Bulletin of Aichi Univ. of Education, 66（Natural Sciences）, pp. 21 − 24, March, 2017

ARM NEON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 ARM NEON

Taichi YASUMOTO

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

1 はじめに

Raspberry Pi 2のCPUから搭載されているSIMD（Single Instruction Multiple Data）方式並列演算命令ARM NEON［1, 2］の評価を，Common Lisp［3］言語処理系であるKCL（Kyoto Common Lisp）［4, 5］の組み込み関数の並列化を通じて行ったので，その結果を報告する．

今日，パーソナルコンピュータはもちろんのこと，スマートフォン，タブレット端末のCPUにも，SIMD方式並列演算命令が搭載されるようになった．筆者は，過去に，IBM PowerPCのSIMD命令AltiVec［6］やIntel Core iのSIMD命令AVX, AVX2［7］を用いて，KCLの組み込み関数の並列化とその評価を行っている［8, 9, 10］．PowerPCやIntel Core iは，パーソナルコンピュータ用のCPUである．AltiVecとAVX, AVX2では，用意されているSIMD命令の違いにより，若干異なる手法で，組み込み関数の並列化を行った．

一方，今回対象としたCPUは，スマートフォンやタブレット端末などで採用実績の高いARMのCortex-A53（ARMv8-Aアーキテクチャのクアッドコア）である．NEONは，ARMv8-Aに搭載されているSIMD方式並列演算命令である．NEONには，過去の研究において，AltiVecで使った種類の命令と，AVX, AVX2で使った種類の命令の双方が含まれているので，1つのCPUで，過去の研究で用いた2つの手法を比較できる．

Cortex-A53を搭載した実機としては2016年2月に発表されたRaspberry Pi 3を用いた．Cortex-A53は64ビット CPUであるが，Raspberry Pi 3の公式OSでは，32ビットモードで使用されている．加えて，今回の評価実験には，32ビットCPUであるARM Cortex-A7（ARMv7-Aアーキテクチャのクアッドコア）を搭載したRaspberry Pi 2（2015年2月発表）も用いた．ARMv7-AでもNEONが使用できるが，ARMv8-Aと比べて命令同時実行の性能が低いという違いがある．2世代のRaspberry Piを用いることによって，筆者が提案しているSIMD命令を用いた高速化の効果が，パイプライン処理と複数命令同時実行の貢献の違いによって，どのような影響を受けるか比較できる．

以上のような理由から，Raspberry Pi 3とRaspberry Pi 2のNEONを用いたCommon Lisp言語処理系の組み込み関数の最適化の考察を行うことにした．

2 過去に行ったSIMD命令による組み込み関数の並列化の概要

筆者の過去の研究では，最大値を求める組み込み関数maxをCPU内蔵のSIMD命令を用いて並列化して，実行効率の向上を評価した．ここでは，過去の研究について，本論文の理解に必要な最低限の説明を行う．

AltiVecのSIMD命令を対象とした時は，32ビット整数4つのベクタ（128ビットのベクタ）を引数として2つ取り，対応する要素のうち大きい方を要素とするベクタを返すSIMD命令vec_maxを用いた．組み込み関数maxの引数を4つごとにまとめ，SIMD命令vec_maxに適用した．最大値の候補4つからなるベクトルとmaxの引数4つからなるベクトルをvec_maxの引数にして，新たな最大値の候補4つを求めるという繰り返しをループで行なっていた．加えて，即値データ［11］として実装されている固定長整数をそのままvec_maxの引数とする手法で，固定長整数を引数とする場合（例:(max 1 2 .... 14999 15000)を10000回実行）の実行時間が逐次版に対して約45%から50%に短縮された．スタックに格納されたmaxの引数をそのまま4つずつ，vec_maxを渡すのだが，vec_maxには先頭のアドレスが128バイト境界になるように渡し，引数の個数が4の倍数でない場合は固定長整数の最小値

で埋めて4の倍数にしていた．

AVX2のSIMD命令を対象とした時は，64ビット版のKCLを対象としたので，64ビット整数4つのベクタ（256ビットのベクタ）を引数とするSIMD命令を用いた．64ビット整数4つのベクタ(256ビットのベクタ）を引数として2つ取り，対応する要素のうち大きい方を要素とするベクタを返すSIMD命令がなかったので，他の複数のSIMD命令（比較命令，AND（論理積）命令，AND NOT（否定論理積）命令，OR（論理和）命令）を組み合わせて，大きい方を要素とするベクタを得る操作を実現した．その他は，AltiVecの時と，ほぼ同様である．その結果，固定長整数を引数とする場合（例:(max 1 2 .... 14999 15000)を10000回実行）の実行時間が逐次版に対して約41%から44%に短縮された．

