D-02

ベクトル演算機能の活用による画像のフィルタリングの高速化

Fast image filtering with vectorization

山野 光範†　　陳 謙†

Yamano mitsunori　Qian Chen

1. はじめに

昨今の技術発展に伴い，デジタル画像，映像の分野では高解像度化が進み，実時間処理の需要が高まっている．カメラを例に挙げると，コンパクトデジタルカメラで１６００万画素以上が主流になり，デジタル一眼カメラの中には３０００万画素を超えるものも存在する．1995 年にカシオが発売したデジタルカメラ「ＱＶ－10」が 25 万画素だったことを考えると，高解像度化が大変に急速に進んでいることがわかり，画像処理の高速化が重要な課題となっている．そこで本研究では，並列処理を用いることで画像処理の高速化を試みる．並列処理とは，一つの大きい仕事を，独立して処理できる複数の小さな仕事に分割し，複数の演算素子を使って同時に処理することにより，処理時間を短縮する手法である．本研究では画像処理のフィルタリングのプログラムを対象に，ベクトル演算機能による並列処理を実装し，その有用性を評価することを目的としている．

2. 並列処理

並列処理とは，一つの大きい仕事を，独立して処理できる複数の小さな仕事に分割し，複数の演算素子を使って同時に処理することにより処理時間を短縮する手法であり，並列計算や並列コンピューティングとも呼ばれる．また，このために設計されたコンピュータを並列コンピュータと呼ぶ．並列処理を実装には，マルチＣＰＵ，ベクトル演算，ＧＰＧＰＵといった手法が存在する．本研究ではベクトル演算を用いて並列処理を実装する．

2. 1.　ベクトル演算

ベクトル演算とは並列処理を実装するための手法の一つであり，同一の演算を繰り返すような操作を一度に行う事ができる．CPU に搭載されるベクトル演算機能は SIMD(Single Instruction，Multiple Data streams)と呼ばれる． この SIMD を扱う命令セットには，PowerPC の Altivec[1][2][3]，ARM アーキテクチャの NEON[4]，Intel の開発した MMX，SSE，SSE2，AVX，といったものが存在する．本研究では， SSE2 を利用してプログラムの作成を行う．

2.1.1　SSE2 の概要

SSE とは Streaming SIMD Extensions の略称であり，Intel の開発した CPU の SIMD 拡張命令セットの総称である．SSE とは 32bit 単精度浮動小数点用の命令を扱う技術で，SSE2 で新たに倍精度浮動小数点，整数演算を扱えるようになった．SSE2 を用いた画像処理関連の研究として，Harris オペレータによるコーナー検出の高速化[5]やブロック・マッチングの高速化[6]などがあり，フィルタリングにおいても高速化を行えることが期待できる．

2.1.2　SSE2 による SIMD 演算

SSE2 では図１のように，128bit の XMM レジスタに，64bit×2，32bit×4，16bit×8，8bit×16 のデータを格納し，同一の命令に対して一括して処理を行うことができる．

SIMD 命令の大半は，二つの XMM レジスタに保存されているデータ間の演算を行うためのものであり，残りはメモリと XMM レジスタ間のデータ移動や他の機能のためのものである．演算機能として，8 個 16 ビット整数型要素のベクトル，4 個 32 ビット整数型要素等のベクトル間で四則演算（割り算を除く），論理演算や比較などを行える．具体的に，8 個 16 ビット整数型要素のベクトル間の足し算は次の指令で行う．

```
paddw　レジスタ１，レジスタ２
```

この命令の機能は，レジスタ１とレジスタ２の要素間の足し算を計算し，その結果をレジスタ２に保存するものである．これを図で説明すると．図２のようになる．

また，gcc などのコンパイラでは，intrinsic と呼ばれる SIMD の命令を直接 C，C++言語から呼び出す機能がある．一つ例を挙げると，8 個の 16 ビット型整数の足し算を行う場合は，下記のコードを記述すればよい．

```
short a[8]，b[8];
__m128i xa = _mm_loadu_si128((__m128i *) a);
__m128i xb = _mm_loadu_si128((__m128i *) b);
 xa = _mm_add_epi16(xa，xb);
_mm_storeu_si128((__m128i) *)a ，xa);
```

このように，アセンブラ言語で記述することに比べて便利になる上，レジスタの分配などのコンパイラによる最適化が有効になる[7]ので，大いに改善されていると言えるが，C 言語の自然な記述に比べると煩雑であることは明白である．そこで本研究では C++言語のデータ型を自由に設計できる機能を利用して，SIMD 命令で多用されるデータ型に対応する C++のクラスを設計し，SIMD 命令をより，直接的，簡潔に表現できるようにする．具体的には，上記と同様の機能を持つコードは下記のようになり，非常に簡潔に書くことができる．

```
short a[8]，b[8];
s8 x = s8(a) + s8(b);
x．store(a);
```

