

画像フィルタ処理の高速化に向けた メディア拡張プロセッサ用 SIMD コンパイラ

京 昭倫 岡崎信一郎 黒田一朗

NEC マルチメディア研究所

〒 216-8555 川崎市宮前区宮崎 4-1-1

s-kyo@cq.jp.nec.com

あらまし 近年、SIMD 命令を備えるプロセッサ、いわゆるメディア拡張プロセッサが増加しているが、プログラムの自動並列化が困難のため、現状ではそれらを有効に利用できる高級言語コンパイラが存在しない。本稿ではまず、画像に対する行単位の並列処理を記述できるデータ並列 C 言語 1DC を用い、画像処理アプリケーションの中でも特に演算量の多い画像フィルタ処理の SIMD 命令による並列化には、行単位の並列演算指定が有効であることを示す。次に、1DC 記述から SIMD 命令コードシーケンスを生成できる SIMD コンパイラ 1DCC について述べる。メディア拡張プロセッサの代表例である Pentium シリーズプロセッサを用いたベンチマークテストでは、1DC 記述に対する 1DCC 生成コードが、通常の C 記述に対する最新 C コンパイラの生成コードと比べ、ワード演算主体の場合で 2~3 倍、バイト演算主体の場合では 7 倍程度高速であるという結果が得られた。

キーワード 画像フィルタ、SIMD 命令、並列処理、画像処理、コンパイラ、MMX、拡張 C 言語、データ並列言語

A SIMD Compiler for Efficient Implementation of Image filters On Media Extended Processors

Shorin KYO Shin'ichiro OKAZAKI Ichiro KURODA

Multimedia Research Laboratories, NEC Corporation

4-1-1, Miyazaki, Miyamae-ku, Kawasaki, Kanagawa 216-8555

s-kyo@cq.jp.nec.com

Abstract Recently, general purpose processors with enhanced SIMD instructions, or so called media extended processors, become more and more popular. However, as automatic extraction of parallelism from conventional sequential C programs is still difficult, currently there exists no effective compiler which can generate efficient code sequences based on the use of these SIMD instructions. This report first describe a data parallel extended C language called 1DC (One Dimensional C) used for succinct description of parallel image filter algorithms based on sweeping a horizontal pixel-updating-line across images, and then describe a SIMD compiler called 1DCC (One Dimensional C Compiler) which can generate effective SIMD instruction based code sequences from 1DC descriptions of image filter applications. Benchmark results of image filter tasks using the Intel Pentium processors show that, codes generated by 1DCC from 1DC descriptions can achieve an two to seven times better performance than codes generated by a latest C compiler from conventional C descriptions.

Key words image filter, SIMD instruction, parallel processing, compiler, MMX, extended C language, data parallel programming language

1 はじめに

近年、メディア処理向けに MMX や SSE[1] に代表されるような SIMD 命令を備えるプロセッサ、いわゆるメディア拡張プロセッサが増加している。しかし、高級言語コンパイラの自動並列化技術がまだ未熟であるため、現状では高級言語で書かれたプログラムのままでは、多くの場合 SIMD 命令の恩恵を被ることができない状態にある。

自動並列化コンパイラの出現を待たずに、高級言語から SIMD 命令を有効に利用する方法の一つに、データ並列言語を設計し、その SIMD コンパイラを開発するアプローチが考えられる。但し SIMD 命令の並列処理対象はメモリ上にある 4~16 個の連続アドレスに存在するデータに限られているため、汎用的あるいは高並列 SIMD マシン向けに設計された既存のデータ並列言語 [2][3][4] を用いたのでは、記述の多くが効率よく SIMD 命令にマッピングできない可能性が高い。こうした中で Purdue 大学において SWARC (SIMD Within A Register C)[5] という、SIMD 命令の利用を考慮したデータ並列型 C 言語、およびその SIMD コンパイラ SCC (Swar C Compiler) が実験的に開発された。残念ながら、SWARC では SIMD 命令を高級言語記述から抽象的に利用できるようにしただけに留まり、アプリケーションレベルでの SWARC の記述性や SCC によるコード生成の効率性がこれまで示されていない。