3 NEONにおけるSIMD演算命令

NEONには，AltiVecのvec_maxに相当する命令vmaxq_s32が用意されている．C言語のプロトタイプ宣言ではint32x4_t vmaxq_s32(int32x4_t a, int32x4_t b)である．int32x4_tは32ビット符号付き整数4つのベクタである．過去の研究で，AltiVecのvec_maxを用いていたところは，ほぼそのままvmaxq_s32で置き換えることができる．

また，NEONには，AVX2の64ビット整数4つのベクタに対する_mm256_cmpgt_epi64に対応する命令そのものはないが，32ビット整数4つのベクタに対する命令vcgtq_s32が用意されている．C言語のプロトタイプ宣言ではuint32x4_t vcgtq_s32(int32x4_t a, int32x4_t b)である．同様に，_mm256_and_si256はvandq_u32に，_mm256_andnot_si256はvbicq_u32に，_mm256_or_si256はvorrq_u32に対応している．uint32x4_tは32ビット符号なし整数4つのベクタである．64ビット版から32ビット版への書き換えはあるが，過去の研究でAVX2の_mm256から始まる名前の命令を用いていたところは，上記の対応により，ほぼそのままNEONのvから始まる名前の命令で書き換えることができる．

4 Raspberry Pi 3の概要と関数maxの並列化

Raspberry Pi 3に用いられているCPUチップは，Broadcom BCM2837で，クロックは1.2GHz，主記憶は1GBである．ARMv8-Aは64ビットCPUであるが，速い32ビットCPUとして利用している．現時点では，Raspberry Pi Foundationから提供されている公式OSRaspbianが32ビット版のみだからである．Raspbianは，Raspberry Pi向けに最適化されたLinuxである．Raspbianカーネルバージョン Jessie4.4はRaspberry Pi 3とRaspberry Pi 2の双方に対応している．Cコンパイラgccのバージョンは4.9.2である．Raspbian Jessie4.4（とプログラムの実行ファイル）が入ったmicroSDカードは，Raspberry Pi 3とRaspberry Pi 2のどちらに挿入しても利用できる．

したがって，本研究では，32ビット版のKCLをRaspbianに移植し，SIMD命令も32ビットデータ4つからなるベクタを対象としている．KCLを選定したのは，KCLがGNU Common Lisp（GCL）の元となっているなど研究者の間で広く知られ，その実装が論文等で公表されていること，筆者が即値データの実装を行ったこと，KCLがCとCommon Lispで記述されていて移植しやすいこと，本研究ではNEONをCから利用しているので成果が他のLisp言語処理系の実装の参考なるからである．

KCLの移植作業は，即値データ対応を含めて，大きな変更なく行うことができた．これは，即値データを含めたKCLの高い移植性を示している．過去の研究で，AVX2を対象とした手法は64ビットデータ4つからなるベクタを用いたが，NEONに適用する場合はアルゴリズムはそのままで32ビットデータ4つからなるベクタ利用に修正した．

評価実験において，CPUの違いよる実行効率の比較のために用意したRaspberry Pi 2のCPUチップは，Broadcom BCM2836で，クロックは900MHz，主記憶は1GBである．ARMv7-Aは32ビットCPUである．Raspbianは，Raspberry Pi 3と共用のJssie4.4（gccは4.9.2）とRaspberry Pi 2用のWheezy3.18（gccは4.6.3）が利用できる．

maxの並列化は，2つのバージョンで行った．一方は，素直にvmaxq_s32を使うもので，max版ということにする．他方は，vcgtq_s32, vandq_u32，vbicq_u32, vorrq_u32を組み合わせて使うもので，cgt版ということにする．

5 評価実験

NEONのSIMD命令による最適化の有無，最適化の方法max版とcgt版の違いによって，組み込み関数maxの実行時間がどのように変化するか，実験を行った．過去の研究と同様，(max 1 2 .... 14999 15000)を10000回実行した時の実行時間を計測した．Raspberry Pi 3で計測した結果のうち，max版を表1に，cgt版を表2に示す．

並列版（2分割）や並列版（4分割）は，過去の研究[8, 10]と同じで，maxの引数を2分割あるいは4分割して，ベクトル演算命令を2回あるいは4回続けて行う代わりに，ループの数を（通常の並列版に対して）2分の1あるいは4分の1に減らすものである．その狙いは，機械語レベルにおける命令実行順序を（通常の並