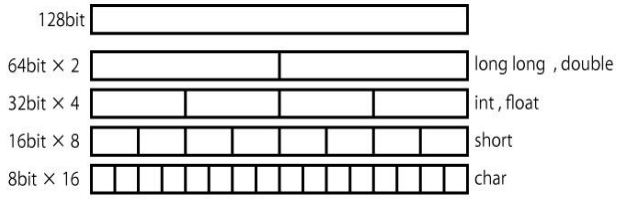

図 1.　XMM レジスタ

‡ 和歌山大学,Wakayama University

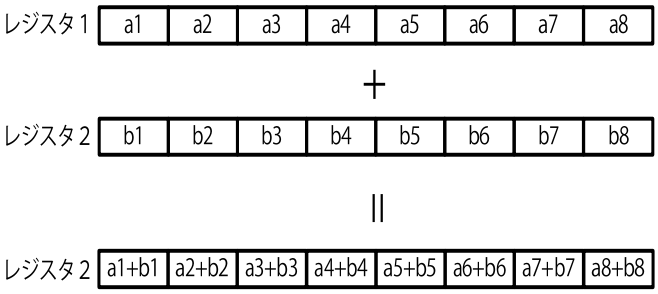

図 2.　8 個の 16bit 整数型要素加算

3. 画像処理におけるフィルタリング

周波数の高い成分，または低い成分を除去する処理は，カーネル関数と入力データとの畳み込み積分によって実現されることが多く，このような処理のことを線形フィルタリング，使用するカーネル関数のことをフィルタと呼ぶ．画像処理では，エッジ検出，ノイズ除去，輪郭強調などのフィルタが存在する．このフィルタリングの具体的な処理の流れを図 3 に示す．入力画像の注目画素，及び近傍の値と，フィルタとの内積を求め，出力画像に代入し，これを対象となる全ての画素に対して行うことで，フィルタリングを行う．この時の計算量は，n 行 m 列のフィルタの場合，乗算 $m \times n$ 回加算 $m \times n - 1$ 回である．

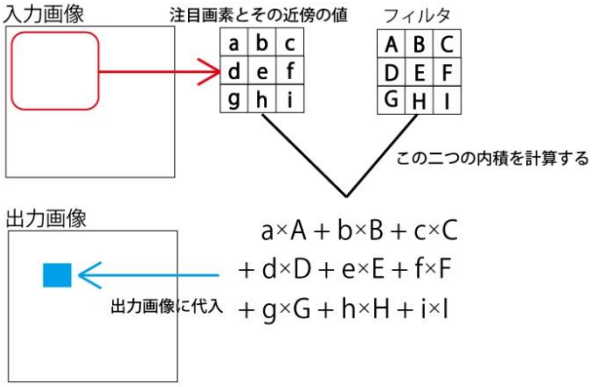

図 3．フィルタリングの流れ

3.1　分離型のフィルタ

また，一部のフィルタは縦横一次元のフィルタに分割し，順次適応することでフィルタリングを行うことができる．この際図 4 のように，分離した縦横一次元のフィルタの各要素間で乗算を行った結果が，元のフィルタの対応する場所の値と同値になっている必要があるこれにより，n 行 m 列のフィルタを 1 行 m 列のフィルタと n 行 1 列のフィルタに分割した場合，計算量を乗算 $m \times n$ 回加算 $m \times n - 1$ 回から乗算 $m + n$ 回加算 $m + n - 2$ 回まで短縮できる．

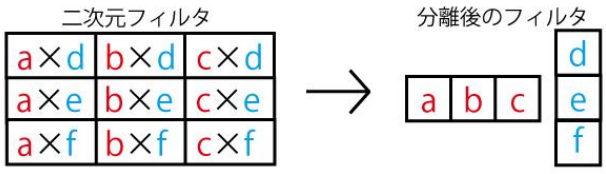

図 4.　分離前と分離後のフィルタ内の値の関係

3．2　ベクトル化によるフィルタリング

ベクトル演算機能を用いてフィルタリングを行った場合，図 5 のようにフィルタの 1 つの要素を取り出し，複数の画素との乗算を同時に行うことができる．これをフィルタの全ての要素に対して実行して乗算結果を加算していき，最終的な合計値を出力画像に代入することで，複数の画素を同時に処理することができる．

