愛知教育大学研究報告，56（自然科学編），pp. 13～17, March, 2007

マルチメディア命令による Common Lisp 言語処理系の最適化の一考察

安本太一

情報教育講座

A Study of Optimization for Common Lisp System Using SIMD Instruction Set

Taichi YASUMOTO

Department of Information Sciences, Aichi University of Education, Kariya 448−8542, Japan

1 はじめに

最近の一般消費者向けのコンピュータに使用されている CPU には，マルチメディア命令というものが標準装備されている。このマルチメディア命令は，SIMD（Single Instruction Multiple Data）方式のベクトル演算機能を提供するためのものである。

このベクトル演算機能はアップルコンピュータ社，IBM 社，モトローラ社（現フリースケールセミコンダクタ社）のプロセッサ PowerPC では AltiVec［1］，インテル社のプロセッサでは SSE［2］と呼ばれている。

筆者が知る限り，これらのマルチメディア命令を Lisp 言語処理系で積極的に活用している事例はない。市販されているコンピュータに，標準でベクトル演算機能が備わっているのに，Lisp 言語処理系では利用されていないのが現状である。

Lisp 言語処理系において，プロセッサ内蔵型のベクトル演算機能を使用することの可否，使用できる場合の実装方法，使用できた場合の速度向上のデータが欠如している状態である。そこで，Common Lisp 言語［3］の処理系である KCL（Kyoto Common Lisp）［4，5］において，マルチメディア命令を使って組み込み関数を最適化することを試みた。

本論文では，PowerPC を搭載したコンピュータ上で，Common Lisp 言語の組み込み関数 max をベクトル演算機能を使って最適化した事例を紹介する。簡単な事例ではあるが，その実装は可能な限りの速度向上を目指したものであり，プロセッサ内蔵型のベクトル演算機能を用いた現実的な速度向上の上限を示唆するものである。

2 本研究の位置付け

過去の研究として，湯淺と安本らによって，SIMD 方式の超並列計算機能を Lisp 言語処理系に付加した報告が行われている［6，7］。

しかしながら，この研究は，1000以上のサブセット Common Lisp 言語処理系があって，これをフロントエンドのフルセット Common Lisp 言語処理系が制御するという新しい計算モデルを導入している。想定しているコンピュータは，1000以上の PE（Processing Element）を装備している高価なものである。

本研究の目標は，対象としているコンピュータも計算モデルもこの報告とは異なり，一般消費者が購入できるコンピュータの CPU に内蔵されているベクトル演算命令を活用し，新しい計算モデルを導入しないで（Common Lisp 言語の仕様を変更しないで），どこまで速度向上が得られるかを探ることである。

3 プロセッサに内蔵されたベクトル演算機能の概要

本論文の冒頭では，プロセッサに内蔵されているベクトル演算機能として AltiVec と SSE を挙げたが，ここでは AltiVec について説明する。これら 2 つの機能について詳細に違いはあるが，基本的な機能は共通しており，AltiVec を用いたプログラムを SSE 向けに書き直すための文献も発表されている［8］。本論文の議論では，AltiVec による説明で差し支えない。

・並列計算機能

ベクトル演算用に，128ビットのベクタレジスタを32個用意する。128ビットのベクタを主に次のように利用する。

―4 分割し，32ビットの整数（符号付きあるいはなし）あるいは32ビットの浮動小数点数の演算を 4 つ同時に実行，

―8 分割し，16ビットの整数（符号付きあるいはなし）の演算を 8 つ同時に実行，

—16分割し，8ビットの文字の演算を16同時に実行，

例えば，4つの32ビット整数が含まれたベクタA，Bを用意し，ベクタの加算命令 vec_add を実行すると，ベクタA，Bを加算した結果をベクタCに格納するといったことができる。

