, 処理時間は短縮する.

続いて, 5 行目の if 文について説明する. if 文は, 演算を実行するタイミングを制御している. このタイミングは, 演算に必要な配列データが二つ揃う時である. 例えば, 図 2 では二周目(0,1)で A[0][1], 三周目(1,0)で A[1][0]を読み込むが, 図 1 において A[0][1]+A[1][0]という演算は存在しない. よって, 不要な演算を実行させないため, if 文を用いる. 図 2 の場合, j>0 の時に演算に必要な二つのデータが揃う.

最後に配列の添字を本来の値に更新する. 図 1 のループ変数は(i,j)=(0,0), (1,0), …, (2,2)と変化するため, 配列 B の添字は, B[0][0], B[1][0], …, B[2][2]となる. しかし, 図 2 の if 文の条件を満たすループ変数は(0,1), (1,1), …, (2,3)となり B[0][1], B[1][1], …, B[2][3]となるため, 図 1 の配列 B の添字と異なる. したがって, 図 1 と同じ添字にするため, 6, 7 行目による添字の更新を行う. 今回の場合, j を-1 すると, 本来の添字と同じ値となる.

2.2 再利用距離が不定の場合

図 3 に再利用距離が不定となる三角ループのコードを示す. 図 3(a)では, 一周目(0,0)にアクセスする A[i+1][j]は二周目(1,0)に A[i][j]で再利用されるため, 再利用距離は 1 となる. しかし, 二周目(1,0)にアクセスする A[i+1][j]は四周目(2,0)に A[i][j]で再利用されるため, 再利用距離は 2 となる. 周回数によって, 再利用距離が異なる. この状態では必要なレジスタの数が変動するため, SR をそのまま適用できない. 従来の SR[1]を適用するには, 再利用距離を一定にしなければならないが, ループの順番を変換することにより解消できる. 図 3(a)のコードに対して, 内側ループと外側ループを入れ換えることによって, 再利用距離は一定となる. 一定に変換したループ構造が図 3(b)である. これにより, 2.1 節と同様に SR の適用が可能となる.

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

(a)元のコード

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

(b)ループの順番変換後

図 3 再利用距離が不定となる三角ループのコード

3. 実験結果

商用高位合成ツールを使用し, 高位合成, 論理合成後のサイクル数, 面積を評価した. また, 配列アクセスはシングルポートメモリで行い, ループパイプライン化をした.

例題は 8 近傍ラプラシアンフィルタ（8lap）, モーションフィルタ（motion）, ソーベルフィルタ（sobel1, sobel2）を用いた. 各例題の再利用可能な配列アクセス間の再利用距離は, 8lap が一定, motion が不定である. なお, ソーベルフィルタについては, 一定と不定の両方を含む. sobel1 は一定のみに SR, sobel2 は不定のみに SR を施した. 各例題を高位合成し, 論理合成した結果を表 1 に示す. 表 1 は SR 前のコード（original）を分母とした百分率を示す. original に比べ, サイクル数は平均 30%減少, 面積は平均 17%増大した. 提案手法を適用した各例題の配列アクセスは, 8lap は 8 から 3, motion は 3 から 1, sobel1 は 12 から 8, sobel2 は 12 から 8 に削減できた. このため, メモリアクセスの競合は減少し, サイクル数は低減した. ソーベルフィルタは, 一定と不定の両方に SR の適用が可能となれば, 更なるサイクル数の削減が期待できる.

表 1 高位合成結果

	original	8lap	sobel1	motion	sobel2	average
サイクル数[%]	100	53	78	71	77	70
面積[%]	100	89	140	98	140	117

4. おわりに

本稿では, 三角ループ中の配列アクセスについて, 再利用距離を一定と不定に分類し, スカラリプレイス（SR）を適用した. 一定の場合, SR は適用可能であるが, 不定の場合, ループの順番を変換することにより再利用距離を一定にし, SR を適用した. 提案手法により, サイクル数は平均 30%減少, 面積は平均 17%増大した. 面積が増大した割合よりサイクル数が減少した割合の方が大きいため, この手法は有効である. 一定と不定の両方を含む場合, いずれも SR の適用が可能となれば, 更なるサイクル数の削減が期待できる.

参考文献

[1] Kenshu Seto, “Shift Register Initialization in Scalar Replacement for Reducing Code Size”, IPSJ Transactions on System LSI Design Methodology Vol.13 2–9 (Feb. 2020)

[2] Hyeonuk Sim, et, “Efficient High-Level Synthesis for Nested Loops of Nonrectangular Iteration Spaces”, IEEE TANSACTIONS ON VERY LATGE SACALE INTEGRATION (VLSI) SYSTEMS, VOL. 24, NO. 8, AUGUST 2016