

# SIMD 並列化

鈴木 貢 (電気通信大学情報工学科)

藤波順久 ((株) ソニーコンピュータエンタテインメント)

## 概要

近年発表されるプロセッサの殆どはメディア処理向けショートベクタ SIMD 拡張命令セットを備えるようになってきたが、コンパイラによる自動適用は非常に限定的で研究段階である。本テーマでは、コンパイラが SIMD 型拡張命令セットを活用するようにプログラムを変換する方式を開発し、並列化コンパイラの共通インフラストラクチャの一部として利用可能な形にした。

## 1 目的

メディア処理アプリケーションでは、PCM 音声や画像のビットマップイメージのように比較的サイズが小さい均質なデータが多数連続しているものを取り扱うことが多い。それらの処理の高速化を比較的小さなコスト増で達成するために、近年発表されるプロセッサでは、IA32 の MMX/SSE/SSE2/SSE3 や PowerPC の AltiVec のように、メディア処理向けショートベクタ SIMD (Single Instruction Multiple Data stream) 拡張命令 (以下 SIMD 命令と略) セットを備え、1 命令で複数のデータを並列処理するようになってきている。しかしながら、これらの拡張命令の既存のコンパイラによる活用は非常に限定的で [1]、多くの場合はアセンブラによる記述や、イントリンシクルーチンといった命令セット依存の方法を用いる必要がある。本テーマでは、SIMD 命令セットを活用するためのプログラム変換の方式と、コード生成方式を開発した。

一方で、その開発途上でコンパイラの SIMD 並列化部分の検証や SIMD 並列化の効果を測定には、既存のテストプログラムやベンチマークプログラム [2] では不十分であることが判った。そこで、そのような目的に合致したベンチマークプログラムを設計し、試験実装した [3]。

## 2 SIMD 並列化

### 2.1 設計

SIMD 並列化系の設計方針は次の通りである。

- それぞれの言語のフロントエンドに対して、SIMD 並列化のための言語仕様の拡張を要求しない。(つまり汎整数拡張のような規約を遵守しなければならない。)

- SIMD 並列化系では「適切なループ展開」や「入れ子になった if 文の引き剥がし」のようなソースコードレベルで実施可能 (つまり HIR レベルで実施すべき) 最適化を行わず、単純な IF 変換と同形オペレーションのまとめ上げに徹する。

SIMD 命令を活用する最適化には、従来のベクトルプロセッサ向けの最適化技術を応用可能な事項もあるが、(1) 並列度は 2 から 16 とあまり大きくないが、ベクタレジスタの出し入れのコストは低い、(2) 整数型データに対しては、飽和演算や差の絶対値、乗算の上位ワードの値等、特殊な演算が用意されている、(3) データサイズの混合を考慮する、といったベクトル命令とは異なる特徴がある。SIMD 並列化ではこれらに留意する。

次の 3 つの最適化方法を LIR から LIR へのターゲット命令セット非依存の変換系として実装した。

- 論理演算を利用した if 変換
- 特殊な SIMD 命令への置き換えを行うパターンマッチ
- 針穴式最適化 (peephole optimization) の拡張による SIMD 命令パターンの生成

その動作例を図 2 に示す。

SIMD 命令パターンからの実マシンコードの生成には、バックエンドの TMD 記述によるコード生成系をそのまま用いた。

結果的に実装した SIMD 並列化系では、図 1 の A や B のようなコードに対しては SIMD 並列化を行えないが、C や D や E のようなコードに対してはそれを行う。ベクトル化を主体として SIMD 並列化をそこから紡ぎ出すコンパイラでは、E のようなコードに対して SIMD 並列化を行えない。