一方近年、顔検出やジェスチャ・表情認識等の動画像を用いたマルチメディア関連応用の実現に向けた研究が盛んである。こうした中で画像演算、特に処理量の多い画像フィルタ演算を汎用プロセッサ上で SIMD 命令を利用し高速に実行したいという要求が強い。しかし、画素毎の単純な算術論理演算ならば、SIMD 命令へのマッピングは一意的に定まるが、画像フィルタ演算では、画像内の各画素位置毎に任意サイズの局所近傍領域内の画素への参照を含む。そうした処理は、並列処理の対象となる 4~16 個のデータがメモリ上で連続に存在しかつ先頭位置が 8 バイト等でアラインメントされていることを要求する SIMD 命令への直接的なマッピングは困難であり、またこれまでその効率的な実現に向けた方法論も提案されていない。

筆者らはこれまで、一次元 SIMD プロセッサアレイ (LPA: Linear Processor Array) 向けに、画像演算をライン単位で並列化する手法 (ライン方式)、およびそれを簡潔に記述できる拡張 C 言語 1DC (One Dimensional C) を提案してきている [6]。SIMD 命

令が有する並列処理機能は LPA のサブセットに相当することから、ライン方式に基づく並列化手法のうち、SIMD 命令の有効利用に適しているものが存在する可能性が高い。このことを実証するために、筆者らは 1DC 記述から汎用プロセッサの SIMD 命令シーケンスを生成する SIMD コンパイラ 1DCC を開発した。その結果、ライン方式の一種である水平型ライン方式に基づく画像フィルタ演算の 1DC 記述に対する 1DCC 生成コードが、従来の C 記述に対する C コンパイラ生成コードと比べ、2~7 倍の性能向上が得られることがわかった。以下 2 章ではライン方式および 1DC、3 章では 1DCC の構成、そして 4 章では各種画像フィルタを用いた 1DCC 生成コードと C コンパイラコードとのベンチマークテストの結果について述べる。これらにより本稿では、画像フィルタ演算の SIMD 命令へのマッピングには、水平型ライン方式に基づくアルゴリズムが有効であり、1DC と 1DCC の組合せがそのための有効なツールであることを示す。

2 ライン方式とその記述言語 1DC

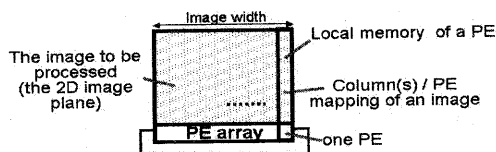


図 1: 仮想的な LPA の構成

図 1 に、処理対象画像の横幅画素数に等しい数の PE を持ち、かつ画像の各列が各 PE のローカルメモリに割り当てられている仮想的な LPA を示す。ライン方式とは、この LPA の全 PE のローカルメモリの集合で表される 2 次元画像平面上で、画素更新ラインと呼ぶライン上に位置する画素を対象に全 PE で並列に処理を行い、かつ画素更新ラインの位置を次々と移動させ、最終的には当該 2 次元画像平面を一掃することで、処理タスクを並列化する手法を総称したものである。図 2 に、画素更新ラインの形状によって水平型・折返し型・傾斜型・自律型と呼ぶ 4 種類のライン方式、および各種画像処理のカテゴリ [7] のうち各ライン方式によって並列化できるものを示す。

1DC[6] は、これらのライン方式に基づく処理を効率よく記述することを目的に設計されたデータ並列型拡張 C 言語である。1DC の C からの拡張は、各スカラー要素が各 PE 上に存在し LPA における PE 数と同数のスカラー要素を表す sep データを宣