上記に掲げた様々なデータ型や，最大値を求めるといった命令のバリエーションがあって，ベクトル演算のために命令が計162個追加されている。

並列度は，演算の対象となるデータの大きさ8ビットから32ビットに応じて，16から4になる。

・データのアラインメント

ベクトル演算の引数や結果の格納先になるベクタレジスタは，128バイト境界に配置される必要があるといった制限がある。

4 Lisp言語処理系へのプロセッサ内蔵型ベクトル演算機能の適用可能性

一般に，プロセッサに内蔵されているベクトル演算機能を，Lisp 言語処理系に利用するには，次の点で不都合があると思われる。

・データの構造の不一致

Lisp のデータは，基本的にはヘッダとデータ本体から構成されているので，ベクタの要素となっているC言語のデータ型と合わない。整数のデータは，ビット数が固定長の整数として表現される場合もあり，その場合はベクタの要素となりえるが，絶対値が大きくなって無限長整数に昇格するとベクタの要素にはなりえない。

・データのアライメント

Lisp の主要なデータ構造であるリストは，セルがポインタでつながれているので，個々のセルは不連続に並ぶ。したがって，ベクトル演算機能をリストの要素に対して使いたい時は，演算のために一時的に連続した領域を確保し，その領域にコピーするといった準備が必要である。ただし，このような準備は，オーバヘッドになり，ベクトル演算機能による速度向上が得られない恐れがある。

5 ベクトル演算機能の KCL への適用

前節では，一般論としては，Lisp 言語処理系で，ベクトル演算機能を適用することは困難だと述べた。しかし，特定の Lisp 言語処理系，ここでは KCL に対象を限って，プロセッサ内蔵型ベクトル演算機能の適用可能性を検討する。

KCL では，即値データといってポインタの中にデータオブジェクトを埋め込む表現方法を用いたデータ型があったり，Lisp の関数への引数の受渡しにおいてスタックを使っているからである。

5.1 KCL の即値データ

Lisp のデータオブジェクトは，通常，そのオブジェクトへのポインタで参照されるが，ポインタ内に直接オブジェクトをコーディングする方法があり，このようにコーディングされたオブジェクトを即値データという。KCL では，この即値データを，固定長整数(fixnum)，単精度浮動小数点数(short-float)，文字(character)として実装している［9］。そのフォーマットを図1に示す。

図1 KCL における Lisp のデータオブジェクトの表現

これらの即値データとして実現されているデータ型は，正味のデータの長さが32ビット以下であるので，ベクタの要素になりうる。即値データとして実現されているデータ型は，通常の32ビットプロセッサにおけるデータ表現と近く，固定長整数，単精度浮動小数点数の仮数部が2ビット小さくなっているだけである。文字は，2バイト文字も扱う場合は16ビットの整数とみなし，1バイト文字のみの場合は16ビットの領域の下位8ビットだけ使い8ビットの整数とみなせる。

ポインタのビット長も32ビットなので，32ビットの整数とみなせば，ベクトル演算の対象となる。

ベクトル演算の対象として適当なのは，並列演算を続けても整数のように昇格などのない，比較，ビット演算，シフト，最大値，最小値である。

5.2 KCL の関数呼び出し

KCL の関数呼び出しでは，引数はスタックに積まれる。積まれるものは，図1に示す即値データあるいはポインタである。各引数は連続した領域に配置される。ただし，最初の引数が，128ビット境界に配置されるとは限らない。

引数スタックは，C言語的には，32ビットのポインタの配列とみなせる。そして，キャストによって，この配列の要素4個分を，ベクトル演算のための1つのベクタとみなすことができる。ただし，その4個のうちの先頭の要素のアドレスは128ビット境界でなければならない。

固定長整数や単精度浮動小数点数が，特に，AltiVec の並列命令の対象として適していると考えられる。こ

れらのデータオブジェクトは，右に２つシフトするだけで，正味のデータが得られるからである。

6　KCL における組み込み関数 max の並列化

与えられた引数の最大値を求める組み込み関数 max をベクトル演算機能を使って，高速化することを試みた。

1．引数のデータ型のチェック
引数が全て即値データの固定長整数かどうかをチェックする。すべて固定長整数でなければ，従来のオリジナルの max のコードを実行する。

2．引数のアラインメントの調整
最初の引数のアドレスが128バイト境界から始まっているか否かをチェックする。128バイト境界から始まっていなければ，図２のように最初の引数から128バイト境界までの引数を最後の引数の後にコピーする。引数の先頭を示すポインタは，最初に到達する128バイト境界を指すように変更する。引数の順番が変わってしまうが，max は最大値を求める関数なので，差し支えない。