### 2.2 データサイズ推論

図 1 の 2 つの整数値の平均を求める AVE() の定義を用いると、全ての演算は 16 ビットの演算器やレジスタで行うことが可能である。しかし特別な宣言を言語仕様に盛り込まずコンパイラの汎整数拡張に忠実に従うと、言語の整数型 (多くの場合は 32 ビット) まで符号/ゼロ拡張して演算を行わねばならない。すると、処理データサイズが大きくなっ

```

#define AVE(x,y) (((x)>>1)+((y)>>1)+(((x)|(y))&1))
struct {
    short r, g, b, a;
} *u1, *u2, *u3;

short *v1, *v2, *v3;
/* Assume that all pointers are aligned,
   and distances of source and destination pointers
   are longer than the size of vector register. */
for (i = 0; i < M; i++) // A
    *v1++ = AVE(*v2++, *v3++);
for (i = 0; i < M; i++) // B
    v1[i] = AVE(v2[i], v3[i]);
for (i = 0; i < M; i += 4) { // C
    v1[i] = AVE(v2[i], v3[i]);
    v1[i+1] = AVE(v2[i+1], v3[i+1]);
    ...
    v1[i+3] = AVE(v2[i+3], v3[i+3]); }
for (i = 0; i < M; i += 4) { // D
    v1[0] = AVE(v2[0], v3[0]);
    v1[1] = AVE(v2[1], v3[1]);
    ...
    v1[3] = AVE(v2[3], v3[3]);
    v1+=4; v2+=4; v3+=4; }
for (i = 0; i < M; i++) { // E
    u1[i].r = AVE(u2[i].r, u3[i].r);
    u1[i].g = AVE(u2[i].g, u3[i].g);
    u1[i].b = AVE(u2[i].b, u3[i].b);
    u1[i].a = AVE(u2[i].a, u3[i].a); }

```

図 1: SIMD 並列化可能な例と不可能な例

て演算の並列度が低下するだけでなく、拡張やパッキングといったオーバーヘッドもかかる。

我々はこの点に注目し、演算子間の有効ビット (最終的な演算結果に影響をもたらすビット) の集合を推論することで、汎整数拡張を遵守する場合と同じ結果を得られる最適な処理データサイズを決定する方法を開発した [4]。

### 2.3 SIMD コード生成系の実装

SIMD 並列化系に連動した SIMD コード生成系を、IA32/SSE2 や PlayStation2(MIPS)/EmotionEngine, それに PowerPC/AltiVec の各通常命令セット/SIMD 拡張命令セットの組み合わせに対して命令記述を用いて実装した。

IA32/SSE2 については、特殊な SIMD 命令とのマッチングと生成が可能なレベルに達している。それ以外は、SIMD 命令の生成には至らなかったが、通常命令の生成は行える程度になっている。これらについては、SIMD 命令生成の記述を引き続き行っていく。

## 3 SIMD ベンチマーク

SIMD ベンチマークの設計は以下の要求を満たすことを目標としている。

- コンパイラによる SIMD 命令活用の充実度の確認
- コンパイラによる SIMD 命令の誤用の検出

- 同一の処理内容に関して、コンパイラが得手とする記述方法のプログラマへの提示

実際のアプリケーションを調査して、SIMD 並列化可能なコードパターンを抽出した結果、ホールインワンのなマッチングを用いない限りは、現状では殆どがそのままでは適切な SIMD 命令の生成ができないと判った。そこで、各々のコードパターンを人手で変形して、SIMD 命令の生成が容易なレベルから、元のコードに至る複数の版を作り、それぞれの版の実行時間を計測できるようにした。それらはループの展開の有無や、入れ子の if 文の皮むきの有無等で分類されている。

例えば、他に比べて実行時間が小さく結果が正しい版は、コンパイラが適切な SIMD 命令を生成したとみなせ、その版の書き方を参考にしてプログラムを記述するとそのコンパイラから最大の性能を引き出せる。また、読みのベクタと書き出しのベクタに重なりがある場合や、非整序アドレスをベクタの先頭として与える例題に対して誤った結果を出していれば、SIMD 命令が誤用されていると考えられる。

## 4 まとめ

SIMD 並列化のマシン依存部と非依存部の明確な切り分けや、SIMD 並列化失敗時の通常コード生成への切り替え等の実装は、次回のリリースには完成する予定である。

特殊な命令へのマッチングに使用するテンプレートの TMD からの自動生成や、ループ並列化との連携等 HIR レベルでの SIMD 最適化、ベンチマークのパターンの拡充が今後の課題として挙げられる。

## 参考文献

- [1] A. J. C. Bik, M. Girkar, P.M. Grey, and X. Tian. Automatic intra-register vectorization for the intel®architecture. *International Journal of Parallel Programming*, Vol. 30, No. 2, pp. 65–98, 2002.
- [2] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *International Symposium on Microarchitecture*, pp. 330–335, 1997.
- [3] 室田朋樹, 鈴木貢, 渡邊坦. Simd ベンチマークの設計. 先進的計算基盤システムシンポジウム SACSIS2004 論文集, pp. 159–160, May 2004.
- [4] 鈴木貢, 藤波順久, 福岡岳穂, 渡邊坦, 中田育男. マルチメディア SIMD 命令活用のためのデータサイズ推論. 情報処理学会 論文誌:プログラミング, Vol. 45, No. SIG 5(PRO21), pp. 1–11, 2004.

```
static unsigned char sa,sr,sg,sb;
static short da,dr,dg,db;
static short k;
```

```
static void
hLineRight(unsigned char *p,int n,
           unsigned char a,short ea,
           unsigned char r,short er,
           unsigned char g,short eg,
           unsigned char b,short eb) {
while(n!=0) {
  *p++=b; *p++=g; *p++=r; *p++=a;
  a+=sa; r+=sr; g+=sg; b+=sb;
  if((ea+=da)>=0) { a++; ea-=k; }
  if((er+=dr)>=0) { r++; er-=k; }
  if((eg+=dg)>=0) { g++; eg-=k; }
  if((eb+=db)>=0) { b++; eb-=k; }
  --n;
}
}
```