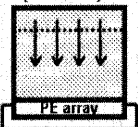
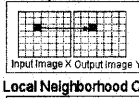
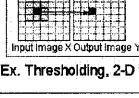
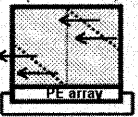
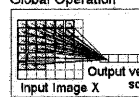
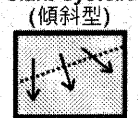
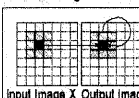
	ライン方式の種類	対応する画像処理 のカテゴリ
1)	Row-wise (水平型) 	Pixel Operation  Local Neighborhood Operation  Ex. Thresholding, 2-D filters
2)	Row-systolic (折返し型) 	Global Operation  Ex. Pixel histogram value, Statistic data of objects, Affine transformation
3)	Slant-systolic (傾斜型) 	Recursive Neighborhood Operation  Ex. Distance transform, Object skeleton detection
4)	Autonomous (自律型) 	Object Operations  Ex. Labeling, Boundary tracing, Region growing

図 2: 4 種類の有効なライン方式

言するための sep 宣言子と、sep データによる演算操作を指定するための 6 種類の拡張記述 (図 3) からなる。以下 1.~6. にこれらの拡張記述を定義する。 E_{sep} 、 A_{sep} 、 E_{sca} をそれぞれ sep データの式、sep データの配列、そしてスカラーデータの式とする。 E'_{sep} や E'_{sca} の表記は、相異なる 2 つの E_{sep} あるいは E_{sca} を区別する場合に用いる。

- sep データ演算記述 (arithmetic)
" $E_{sep} \text{ op } E'_{sep}$ " または " $\text{op } E_{sep}$ " で PE アレイによる「SIMD 演算」を指定する。 op のところは、+、-、×、÷ 等、C と同様の 2 項または単項演算子が入る。
- 近隣要素参照記述 (left/right reference)
" $>E_{sep}$ "、" $<E_{sep}$ " でそれぞれ左右 PE 上の E_{sep} のスカラー値を参照する「近隣要素参照」動作を指定する。
- PE グループ記述 (PE grouping)
" $\text{mif} \dots [\text{melse} \dots]$ "、" $\text{mdoldots} [\text{mwhile} \dots]$ "、そして " $\text{mfor} (\dots) \dots$ " などの sep 制御文を用い、sep データの値を返す条件式を評価し、その結果が偽 (0) のグループと真 (1) のグループに PE アレイを 2 分し、真のグループの PE のみが、後続文や式を実行するという「PE グループ記述」動作を指定する。たとえば " $\text{mif}(E'_{sep})$ "、" $E_{sep} = E_{sca}$ " の場合、 E'_{sep} の結果要素が 1 の PE のみにおいて、" $E_{sep} = E_{sca}$ " が実行される。
- 間接アドレッシング記述 (index addressing)
" $A_{sep}[E_{sep}]$ " で sep 配列 A_{sep} に対し sep データ E_{sep} でインデックス参照を行なう動作を指定する。 E_{sep} に

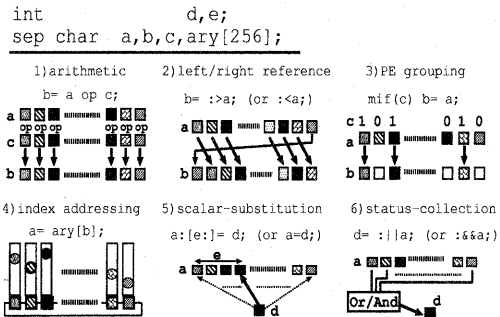


図 3: IDC の 6 つの拡張記述

対応する各 PE 上のスカラー要素が相異れば、各 PE は相異なるメモリアドレスにアクセスする「間接アドレッシング」動作を行うよう指定できる。

- スカラーデータの抽出・放送記述 (scalar-substitution)
スカラーデータを sep データに代入する「放送」動作、または逆に sep データから特定のスカラー要素を抽出する「データ抽出」動作を指定する。 $E_{sep}:[E'_{sca}]$ で E'_{sca} 番目の PE 上の E_{sep} のスカラー要素を指定する。
- ステータス集計記述 (status collection)
" $:\&\& E_{sep}$ " で E_{sep} の全ての要素が 1 であれば 1、そうでなければ 0 というスカラー値を戻し、同様に " $|| E_{sep}$ " で E_{sep} の全ての要素が 0 であれば 0、そうでなければ 1 というスカラー値を戻す、という「ステータス集計」動作を指定する。

