I-037

拡張演算命令を用いた画像処理用ドメイン固有言語 LIC の高速化

Acceleration of Domain-Specific Language 'LIC' for Image Processing Using Extended Instruction Set

豊住 耕一† 高橋時市郎†
Kouichi TOYOZUMI† Tokiichiro TAKAHASHI†

1. まえがき

我々は，少ない記述量で高速な画像処理モジュールを開発できる，画像処理用ドメイン固有言語 LIC(Lisp Image Composer)[1]を開発している．文字列処理やネットワーク処理等のドメイン固有言語と比べて，画像処理では扱うデータ量が大きいため，画像処理用ドメイン固有言語には，生成したコードが高速に実行されることが要求される．High Definition (HD)画像が日常的に用いられる今日，一段の高速処理が要求される．そこで，

- 画素間演算や空間フィルタリング等，画像全体を均一処理する画像処理演算の高速化
- インテル製 CPU の SIMD (Single-Instruction Multiple Data)演算を行うストリーミング SIMD 拡張命令(Streaming SIMD Extended Instructions)セットを利用することによる高速化

を図るために，LIC のフレームワークを拡張する．

拡張演算命令により，一層の高速化が可能である．しかし，そのためには，種類が限られている拡張演算命令を複雑に組み合わせて，実現したいコードを生成する技術が必要となる．また，データの並びが処理速度を大きく左右することもある．

そこで，我々は，拡張演算命令を用いた画像処理プログラムを簡潔に記述するための高階関数を実装し，高速化を実現する．代表的な画像処理用ドメイン固有言語 Processing[2]と，画像合成処理速度を比較したところ，平均約 9 倍，最大約 14 倍の高速化を実現したので報告する．

2. 均一な画像処理演算の高速化

プログラムの最適化手法のひとつに，実行頻度の高いコードを重点的に最適化し，全体の性能を高める手法がある．

2.1 map-image 関数系の高階関数としての実装

画像処理の場合，画像全体をラスター走査して，画素間演算や空間フィルタリングを行うような処理が多い．こうした処理は，画素の数だけ繰り返し実行され，しかも画像全体にわたって均一的な処理を行うので，最適化の効果が大きい．こうした処理は，関数型プログラミングにおける map 処理と同じ性質のものである．そこで，ここでは map-image 系関数と呼ぶことにする．

LIC では，map-image 系関数を高階関数として実装し，これを手動で最適化し，高速化を実現した．

2.2 SIMD 演算関数の呼び出しオーバーヘッド低減

LIC の母言語である，Common Lisp は動的プログラミング言語である．高階関数である map-image 関数系の引数として，SIMD 演算を利用する関数を渡す際，関数呼び出しのオーバーヘッドが大きく，SIMD 演算による高速化が相殺されるという問題がある．

そこで，呼び出される関数を実行時にインライン展開するように実装し，関数呼び出しのオーバーヘッドを低減するように実装することとした．

† 東京電機大学大学院未来科学研究科
Graduate School of Science and Technology for Future Life, Tokyo Denki University

3. SSE による高速化

SSE を利用すれば，一命令で複数のデータに対して演算を適用できるため，画像処理を高速に行うことができる．その一方で，画像処理で SSE を利用するには，いくつかの問題がある．本章では，その解決法を述べる．

3.1 SSE の問題点

SSE の問題点は次の 2 点に要約される．

(1) 命令セット

SSE は命令数が少ないため，プログラマは処理を多くの命令の組み合わせとしてコードを書かなければならない．そのため，プログラミング作業が煩雑になる．

その上，SSE の条件分岐命令は動作が難解である．SSE において条件分岐の結果はマスクとして提供されるため，高級言語であれば条件分岐文ひとつで簡潔かつわかりやすく表現できた処理を複数の論理演算によって表現しなくてはならない．

また，SSE で扱える数値の型にはさまざまな種類が用意されているが，型によって利用できる命令が異なる．そのため，プログラマは，値をどの型にどのタイミングで変換して演算するかを考えながらプログラムを書かなければならないので，負担が大きい．

また，型変換には，使えるレジスタの本数が減り，ソースコードの記述量が増えるという別の問題もある．例えば，16bit 整数を単精度浮動小数点数に変換する際には，レジスタを 2 倍使用してしまう．

桁あふれが頻繁に発生することも問題の一つである．桁あふれが発生した際に，型の範囲内に値を丸め込む手法の選択を誤ると，大量の条件分岐を発生させ，実行時性能を低下させる．

(2) ハードウェアリソースとプログラムとの乖離

128bit のレジスタを有する SIMD 演算能力をフルに活用するには，プログラムのみならず，データも再構成する必要がある．

3.2 SSE による高速化

本節では，前節で述べた SSE の問題点を解決する具体的な方策を説明する．実装には，Common Lisp から SSE を利用するために，Gavrilov による Common Lisp 用の SSE イントリンシック命令実装である cl-simd[3]，および Common Lisp 処理系の Steel Bank Common Lisp[4]の Gavrilov による派生版[5]を使用した．さらに，SSE4 の命令を使用するために，種々の改良を加えている．なお，先に開発した LIC と区別するため，新たに開発する LIC を LIC/SSE と呼ぶ．先に開発した LIC は，単に LIC と記す．

(1) 命令セットに関する問題の解決

先に開発した LIC で，単精度浮動小数点数で表現していた画素の各チャネル値を 16bit 整数とした．ただし，実際に演算する際には，単精度浮動小数点数で行う．こうすることによって，命令セットに関するいくつかの問題を同時に解決した．