3．引数の個数の調整
引数の個数が４の倍数か否かをチェックする。４の倍数でないときは，図３のように，４の倍数になるのに足りない個数だけ，ダミーの引数として表現可能な最小値を示すデータを付け加える。
このようにすることによって，見かけ上の引数のバイト長が AltiVec のベクタの長さである128の整数倍となる。C 言語のキャストにより，引数４つ分を AltiVec の一つのベクタとみなし，AltiVec のベクトル演算を全ての引数に適用できる。
ダミーの引数は最小値なので，max の結果に影響を与えない。引数の数が４の倍数になっていない場合，倍数に満たない端数に応じた場合わけするコーディングも可能であるが，コードが汚くなる。また，端数が２，３の場合は，ダミーで補ってベクトル演算をする価値はある。

4．ポインタ表現から正味のデータへの復元
行わない。整数の大小比較なので，右に２つシフトして元にもどさずにそのまま比較する。したがって，Lisp のデータから C 言語へのデータへの変換に伴うオーバヘッドはない。

5．最大値の探索
図４のように，引数４つ分を一つのベクタとみなし，ベクトル演算命令 vec_max を適用する。４つの整数を含むベクタにおいて，添字が同じ要素を比較し，大きい方の数を得る。これを繰り返す。その結果，ベクタ０の０～３番目のいずれかの要素が最大値である。図５のように，ベクタ０を作業用ベクタに右に64ビットシフトしてからコピーし，ベクタ０と作業用ベクタを引数に vec_max