図 2 からわかる通り、水平型ライン方式を除く 3 種類のライン方式の実現には、各 PE が相異なるメモリアドレスにアクセスできる「間接アドレッシング (index addressing)」の動作が必要であるが、残念ながら既存の SIMD 命令では、間接アドレッシングの動作を効率よく実現できない。そこで以降では、水平型ライン方式によって並列化できる画像フィルタ演算に係わる IDC 記述、およびその SIMD 命令へのマッピング方法のみに着目する。

画像フィルタに対する通常の C 記述と比べ、水平型ライン方式に基づく IDC 記述は、1) 配列アクセスに伴うアドレス計算指定が少ない、2) フィルタ係数の対称性を利用した演算量削減を簡潔かつ容易に表現できる、の 2 つの特徴を持つ。以下、典型的な画像フィルタ演算であるコンボリューションを例にそれらを示す。図 4 の (a)~(c) が、それぞれ下記 (X)~(Z) の 3 種類の 5×5 のフィルタ係数を用いた場合の、コンボリューション演算の IDC 記述である。なお対象画像は NROW 行からなるとする。

(X) 非対称	(Y) 左右対称	(Z) 点対称
$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 6 & 8 & 10 \\ 3 & 6 & 9 & 12 & 15 \\ 4 & 8 & 12 & 16 & 20 \\ 5 & 10 & 15 & 20 & 25 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$

まずフィルタ係数の対称性を利用しない図 4(a) の IDC 記述では、最初に一行内の各画素位置につ

```

#define uchar unsigned char
sep uchar src[NROW];
sep int dst[NROW];

void non_sym_filter(sep uchar src, sep int dst) {
    sep int UP2,UP1,CTR,DN1,DN2;
    sep int L1,L2,X,R1,R2;
    int i;
    for(i=2; i<NROW-2; i++) {
        UP2= src[i-2];
        UP1= src[i-1];
        CTR= src[i];
        DN1= src[i+1];
        DN2= src[i+2];
        /*1*/ L2 = a*UP2 + f*UP1 + l*CTR + q*DN1 + v*DN2;
        /*2*/ L1 = b*UP2 + g*UP1 + m*CTR + r*DN1 + w*DN2;
        /*3*/ X = c*UP2 + h*UP1 + n*CTR + s*DN1 + x*DN2;
        /*4*/ R1 = d*UP2 + j*UP1 + o*CTR + t*DN1 + y*DN2;
        /*5*/ R2 = e*UP2 + k*UP1 + p*CTR + u*DN1 + z*DN2;
        /*6*/ dst[i]=(L2>2) + :>L1 + X + :<R1 + (R2:<2);
    }
} (a) 非対称型フィルタ係数時 (1DC 記述)

void sym_filter(sep uchar src, sep int dst) {
    sep int UP2,UP1,CTR,DN1,DN2;
    sep int L1,L2,X;
    int i;
    for(i=2; i<NROW-2; i++) {
        UP2= src[i-2];
        UP1= src[i-1];
        CTR= src[i];
        DN1= src[i+1];
        DN2= src[i+2];
        /*1*/ L2 = a*UP2 + f*UP1 + l*CTR + q*DN1 + v*DN2;
        /*2*/ L1 = b*UP2 + g*UP1 + m*CTR + r*DN1 + w*DN2;
        /*3*/ X = c*UP2 + h*UP1 + n*CTR + s*DN1 + x*DN2;
        /*6*/ dst[i]=(L2>2) + :>L1 + X + :<L1 + (L2:<2);
    }
} (b) 左右対称型フィルタ係数時 (1DC 記述)

void point_sym_filter(sep uchar src, sep int dst) {
    sep int CTR,T1,T2;
    sep int L1,L2,X;
    int i;
    for(i=2; i<NROW-2; i++) {
        T2 = src[i-2]+src[i+2];
        T1 = src[i-1]+src[i+1];
        CTR = src[i];
        /*1*/ L2 = a*T2 + f*T1 + l*CTR;
        /*2*/ L1 = b*T2 + g*T1 + m*CTR;
        /*3*/ X = c*T2 + h*T1 + n*CTR;
        /*6*/ dst[i]=(L2>2) + :>L1 + X + :<L1 + (L2:<2);
    }
} (c) 点対称型フィルタ係数時 (1DC 記述)

