SIMD命令を生成するコンパイル手法の提案

2N－6

酒井淳嗣　枝廣正人　小長谷明彦

NEC C&C研究所

1　はじめに

マルチメディア向けにSIMD命令と呼ばれる命令セットを有するマイクロプロセッサが登場しつつある．本稿では，C等の高級言語で記述されたプログラムからSIMD命令を生成するためのコンパイル手法を提案する．この手法は，ソースプログラム中で配列操作を行うループ構造に対してループアンローリング及び命令並べ換えを施し，その結果隣接する同種の命令列を1つのSIMD命令に置き換えることを特徴としている．命令並べ換えは2つの順序数を付与することで容易に行うことができる．

2　ターゲットプロセッサと中間コード

本稿では，64bit整数レジスタセットを持ち，以下のようなSIMD演算命令を備えている汎用的なRISC風プロセッサをターゲットとする．

- 単純SIMD演算命令 — ソースレジスタ r_S と r_T をそれぞれ16bitのフィールド4つに区切り，対応する各々のフィールド毎に算術論理演算を行った結果を，デスティネーションレジスタ r_D の各々のフィールドに格納する命令．
- マスク付きSIMD演算命令 — マスクレジスタと呼ばれるレジスタ r_M の各フィールドを参照し，その値が0か否かによって，ソースレジスタ間のSIMD演算結果を破棄するかデスティネーションレジスタに格納するかを決定するSIMD演算命令．
- マスク生成命令 — ソースレジスタの各フィールドの値を0と比較し，その結果によってデスティネーションレジスタの対応するフィールドに1又は0を設定する命令．マスクレジスタに値を設定するのに用いる．
- 総和演算命令 — ソースレジスタの各フィールドの値の総和を求めてデスティネーションレジスタに格納する命令．
- 拡張転送命令 — ソースレジスタの下位16bitの値をデスティネーションレジスタの各フィールドに複写する命令．

そして，これらの命令に対して以下のような中間コード表記を用いる．

$r_D \leftarrow r_S$	r_S の値を r_D に格納．
$r_D \leftarrow r_S \circ r_T$	r_S と r_T 間で演算 $\circ$ を行った結果を r_D に格納．
for r_i... begin ··· end	r_i を制御変数とし，··· を本体とするループ構造．
if r_C ···	r_C が真の場合のみ ··· を実行．

ここで r_n は中間項(仮想レジスタ)，変数あるいは定数を表す．実際の中間コードでは通常の中間項は $r_1, r_2, r_3, \cdots$，SIMD型中間項は $r_{S1}, r_{S2}, r_{S3}, \cdots$ のように表記する．

3　SIMD化

SIMD化処理の中心部分について，図1に示す簡単な例を用いて説明する．図2はこのソースプログラムから得られた中間コードである．なお本稿では16bit×4のSIMD命令のみを扱うが，本手法の適用範囲はそれに限定されるものではない．

```
for(i=0;i<16;++i) {
  D[i]=S[i]*5;
}
```

図1　ソースプログラム

```
for r1←0 to 15 begin
    r2 ← S[r1]
    r3 ← r2 * 5
    D[r1] ← r3
end
```

図2　入力中間コード

3.1　前処理

前処理ではループの正規化と帰納変数の展開処理を行った後，ループ内の配列要素参照の参照間隔がSIMD命令に変換可能なものであるかどうかを，添字式を解析して判断する．SIMD化可能であるためには等間隔アクセスでなくてはならず，しかもその間隔が1又は2のものが望まれる．この他，総和演算などのイディオム処理も行う．

A Compiler Technique with SIMD Instructions
Junji Sakai, Masato Edahiro, and Akihiko Konagaya
C&C Research Laboratories, NEC Corporation

3.2 ループアンローリング

ループを4段にアンローリングする．その際，各中間コードに対して以下のような番号付けを行う．

まずアンローリング前にループ内の各中間コードに対してテキスト順に1, 2, 3,･･･の番号を振る．これを第1のSIMD化順序数と呼ぶ．次にループを4段にアンローリングするが，k回目のアンローリングでは各中間コードに第2のSIMD化順序数kを振る．

アンローリング後の中間コードのうち，ループ本体の部分を図3に示す．左端の2つの数値が第1，第2のSIMD化順序数である．

1	1	$r_{101} \leftarrow r_1 + 0$
2	1	$r_2 \leftarrow$ S[r_{101}]
3	1	$r_3 \leftarrow r_2 * 5$
4	1	D[r_{101}] $\leftarrow r_3$
1	2	$r_{102} \leftarrow r_1 + 1$
2	2	$r_4 \leftarrow$ S[r_{102}]
3	2	$r_5 \leftarrow r_4 * 5$
4	2	D[r_{102}] $\leftarrow r_5$
1	3	$r_{103} \leftarrow r_1 + 2$
2	3	$r_6 \leftarrow$ S[r_{103}]
3	3	$r_7 \leftarrow r_6 * 5$
4	3	D[r_{103}] $\leftarrow r_7$
1	4	$r_{104} \leftarrow r_1 + 3$
2	4	$r_8 \leftarrow$ S[r_{104}]
3	4	$r_9 \leftarrow r_8 * 5$
4	4	D[r_{104}] $\leftarrow r_9$