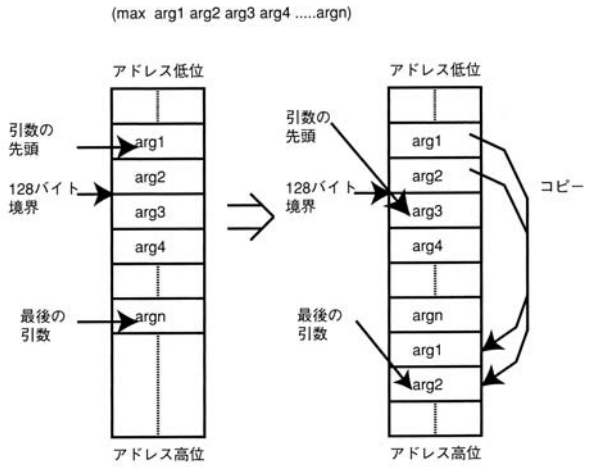

図２　128バイト境界に合わせるための引数のコピー

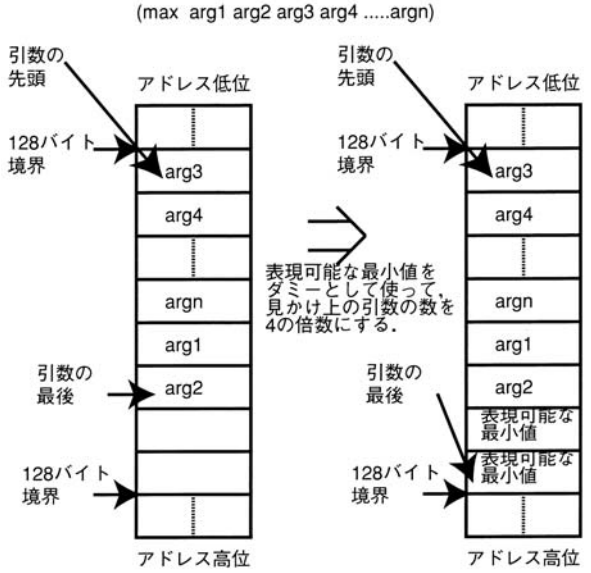

図３　引数の数の調整

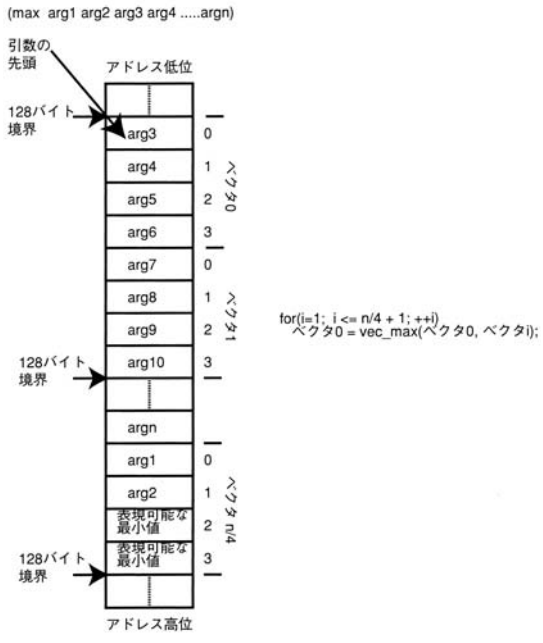

図４　最大値の探索（１）

を適用する。それぞれのベクタの２番目と３番目を同時に比較し，それぞれの大きい方の数をベクタ０の２番目と３番目に格納する。２番目と３番目の比較を通常の大小比較で行い，大きい方が最大値である。

7 評　　価

7.1 評価実験

並列化したmaxの速度向上を計測した。(max 1 2....14999　15000）を10000回実行し，オリジナルのmaxの場合と並列化したmaxの場合の実行時間を比較した。使用した計算機は，アップルコンピュータ社製Power Mac G5（PowerPC 970 2GHz×２，メインメモリ２GB，一次命令キャッシュ64KB，一次データキャッシュ32KB，二次内部キャッシュ512KB，OSはMac OSX10.4.7）である。

10000回の繰り返しにはdotimesを使っているが，コンパイルしてから実行しているので，ifとgotoによって実現されたループにおいて(max 1 2....14999 15000）が実行されている。

実行時間を表１に示す。

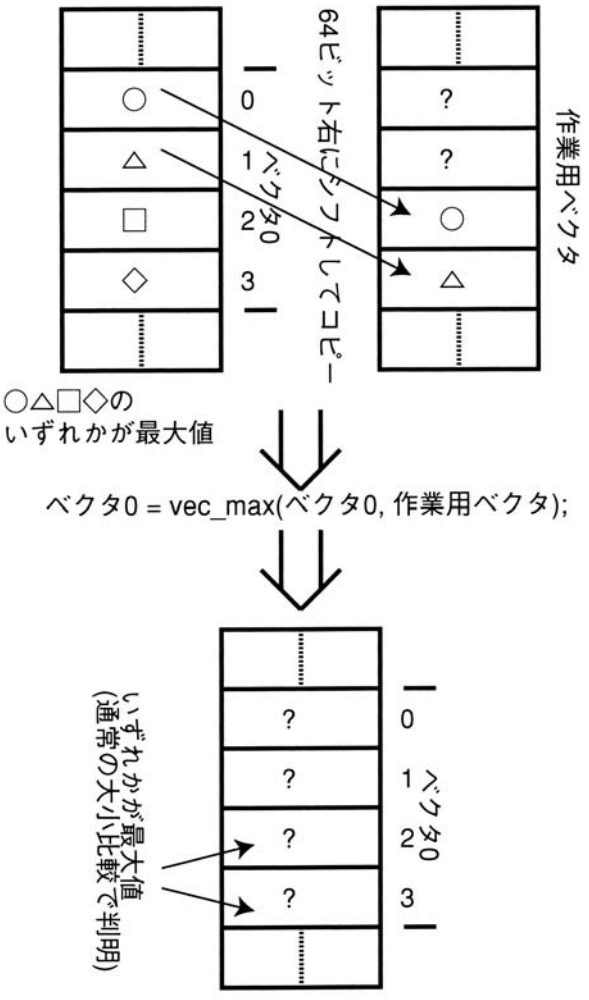

図５　最大値の探索（２）

表１　オリジナルのmaxと並列版のmaxの比較

maxのバージョン	実行時間	比
オリジナル(逐次版)	5.333	1.0
並列版	2.66	0.499
並列版(２分割)	2.48	0.465
並列版(４分割)	2.42	0.454

(単位は秒)

なお，並列版（２分割）や並列版（４分割）は，パイプラインやアウトオブオーダの効果をねらったものである。最大値を探索しているところで，

```
for（i＝１；i＜n；++i）
  ベクタ０＝vec_max（ベクタ０，ベクタi）；
```