```

図 4: 5×5 コンボリューション演算記述

いて、上下方向の1×5の画素 ($src[i-2] \dots src[i+2]$) と、(X)における縦方向の5通りの1×5のフィルタ係数との積和演算を行わせ、結果をそれぞれ sep 変数 L2、L1、X、R1、そして R2 に格納する (図 4(a) の /*1*/.../*5*/)。次に、2つ左隣 PE の L2、左隣 PE の L1、右隣 PE の R1、そして2つ右隣 PE の R2 をそれぞれ、近隣要素参照記述により参照し自 PE の X と合計させることで、1行分の演算結果 (図 4(a) の /*6*/) を得る。この図 4(a) の 1DC 記述でのアドレス計算回数は各画像行毎に5回である。一般的な5×5 コンボリューション演算のC記述では各画素位置毎に計5×5=25回のアドレス計算が必要なのと比べ、大幅にアドレス計算回数が低減されていることがわかる。

次に、図 4(b) の 1DC 記述では、単純に図 (a)

での R1 と R2 の計算を省略し代わりに右隣 PE の L1 と、2つ右隣 PE の L2 を参照することで、フィルタ係数 (Y) の左右対称性を利用した演算量削減を実現している。図 (c) の 1DC 記述でも、同様に簡単な記述修正だけでフィルタ係数 (Z) の点対称性を利用した演算量削減を実現している。図 4では (a),(b),(c) の順に、一つの結果画素の計算に要する加算と乗算の合計回数がそれぞれ 49,31,21 と理想的に減少してきているのに対し、対応する各 1DC 記述は簡潔な表現となっている。このことから、水平型ライン方式に基づく 1DC 記述は、フィルタ係数の対称性の利用による演算量削減に適したコンパクトな表現を実現しているといえる。

3 SIMD コンパイラ 1DCC

1DCC は翻訳部、マクロ最適化部、そしてコード生成部で構成されている。翻訳部では構文解析や基本ブロックレベルの各種最適化処理を実施後、1DC 記述をマクロコード列と呼ぶ内部表現に変換する。個々のマクロコードは即値、メモリアドレス、仮想レジスタ (scalar レジスタや sep レジスタ) 等をオペランドを持つ。sep レジスタは sep データの格納領域であり、以降 sep レジスタをオペランドに持つマクロコードを SIMD マクロと呼ぶとする。

SIMD 命令が持つ並列度を P、sep データ内における要素データ数 (=処理対象画像の横断画素数) を S すると、P は高々4~8であるため一般に $S > P$ である。通常のメディア拡張プロセッサには、最大 P 個のデータまでを同時に格納できるレジスタ・リソースしか存在しない。そのためコード生成部は、全ての sep レジスタをメモリにマッピングし、sep レジスタ間演算を表す個々の SIMD マクロに対しては、P 個のデータを並列に処理できる SIMD 命令を利用した「メモリ (sep レジスタ) リード⇒演算処理⇒メモリ (sep レジスタ) ライト」からなる SIMD 命令シーケンスを $S \div (P \times M)$ 回繰り返すコードを生成する。M はループ本体のコード量を増やしループオーバーヘッドを減らすためのループ展開数を表し、その値はターゲットマシン毎の利用可能な SIMD レジスタ数や、SIMD マクロの種類に応じ適宜に定められる。なお、SIMD マクロ以外のマクロに対しては、一般的なスカラー命令シーケンスを生成する。