また，浮動小数点型のフィルタの場合，浮動小数点型のまま演算するより，整数型に変換して演算を行うほうがより高速に処理することができる．具体的には，フィルタの全値に 2^x を掛けた後 16 ビット整数型に変換して計算を行い，計算結果を x だけ右シフトすることにより合計値を 2^x で割り，出力画像に代入することでフィルタリングを行う．この場合，フィルタを 16 ビット整数型に変換する際小数点以下の値が切り捨てられるので，当然精度は低下する．しかし，画素値は 8 ビット符号無し整数型の範囲である 0 から 255 の整数で表現され，最終的に小数点以下の部分は切り捨てられるため，x の値が十分大きい場合，多少の誤差は問題にならない．

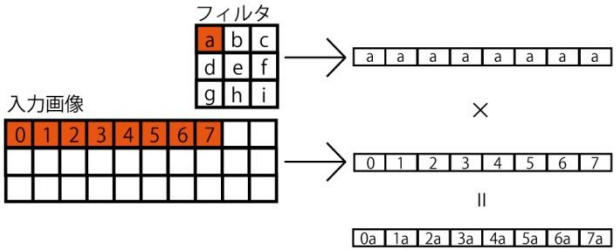

図 5.　ベクトル化によるフィルタリング

4. 評価実験

ベクトル化による並列処理により，フィルタリングの処理時間がどの程度短縮できるのか，評価実験を行った．この実験では，二次元のフィルタを適用するプログラムとして任意のフィルタを適用するプログラム，分離型のフィルタを適用するプログラムとしてガウシアンフィルタを適用するプログラムを選択し，SSE を用いて作成したプログラム，SSE 無しで作成したプログラム，及び有名な画像処理のライブラリである OpenCV を用いて作成したプログラムを用いて比較実験を行う．

この実験に使用したマシンの性能は，OS は Linux3．5．3，CPU は AMD Phenom(tm) 9500 Quad-Core Processor，メモリが 8G となっている．実験は一定の大きさの画像に対して複数の異なるサイズのフィルタを適用する場合と，一定のサイズのフィルタを異なる大きさの画像に適用する二種類のパターンで計測を行った．

4.1　任意のフィルタを適用した結果

5×5 の移動平均フィルタを画像に適用した結果を図 6，任意のフィルタを適用するプログラムに対して，一定の大きさの画像に対して 5 種類のサイズのフィルタを適用した場合の処理時間を表 1，SSE2 を利用したプログラムの処理速度を基準にした場合の速度比を図 7 に示す．SSE2 を用いて作成したプログラムは，SSE2 無しと比較して 9 割程度，OpenCV と比較して 2 から 4 割程度処理時間が短縮されている．

また，1024×1024 の画像に対して 5 種類のサイズのフィルタを適用した際の処理時間を表 2，SSE2 を利用したプログラムの処理速度を基準にした場合の速度比を図 8 に示す．　SSE2 を用いて作成したプログラムは，SSE2 無しと

比較して 9 割以上，OpenCV と比較して 4 から 6 割程度処理時間が短縮されている．

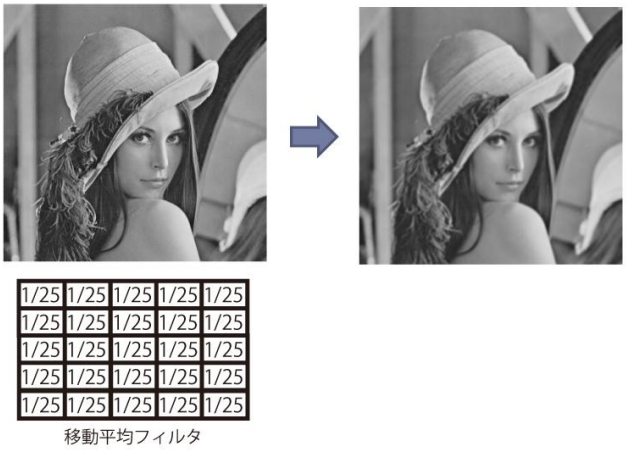

図 6．移動平均フィルタの適用

表 1．複数のフィルタを適用した場合の処理時間

	3×3	5×5	7×7	9×9	11×11
SSE2 無	4.32s	11.04s	19.20s	28.92s	41.56s
OpenCV	0.49s	1.44s	2.51s	3.93s	5.83s
SSE2 有	0.43s	0.86s	1.48s	2.13s	3.02s