としているのを，ループの回数を減らし，前半の最大値と後半の最大値を各繰返し毎に求めるようにしたのが，２分割である。

```
for（i＝１；i＜n/２；＋＋i）｛
  ベクタ０＝vec_max（ベクタ0,ベクタi）；
  ベクタn/２
    ＝vec_max（ベクタn/２，ベクタn/２＋i）；
｝
ベクタ０＝vec_max（ベクタ０，ベクタn/２）；
```

４分割も同様である。

7.2 考察

並列版の実行時間は,オリジナルの約半分となった。ベクトル演算のための準備として，引数が全て固定長整数か否かのチェック，引数のアドレスのアラインメントの調整，ベクトル演算適用のための引数の調整をやったり，最大値の候補である４つの整数から最大値を求めるリダクションをやったりしているので，理想値の４分の１にはならないのだろう。

並列版（２分割）や並列版（４分割）では，実行時間がさらに短縮されている。素朴な並列版では，まだ整数ベクトル演算ユニットに余裕があり，独立に並列計算できる命令を続けて供給するようなコーディングをすることよって，ある程度の速度向上が可能なことがわかる。ベクトルレジスタへのデータのロードなどにコストがかかることを考えれば，命令キューにおいて並列命令が可能な限り途絶えないようにすることが，速度向上を得るために重要である。AltiVecにおいて，128ビットのベクトルレジスタが32本もあるのは，可能な限り前もって必要なデータをベクトルレジスタにロードしておいて欲しいということだろう。

8 ま　と　め

Lisp言語処理系を，最近のプロセッサに搭載されているマルチメディア命令を使って，最適化することを試みた。

マルチメディア命令のベクタの要素がサポートしているデータ型で表現できる，Lisp のデータ型があれば，マルチメディア命令を使うことができる。ただ，その場合も，ベクタにデータをセットするコストが高ければ，速度向上の恩恵は得られないだろう。

本論文では，KCL という Lisp 言語処理系において，即値データで表現された固定長整数を対象に，ベクタに Lisp のデータオブジェクトをセットするコストを可能な限り抑える形で，組み込み関数 max の最適化を試みた。理論上並列度最大 4 の場合で，実行時間がおおよそ半分になった。Lisp の関数呼び出しの引数を，コピーもせずそのまま使っているくらいなので，この速度向上 2 倍の数値は，現実的な条件下における速度向上の上限だと思われる。

Common Lisp は，組み込み関数の数が多いこと，コンパイルを前提とした型宣言が可能なことを考えると，マルチメディア命令の適用範囲は多く見つけれると思われる。ポインタのビット長が，マルチメディア命令のベクタの要素となりうる範囲で納まっているならば，データオブジェクトの同一性の判定や，メモリ管理に，マルチメディア命令を適用できる可能性はある。

今後の課題としては，マルチメディア命令のさらなる適用範囲を探ること，今回は取り扱わなかった SSE などの他のプラットフォームを使って性能を評価することである。

参 考 文 献

［1］Apple Computer, inc.: *Velocity Engine Introduction and Overview*, http://developer.apple.com/hardwaredrivers/ve/.

［2］大原雄介：図解64ビットがわかる，技術評論社（2006）.

［3］Steele, G. L. Jr. : *Common Lisp the language*, Digital Press（1984）.

［4］Yuasa, T. and Hagiya, M. : *Kyoto Common Lisp Report*, Teikoku Insatsu Publishing（1985）.

［5］Yuasa, T. : Design and Implementation of Kyoto Common Lisp, *Journal of Information Processing*, Vol. 13, No. 3, pp. 284–295（1990）.

［6］湯淺太一，安本太一，永野佳孝，畑中勝美：TUPLE: SIMD 型超並列計算のための拡張 Common Lisp，情報処理学会論文誌，Vol. 35, No. 11, pp. 2392–2402（1994）.

［7］安本太一，湯淺太一，貴島寿郎：SIMD 型超並列計算機上の拡張 Common Lisp 処理系におけるごみ集めとその評価，情報処理学会論文誌，Vol. 35 No. 12, pp. 2569–2580（1994）.

［8］Apple Computer, inc.: *AltiVec/SSE Migration Guide*（2005）.

［9］湯淺太一，安本太一：KCL における即値データの実装とその評価，電子情報通信学会春季全国大会講論集，D–357（1989）.

（平成18年 9 月19日受理）