このように個々の SIMD マクロに対する生成コードは、先頭と末尾に必ずメモリアクセス処理を伴いことから、SIMD の個数分だけこうしたメモリアクセス・オーバーヘッドが発生する。そこでマクロ

最適化部では SIMD マクロ同士のデータ依存関係を解析し、その結果を元に SIMD マクロ数を圧縮する作業を行う。これにより例えば、データ依存関係にある下記の SIMD マクロ A~C は、sep レジスタ X と Y が他の SIMD マクロから参照されなければ、X と Y の生成を省略した複合 SIMD マクロ D の表現に変換される。

A) sep レジスタ W リード⇒演算 A⇒ sep レジスタ X ライト
B) sep レジスタ X リード⇒演算 B⇒ sep レジスタ Y ライト
C) sep レジスタ Y リード⇒演算 C⇒ sep レジスタ Z ライト

(マクロ最適化部)

D) sep レジスタ W リード⇒演算 A~C⇒ sep レジスタ Z ライト

実際に開発した Pentium シリーズプロセッサ用のコード生成部による、S=256 とした場合のコード生成例を図 5 に示す。2 バイト演算を行う SIMD マクロの例 (図 5(a)) に対し M は 8 (=利用可能な SIMD レジスタの最大数)、Pentium シリーズプロセッサでは 2 バイト演算の場合 P は 4、したがってループ回数が $S \div (P \times M) = 256 \div (4 \times 8) = 8$ の SIMD 命令シーケンスを生成している。

(a) 2 バイト加算 (M=8) (b) 1-to-2 バイトキャスト加算 (M=4)

```

mov esi, r0      ループ回数設定
mov ebx, r1
mov edi, r2
mov ecx, 8
align 16
LABEL_MMX_ADD:
// sep_reg リード
movq mm0, [esi]
movq mm1, [esi+8]
movq mm2, [esi+16]
movq mm3, [esi+24]
M movq mm4, [esi+32]
movq mm5, [esi+40]
movq mm6, [esi+48]
movq mm7, [esi+56]
// 加算
paddw mm0, [ebx]
paddw mm1, [ebx+8]
paddw mm2, [ebx+16]
paddw mm3, [ebx+24]
M paddw mm4, [ebx+32]
paddw mm5, [ebx+40]
paddw mm6, [ebx+48]
paddw mm7, [ebx+56]
// sep_reg ライト
movq [edi], mm0
movq [edi+8], mm1
movq [edi+16], mm2
M movq [edi+24], mm3
movq [edi+32], mm4
movq [edi+40], mm5
movq [edi+48], mm6
movq [edi+56], mm7
// ループ処理
add esi, 64
add ebx, 64
add edi, 64
dec ecx
jnz LABEL_MMX_ADD

```

1命令で4ワードを処理 (P=4)

図 5: Pentium プロセッサ向けコード生成例

一方、キャスト処理と加算処理の合体による複合 SIMD マクロの例 (図 5(b)) では、P は 4、キャスト処理がワーク用に 1 つの SIMD レジスタ (mm7) を消費するため、コード生成の都合上 4 の倍数の条件を持つ M は 4、したがってループ回数が $256 \div (4 \times 4) = 16$ の SIMD 命令シーケンスを生成している。

1DCC のパス構成を図 6 に示す。コード生成部はマイクロソフト社製アセンブラ (MASM) 互換のアセンブリコードを出力するため、他の MASM を利用する C コンパイラとはオブジェクトファイルレベルでの互換性を持つ。また、デバッグ時に sep データを配列として認識するよう、オブジェクトファイルに対し追加のデバッグ情報を挿入するパス (1dc53P) を用意したことで、C と 1DC のオブジェクトコードが混在した実行形式でも、既存の GUI デバッガを利用した両者のデバッグが可能である。