SIMD instructions

```
paddb    %mm1, %mm0
paddw    %mm3, %mm2
pxor     %mm4, %mm4
pcmpgtw  %mm2, %mm4
```

normal instructions

```
(SET (REG I8 t0) (ADD I8 (REG I8 t0) (REG I8 t4)))
(SET (REG I8 t1) (ADD I8 (REG I8 t1) (REG I8 t5)))
(SET (REG I8 t2) (ADD I8 (REG I8 t2) (REG I8 t6)))
(SET (REG I8 t3) (ADD I8 (REG I8 t3) (REG I8 t7)))
(SET (REG I16 t8) (ADD I16 (REG I16 t8) (REG I16 t12)))
(JUMP2 (TSTGE I1 (REG I16 t8) (INTCONST I16 0)) (LABEL L0) (LABEL L4))
(LABEL L0) (SET (REG I8 t0) (ADD I8 (REG I8 t0) (INTCONST I8 1)))
(SET (REG I16 t8) (ADD I16 (REG I16 t8) (REG I16 t28)))
(LABEL L4)
(SET (REG I16 t9) (ADD I16 (REG I16 t9) (REG I16 t13)))
(JUMP2 (TSTGE I1 (REG I16 t9) (INTCONST I16 0)) (LABEL L1) (LABEL L5))
(LABEL L1) (SET (REG I8 t1) (ADD I8 (REG I8 t1) (INTCONST I8 1)))
(SET (REG I16 t9) (ADD I16 (REG I16 t9) (REG I16 t28)))
(LABEL L5)
(SET (REG I16 t10) (ADD I16 (REG I16 t10) (REG I16 t14)))
(JUMP2 (TSTGE I1 (REG I16 t10) (INTCONST I16 0)) (LABEL L2) (LABEL L6))
(LABEL L2) (SET (REG I8 t2) (ADD I8 (REG I8 t2) (INTCONST I8 1)))
(SET (REG I16 t10) (ADD I16 (REG I16 t10) (REG I16 t28)))
(LABEL L6)
(SET (REG I16 t11) (ADD I16 (REG I16 t11) (REG I16 t15)))
(JUMP2 (TSTGE I1 (REG I16 t11) (INTCONST I16 0)) (LABEL L3) (LABEL L7))
(LABEL L3) (SET (REG I8 t3) (ADD I8 (REG I8 t3) (INTCONST I8 1)))
(SET (REG I16 t11) (ADD I16 (REG I16 t11) (REG I16 t28)))
(LABEL L7)
```

if-conversion

```
(SET (REG I8 t0) (ADD I8 (REG I8 t0) (REG I8 t4)))
(SET (REG I8 t1) (ADD I8 (REG I8 t1) (REG I8 t5)))
(SET (REG I8 t2) (ADD I8 (REG I8 t2) (REG I8 t6)))
(SET (REG I8 t3) (ADD I8 (REG I8 t3) (REG I8 t7)))
(SET (REG I16 t8) (ADD I16 (REG I16 t8) (REG I16 t12)))
(SET (REG I16 t16) (TSTGE I16 (REG I16 t8) (INTCONST I16 0)))
(SET (REG I8 t0) (ADD I8 (REG I8 t0) (NEG I8 (CONVIT I8 (REG I16 t16)))))
(SET (REG I16 t8) (ADD I16 (REG I16 t8) (BAND I16 (REG I16 t28) (REG I16 t16))))
(SET (REG I16 t9) (ADD I16 (REG I16 t9) (REG I16 t13)))
(SET (REG I16 t17) (TSTGE I16 (REG I16 t9) (INTCONST I16 0)))
(SET (REG I8 t1) (ADD I8 (REG I8 t1) (NEG I8 (CONVIT I8 (REG I16 t17)))))
(SET (REG I16 t9) (ADD I16 (REG I16 t9) (BAND I16 (REG I16 t28) (REG I16 t17))))
(SET (REG I16 t10) (ADD I16 (REG I16 t10) (REG I16 t14)))
(SET (REG I16 t18) (TSTGE I16 (REG I16 t10) (INTCONST I16 0)))
(SET (REG I8 t2) (ADD I8 (REG I8 t2) (NEG I8 (CONVIT I8 (REG I16 t18)))))
(SET (REG I16 t10) (ADD I16 (REG I16 t10) (BAND I16 (REG I16 t28) (REG I16 t18))))
(SET (REG I16 t11) (ADD I16 (REG I16 t11) (REG I16 t15)))
(SET (REG I16 t19) (TSTGE I16 (REG I16 t11) (INTCONST I16 0)))
(SET (REG I8 t3) (ADD I8 (REG I8 t3) (NEG I8 (CONVIT I8 (REG I16 t19)))))
(SET (REG I16 t11) (ADD I16 (REG I16 t11) (BAND I16 (REG I16 t28) (REG I16 t19))))
```