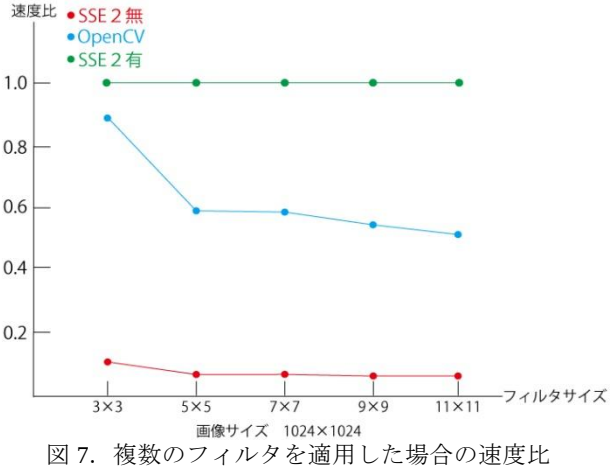

図 7．複数のフィルタを適用した場合の速度比

表 2．複数画像にフィルタを適用した場合の処理時間

	512×512	1024×1024	1536×1536	2048×2048
SSE 2 無	5.08s	19.20s	44.20s	78.16s
OpenCV	0.76s	2.51s	5.42s	9.53s
SSE 2 有	0.34s	1.48s	3.09s	5.48s

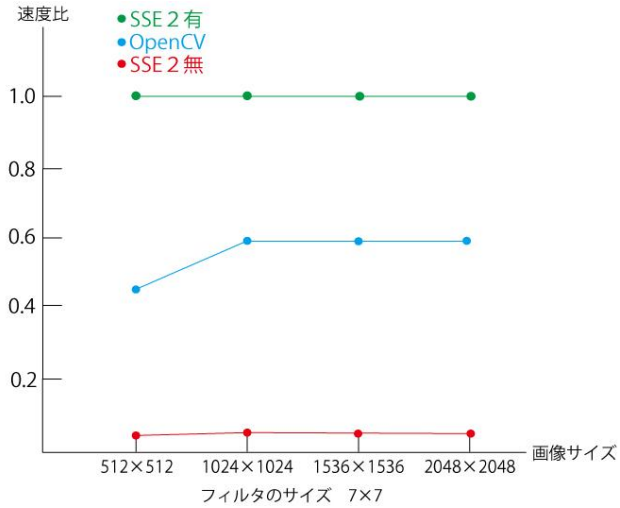

図 8．複数の画像にフィルタを適用した場合の速度比

4.2 ガウシアンフィルタを適用した結果

次にガウシアンフィルタを適用するプログラムに対して，5×5 のガウシアンフィルタを画像に適用した結果を図 9，一定の大きさの画像に対して 5 種類のサイズのフィルタを適用した場合の処理時間を表 3，SSE2 を利用したプログラムの処理速度を基準にした場合の速度比を図 10 に示す． フィルタサイズが小さい場合はベクトル化の恩恵が少なく，OpenCV より処理が遅い部分もあるが，フィルタサイズが大きくなるにつれ改善されている．

また，1024×1024 の画像に対して 5 種類のサイズのフィルタを適用した際の処理時間を表 4，SSE2 を利用したプログラムの処理速度を基準にした場合の速度比を図 11 に示す．

SSE2 を用いて作成したプログラムは，SSE2 無しと比較して 8 割程度，OpenCV と比較して 3 から 4 割程度処理時間が短縮されている．

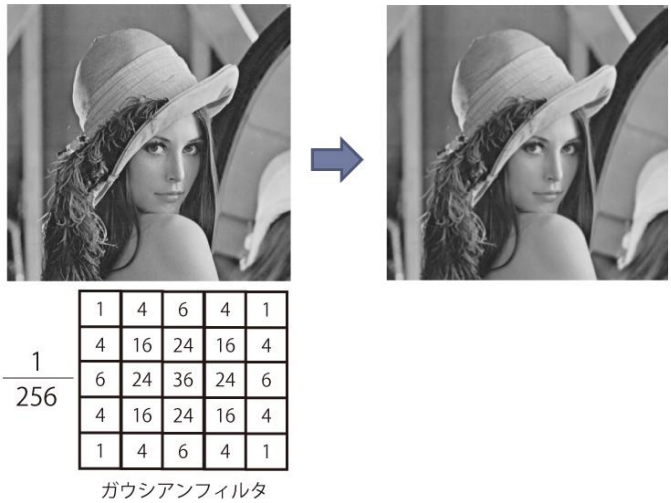

図 9．ガウシアンフィルタの適用

表３．複数のフィルタを適用した場合の処理時間
(ガウシアンフィルタ)

	3×3	5×5	7×7	9×9	11×11
SSE 無	0.94s	1.25s	1.77s	2.10s	4.79s
OpenCV	0.23s	0.38s	0.60s	0.78s	0.85s
SSE 有	0.31s	0.37s	0.43s	0.47s	0.54s