図3 アンローリング後

1	1	$r_{101} \leftarrow r_1 + 0$
1	2	$r_{102} \leftarrow r_1 + 1$
1	3	$r_{103} \leftarrow r_1 + 2$
1	4	$r_{104} \leftarrow r_1 + 3$
2	1	$r_2 \leftarrow$ S[r_{101}]
2	2	$r_4 \leftarrow$ S[r_{102}]
2	3	$r_6 \leftarrow$ S[r_{103}]
2	4	$r_8 \leftarrow$ S[r_{104}]
3	1	$r_3 \leftarrow r_2 * 5$
3	2	$r_5 \leftarrow r_4 * 5$
3	3	$r_7 \leftarrow r_6 * 5$
3	4	$r_9 \leftarrow r_8 * 5$
4	1	D[r_{101}] $\leftarrow r_3$
4	2	D[r_{102}] $\leftarrow r_5$
4	3	D[r_{103}] $\leftarrow r_7$
4	4	D[r_{104}] $\leftarrow r_9$

図4 命令並べ換え後

3.3 命令並べ換え

アンローリングされたループ本体の各中間コード間の依存関係を解析し，次にループ本体の中間コードをソートする．ソートの第1，第2のキーは各々第1，第2のSIMD化順序数であり，これらの昇順にソートする．これは中間コード列を逐次実行順序からSIMD実行順序に並べ換えることを意味する．図3の中間コードを並べ換えたものを図4に示す．

3.4 SIMD命令への置換

命令並べ換えの結果，中間コード列はSIMD実行順に並んでいる．1つのSIMD演算は4つの中間コードに対応し，これらの中間コードはオペランドの番号が異なる以外はオペコード等同一の形式を持っている．そこで中間コード列を先頭から走査し，連続する4つの中間コードを1つのSIMD型中間コードに置き換えていく．SIMD型中間コードは通常の非SIMD型中間コードと同様の構造であるが，オペランドレジスタに対して各フィールド毎にSIMD演算を施す点が異なる．これらのSIMD型中間コードは，ターゲットプロセッサのSIMD演算命令に容易に対応付けられる．図4の中間コードをSIMD型中間コードへ置き換えたものを図5に示す．

$r_{S1} \leftarrow r_1 + [0, 1, 2, 3]$
$r_{S2} \leftarrow$ S[r_{S1}]
$r_{S3} \leftarrow r_{S2} * [5, 5, 5, 5]$
D[r_{S1}] $\leftarrow r_{S3}$

図5 SIMD中間コードへ置換後

3.5 後処理

無用演算 / 共通部分式の削除，ループ不変式のループ外移動，演算強度の軽減等，従来からの最適化手法を施す．最適化後の中間コードを図6に示す．

$r_{S6} \leftarrow [4, 4, 4, 4]$
$r_{S7} \leftarrow [0, 1, 2, 3]$
for $r_1 \leftarrow 0$ to 15 step 4 begin
　$r_{S2} \leftarrow$ S[r_{S7}]
　$r_{S3} \leftarrow r_{S2} * [5, 5, 5, 5]$
　D[r_{S7}] $\leftarrow r_{S3}$
　$r_{S7} \leftarrow r_{S7} + r_{S6}$
end

図6 最終中間コード

4 おわりに

ループアンローリングと命令並べ換えに基づくSIMD化手法を提案した．基本的なコンパイル手法の有効性は確認できたが，適用範囲をより広げるためには以下のような点を検討していく必要がある．(1) ループ制御変数の初期値 / 終値がフィールド数の倍数でない場合や非定数の場合への対処．(2) メモリ上のワード境界以外から開始する連続要素アクセスをSIMD処理する場合への対処．(3) SIMD化できない文を含むループの部分的なSIMD化．(4) 実際にアンローリングせずに直接SIMD命令に変換する方法．

今後はこれらの課題に対して検討を加えていくとともに，実際のコンパイラ内に実装して詳細な評価を行う予定である．

参考文献

[1] A.V.Aho, R.Sethi, J.D.Ullman 原著, 原田賢一訳: “コンパイラ I,II,” サイエンス社, 1990

[2] H.Zima, B.Chapman 原著, 村岡洋一訳: “スーパーコンパイラ” (“Supercompilers for Parallel and Vector Computers”), オーム社, 1995