match & move

```
(SET (REG I8 t0) (ADD I8 (REG I8 t0) (REG I8 t4)))
(SET (REG I8 t1) (ADD I8 (REG I8 t1) (REG I8 t5)))
(SET (REG I8 t2) (ADD I8 (REG I8 t2) (REG I8 t6)))
(SET (REG I8 t3) (ADD I8 (REG I8 t3) (REG I8 t7)))
(SET (REG I16 t8) (ADD I16 (REG I16 t8) (REG I16 t12)))
(SET (REG I16 t9) (ADD I16 (REG I16 t9) (REG I16 t13)))
(SET (REG I16 t10) (ADD I16 (REG I16 t10) (REG I16 t14)))
(SET (REG I16 t11) (ADD I16 (REG I16 t11) (REG I16 t15)))
(SET (REG I16 t16) (TSTGE I16 (REG I16 t8) (INTCONST I16 0)))
(SET (REG I16 t17) (TSTGE I16 (REG I16 t9) (INTCONST I16 0)))
(SET (REG I16 t18) (TSTGE I16 (REG I16 t10) (INTCONST I16 0)))
(SET (REG I16 t19) (TSTGE I16 (REG I16 t11) (INTCONST I16 0)))
.....
```

parallelize

```
(PARALLEL
 (SET (SUBREG I8 (REG I32 mm0) 0) (ADD I8 (SUBREG I8 (REG I32 mm0) 0) (SUBREG I8 (REG I32 mm1) 0)))
 (SET (SUBREG I8 (REG I32 mm0) 1) (ADD I8 (SUBREG I8 (REG I32 mm0) 1) (SUBREG I8 (REG I32 mm1) 1)))
 (SET (SUBREG I8 (REG I32 mm0) 2) (ADD I8 (SUBREG I8 (REG I32 mm0) 2) (SUBREG I8 (REG I32 mm1) 2)))
 (SET (SUBREG I8 (REG I32 mm0) 3) (ADD I8 (SUBREG I8 (REG I32 mm0) 3) (SUBREG I8 (REG I32 mm1) 3)))
 )
(PARALLEL
 (SET (SUBREG I16 (REG I64 m2) 0) (ADD I16 (SUBREG I16 (REG I64 m2) 0) (SUBREG I16 (REG I64 m3) 0)))
 (SET (SUBREG I16 (REG I64 m2) 1) (ADD I16 (SUBREG I16 (REG I64 m2) 1) (SUBREG I16 (REG I64 m3) 1)))
 (SET (SUBREG I16 (REG I64 m2) 2) (ADD I16 (SUBREG I16 (REG I64 m2) 2) (SUBREG I16 (REG I64 m3) 2)))
 (SET (SUBREG I16 (REG I64 m2) 3) (ADD I16 (SUBREG I16 (REG I64 m2) 3) (SUBREG I16 (REG I64 m3) 3)))
 )
(PARALLEL
 (SET (SUBREG I16 (REG I64 m4) 0) (TSTGE I16 (SUBREG I16 (REG I64 m2) 0) (INTCONST I16 0)))
 (SET (SUBREG I16 (REG I64 m4) 1) (TSTGE I16 (SUBREG I16 (REG I64 m2) 1) (INTCONST I16 0)))
 (SET (SUBREG I16 (REG I64 m4) 2) (TSTGE I16 (SUBREG I16 (REG I64 m2) 2) (INTCONST I16 0)))
 (SET (SUBREG I16 (REG I64 m4) 3) (TSTGE I16 (SUBREG I16 (REG I64 m2) 3) (INTCONST I16 0)))
 )
.....
```

図 2: SIMD 並列化の例