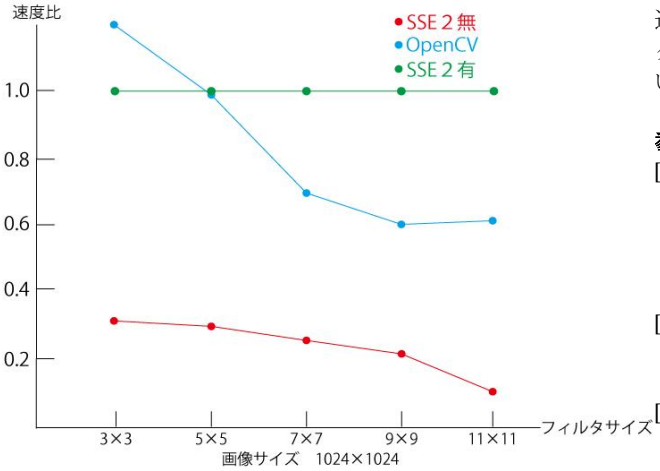

図 10．複数のフィルタを適用した場合の速度比
(ガウシアンフィルタ)

表４．複数画像にフィルタを適用した場合の処理時間
(ガウシアンフィルタ)

	512×512	1024×1024	1536×1536	2048×2048
SSE 無	0.45s	1.77s	3.79s	6.60s
OpenCV	0.16s	0.60s	1.30s	2.77s
SSE 有	0.09s	0.43s	0.93s	1.57s

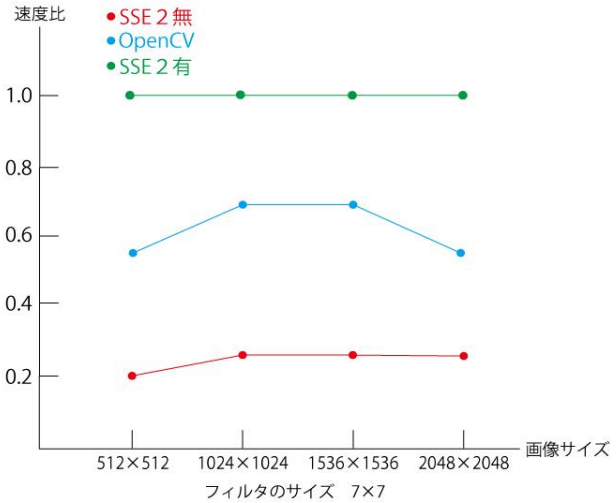

図 11．複数の画像にフィルタを適用した場合の速度比
(ガウシアンフィルタ)

5．まとめ

本研究では，SSE2 を用いたベクトル演算機能の活用により画像のフィルタリングの高速化を行い，その有用性を検証した．SSE2 を用いて作成したプログラムと，一般的な C 言語で作成したプログラムとの処理時間を比較した結果，9 割以上処理時間を短縮することができた，また広く普及している画像処理のライブラリである OpenCV で作成したプログラムと比較しても，大多数の場合で高速化できており，SSE2 によるベクトル化は画像のフィルタリングの高速化に対して，非常に有用な手法であるといえる．

参考文献

[1] K. Diefendorff, P.K.Dubey, R.Hochsprung, H.Scale, “AltiVec extension to PowerPC accelerates media processing, IEEE Micro – Computing & Processing (Hardware/Software)”, Volume 20, pp. 85-95, Issue 2, 2000

[2] Linley Gwennap, “AltiVec Vectorizes PowerPC”, http://mac3.org/a/altivec-vectorizes-powerpc-5-11-98-w7768.html , 1998

[3] “Altivec Technology Programming Interface Manual”, http://www.freescale.com/files/32bit/doc/ref_manual/ALTIVECPIM. pdf , 1999

[4] RealView Compilation Tools, バージョン 4．0 コンパイラリファレンスガイド http://infocenter.arm.com/help/topic/com.arm.doc.dui0348bj/DUI0348BJ_rvct_comp_ref_guide.pdf .

[5] Skoglund, J. and Felsberg, M. (2005). “Fast image processingusing SSE2. In Proceedings of the SSBA Symposium onImage Analysis”, 4 pp.

[6] Intel Corporation, “ストリーミング SIMD 拡張命令２（SSE2）を使用したブロックマッチング，動体予測アルゴリズム バージョン 2.0”, 資料番号 248605J-001, 2000

[7] Intel Corporation, “インテルアーキテクチャ最適化リファレンス・マニュアル”, 資料番号 730795J-001, 1998，1999