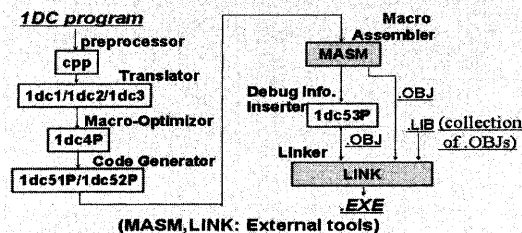


図 6: 1DCC のパス構成

4 性能評価

733MHz の Pentium III (P3) および 2.4GHz の Pentium4(P4) の 2 種類の PC 上で、256x240 の濃淡画像に対する各種画像フィルタについて、C 記述に対する C コンパイラ (Intel C コンパイラ Ver.7) の生成コードと、1DC 記述に対する 1DCC の生成コードの処理時間を比較した結果について述べる。なおコード生成は両者共に速度最優先のオプションを使用し、1DCC の場合はコンパイル時オプションで、P3 向けには MMX と SSE、P4 向けには SSE2 命令も利用したコードを生成させた。まずはカーネルサイズ 3x3~13x13 のコンボリューション演算に対し、下記 a)~e) の各種記述を用いた場合の結果を図 7 に示す。

- フィルタ係数を変数とした 4 重ループ (画像行数×列数×マスク行数×マスク列数) の C 記述
- 定数のフィルタ係数とし、a) での内側 2 重ループ (マスクサイズ分) を展開した C 記述
- b) に加えフィルタ係数の対称性を利用した C 記述
- 定数フィルタ係数で図 4(a) 同様の 1 重ループ 1DC 記述
- d) に加え図 4(c) 同様のフィルタ係数の対称性を利用した 1DC 記述

C 記述同士の比較では、P3 では (a) と (b) の処理時間差が小さいのに対し、P4 では大きな差が見られた。これは 20 段のパイプラインを持つ P4 にとり、ループ展開が効果的であったことを示している。一方 P4 では c) よりも b) の方が高速であった。c) では b) と比べ演算量が削減されている代わりに

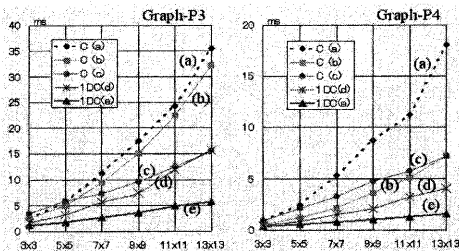


図 7: 性能比較結果:コンボリューション演算

中間結果を格納したメモリ領域へのアクセスが増加している。このことから P4 の場合、演算量削減よりもメモリアクセス数を増やさない方が高速なコードになると考えられる。C と 1DC との比較では、カーネルサイズの増加に応じ、フィルタ係数の対称性を利用しない b) と d) との比較では最大 2 倍まで、対称性を利用した c) と e) との比較では最大 4 倍まで、1DC 記述の方が高速であるという結果が得られた。コンボリューション演算は 2 バイト計算が主体であるため、SIMD 命令利用時の理論的な高速化の上限は 4(=P) 倍である。このことは、カーネルサイズが大きいコンボリューション演算では、1DC 記述に対する 1DCC の生成コードが、最大限の高速化を実現していることを示す。

次に、単純な画素間演算処理 (1. と 2.)、およびコンボリューション以外の各種フィルタ演算 (3. ~ 6.) を用いた場合の結果を図 8 に示す。

1. 単純単項画素演算 (not): 画素毎の反転処理
2. 単純画素間二項演算 (and): 画素間論理積
3. 5x5 のテキストチャ検出フィルタ (var5oct): 画素位置毎の中心と 5x5 近傍内全画素との差の自乗和
4. canny エッジ検出フィルタ (canny): 画素位置毎の 2 次微分と勾配量を利用したエッジフィルタ [8]
5. 5x5 エッジ保存型スムージング処理 (smoothing): 各画素位置を端とする 4 つの 3x3 の方形領域のうち、領域内画素差が最小の方形領域内の平均画素値を求める
6. 3x3 濃淡モーフォロジー演算 (greyopen3): 画素位置毎の 3x3 近傍内の最大画素値を結果画素とする中間画像 B を求めた後、画像 B の各画素位置を中心とする 3x3 近傍内の最小画素値を求める