16bit 整数として表現した画素の各チャネル値を，on-the-fly で単精度浮動小数点演算を行うことにより，演算精度を確保した．結果は 16bit 整数として格納することにより，メモリの使用量を抑えることができる．また，SSE は単精度浮動小数点数用に多くの命令を提供しているから，多くの命令の使用が可能となる．さらに，桁あふれを一つの型変換命令で防ぐこともできる．

先に開発した LIC の関数群を，SSE を利用して演算できるように実装し直した．条件分岐文のうち，頻繁に使われるものを，SSE に対応したコードを生成するように，新たに実装した．

その結果，LIC/SSE は，従来の LIC と同等量のソースコードで画像処理プログラムを実装できる．

(2) ハードウェアリソースとプログラムとの乖離

画素の各チャネル値の並びを変更した．LIC では RGBA という並びであったが，それぞれのチャネルを 8 画素分ずつ，$R_0R_1\cdots R_7G_0G_1\cdots G_7B_0B_1\cdots B_7$ と並べることにした．こうすることで，SSE が使う XMM レジスタのレジスタ長である，128bit をフルに使うことができるので，同時処理可能な最大データ量を処理することができる．

4. 評価

4.1 高速化

今回提案・開発した LIC/SSE，先に開発した LIC，および Processing による画像合成処理のベンチマークテストを行った結果を表 1 に示す．表 1 に示すように，提案手法 LIC/SSE は，Processing に比して平均約 9 倍，最大約 14 倍，高速であった．また，従来の LIC に比して，平均約 23 倍，最大約 47 倍，高速であった．

提案手法は，画像処理用ドメイン固有言語として，十分高速な実行性能を有することが示された．

表 1. 画像合成処理によるベンチマークテストの結果 (単位:ミリ秒，括弧内は提案手法との比)

	Processing		LIC		LIC/SSE
SourceOver	179	(14.9)	25	(20.8)	12
Add	141	(11.8)	571	(47.6)	12
Darken	151	(11.6)	333	(25.6)	13
Multiply	182	(7.9)	583	(25.3)	23
LinerBurn	144	(9.0)	267	(16.7)	16
Lighten	146	(11.2)	281	(21.6)	13
Screen	192	(9.1)	589	(28.0)	21
ColorDodge	229	(2.4)	561	(5.8)	97
LinerDodge	149	(8.3)	321	(17.8)	18
Difference	148	(12.3)	252	(21.0)	12
Subtract	177	(14.8)	55	(45.8)	12
HardLight	196	(1.0)	564	(3.0)	191
平均		(9.5)		(23.2)	1.0

4.2 記述の簡潔さ

リスト 1 に LIC による，リスト 2 には LIC/SSE による覆い焼き合成処理を実装したソースコードの例を示す。リスト 1 およびリスト 2 からは，従来の LIC と LIC/SSE の間において，ユーザが実際に記述するソースコード上に大きな違いがないことがわかる。

```
(defcomposite liner-dodge (bg fg)
  (declare (pixel-t bg fg))
  (new-pixel
    (if (< 1.0 (+ (pixel-red fg)
                  (pixel-red bg)))
        1.0
        (+ (pixel-red fg) (pixel-red bg)))
    (if (< 1.0 (+ (pixel-green fg)
                  (pixel-green bg)))
        1.0
        (+ (pixel-green fg) (pixel-green bg)))
    (if (< 1.0 (+ (pixel-blue fg)
                  (pixel-blue bg)))
        1.0
        (+ (pixel-blue fg) (pixel-blue bg))) 0.0)
```

リスト 1. LIC による覆い焼き合成の実装

```
(def-cp-composer cp-compose-liner-dodge
        (redb greenb blueb redf greenf bluef)
  (values (cpif (cp< @ (cp+ redf redb))
                @ (cp+ redf redb))
          (cpif (cp< @ (cp+ greenf greenb))
                @ (cp+ greenf greenb))
          (cpif (cp< @ (cp+ bluef blueg))
                @ (cp+ bluef blueb))))
```

リスト 2. LIC/SSE による覆い焼き合成の実装

以上の評価結果から，提案手法は高い実行速度と少ない記述量を両立するものであるといえる．

5. むすび

画像処理用ドメイン固有言語 LIC に，SIMD 演算用拡張命令 SSE を用いたフレームワークを実装した．画像合成処理速度を比較したところ，Processing に対して平均約 9 倍，最大約 14 倍，従来の LIC に対して平均約 23 倍，最大約 47 倍，高速であり，SSE を用いた LIC の有用性を示した．

参考文献

[1] Kouichi Toyozumi, Kazuki Kumagai, and Tokiichiro Takahashi, "LIC: A Domain-Specific Language for Video and Image Processing," *Proceedings of International Workshop on Advanced Image Technology*, Jan. 2012.

[2] Casey Reas, Ben Fry, and John Maeda, *Processing: A Programming Handbook for Visual Designers and Artists*.: The MIT Press, 2007.

[3] Alexander Gavrilov. (2012, Sep.) angavrilov/cl-simd. [Online]. https://github.com/angavrilov/cl-simd/

[4] Steel Bank Common Lisp. [Online]. http://www.sbcl.org/

[5] Alexander Gavrilov. (2012, Feb.) angavrilov/sbcl at sse. [Online]. https://github.com/angavrilov/sbcl/tree/sse