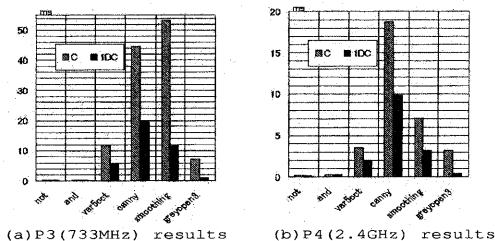


図 8: 性能比較結果:各種画像フィルタ

1. と 2. のような単純な画素間演算処理の場合、今回利用した C コンパイラでは自動並列化がうま

く機能し SIMD 命令利用コードを生成できたため、1DCC 生成コードとは性能差が見られなかった。一方、3. ~ 6. のフィルタ演算では、2 バイト演算が中心の *var5oct* ~ *smoothing* では 1DC 記述の方が 2 ~ 4 倍、バイト演算が中心の *greyopen3* では 1DC 記述の方が 7 倍近く高速であるという結果が得られた。

5 終わりに

メディア拡張プロセッサが有する SIMD 命令を有効に利用し、画像フィルタ処理の高速化を実現するために、本稿では水平型ライン方式と呼ぶ並列化方式、およびそれを効率よく記述できるデータ並列型拡張 C 言語 1DC、そして 1DC 記述を効率的な SIMD 命令シーケンスに変換可能な SIMD コンパイラ 1DCC について述べた。従来の C 記述と比べ、水平型ライン方式に基づく画像フィルタに対する 1DC 記述は、配列アクセスに伴うアドレス計算指定が少ない点、および画像フィルタが持つ処理の対称性を利用した演算量削減を簡潔に表現できる点に特徴を持つ。Pentium シリーズプロセッサをターゲットマシンとし、各種画像フィルタ処理を用いたベンチマークテストでは、1DC 記述に対する 1DCC 生成コードが、通常の C 記述に対する最新 C コンパイラの生成コードと比べ、ワード演算主体の場合で 2 ~ 3 倍、バイト演算主体の場合では最大 7 倍程度も高速であるという結果が得られた。

画像フィルタ処理のみならず、より多くの種類の画像処理を SIMD 命令で高速化できるようにするためには 2 章で述べたように、間接アドレッシング機能を持つ SIMD 命令が必須である。今後、そうした SIMD 命令の出現を期待したい。

参考文献

- [1] Intel Corp.: "http://www.intel.com/"
- [2] Thinking Machines Corp.: "C* Programming Guide", Ver. 6, 1990.
- [3] Van Waveren G M: "High Performance Fortran", Lect Notes Comput Sci, Vol.797, pp.429-433, 1994.
- [4] 湯浅他: "データ並列計算のための拡張 C 言語 NCX", 信学論, Vol.J78-D-I, No.2, pp.200-209, 1995.
- [5] R.J.Fisher et al.: "The Scc Compiler: SWARing at MMX and 3DNow!", Lect Notes Comput Sci, Vol.1863, pp.399-414, 2000.
- [6] 許他: "一次元プロセッサアレイに基づく超高速画像処理システムの開発環境", 情報処理学会論文誌, Vol.39, No.6, pp.1790-1800, 1998.
- [7] P.P.Jonker: "Architectures for Multidimensional Low- and Intermediate Level Image Processing," Proc. of IAPR Workshop on Machine Vision Applications (MVA'90), pp.307-316, 1990.
- [8] J.Canny: "A Computational Approach to Edge Detection", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-9, No.6, pp.679-698, 1986.