表1：オリジナルのmaxと並列版のmaxの比較
ARM Cortex-A53 @ 1.2GHz max版Jssie

maxのバージョン	実行時間	比1	比2
オリジナル（逐次版）	8.350	1.0	
並列版	3.600	0.431	1.0
並列版（2分割）	3.590	0.430	0.997
並列版（4分割）	3.590	0.430	0.997

（単位は秒）

表2：オリジナルのmaxと並列版のmaxの比較
ARM Cortex-A53 @ 1.2GHz cgt版Jssie

maxのバージョン	実行時間	比1	比2
オリジナル（逐次版）	8.350	1.0	
並列版	3.820	0.457	1.0
並列版（2分割）	3.790	0.454	0.992
並列版（4分割）	3.690	0.442	0.966

（単位は秒）

表3：オリジナルのmaxと並列版のmaxの比較
ARM Cortex-A7 @ 900MHz max版Jssie

maxのバージョン	実行時間	比1	比2
オリジナル（逐次版）	12.250	1.0	
並列版	5.300	0.433	1.0
並列版（2分割）	5.280	0.431	0.996
並列版（4分割）	5.270	0.430	0.994

（単位は秒）

表4：オリジナルのmaxと並列版のmaxの比較
ARM Cortex-A7 @ 900MHz cgt版Jssie

maxのバージョン	実行時間	比1	比2
オリジナル（逐次版）	12.260	1.0	
並列版	5.940	0.485	1.0
並列版（2分割）	5.940	0.485	1.0
並列版（4分割）	5.810	0.474	0.978

（単位は秒）

列版から）変えて，パイプライン処理や命令同時実行の効果を期待したものである．

1つのSMID命令で行うmax版の方が，複数のSIMD命令を組み合わせるcgt版より実行時間短縮の効果が大きい．cgt版も並列化の効果が十分得られている．通常の並列版から，2分割，4分割と進めていく時の実行時間の短縮の割合は，cgt版の方が大きい．

次に，Raspberry Pi 2上で，同様な計測を行った．OSのイメージ，KCLの実行ファイルのイメージは，Raspberry Pi 3の場合と同じである．max版を表3に，cgt版を表4に示す．

こちらの場合も，max版の方が，cgt版より実行時間短縮の効果が大きい．通常の並列版から，2分割，4分割と進めていく時，cgt版は2分割の時に実行時間短縮が得られなかった．この2分割の時の状況に固有なCPUにおける命令同時実行の事情が原因なのかもし

表5：オリジナルのmaxと並列版のmaxの比較
ARM Cortex-A7 @ 900MHz max版Wheezy

maxのバージョン	実行時間	比1	比2
オリジナル（逐次版）	12.000	1.0	
並列版	5.900	0.492	1.0
並列版（2分割）	5.880	0.490	0.997
並列版（4分割）	5.860	0.488	0.993

（単位は秒）

表6：オリジナルのmaxと並列版のmaxの比較
ARM Cortex-A7 @ 900MHz cgt版Wheezy

maxのバージョン	実行時間	比1	比2
オリジナル（逐次版）	12.110	1.0	
並列版	6.410	0.529	1.0
並列版（2分割）	6.360	0.525	0.992
並列版（4分割）	6.330	0.523	0.988

（単位は秒）

れない．4分割の時は，cgt版において実行時間の短縮が得られ，時間短縮の割合がmax版より大きい．

cgt版では2分割の時に実行時間の短縮が得られなかったことがきっかけで，Raspberry Pi 2で使用するOSをWheezy3.18に変えて計測した．KCLのコンパイルをgcc4.6.3でやり直した．Jssie4.4上のgcc4.9.2でコンパイルしたKCLの実行ファイルは，Wheezy3.18上では実行できなかったからである．max版を表5に，cgt版を表6に示す．

Whezzyでgcc4.6.3の場合は，並列版，並列版（2分割），並列版（4分割）の実行時間が，Jssieでgcc4.9.2の場合より実行時間が長くなっているが，2分割，4分割と進めるにつれて，実行時間が短くなっている．cgt版の方が時間短縮の割合が大きい．

6 考察

Raspberry Pi 3上で，NEONのSIMD命令を用いた場合のmaxの並列版の実行時間は逐次版の43%から44%ぐらいで，おおざっぱに言えば，過去のAltiVecやAVX2の研究と同程度であった．

1命令で済むmax版の方が実行時間が短いが，複数命令を組み合わせるcgt版も十分な実行時間短縮が得られており，両者の差が著しく大きいわけではない．その理由として，cgt版は（SIMD命令の組み合わせの間に分岐命令がないので）パイプライン処理がうまくいっていることと，1命令で済むvmaxq_s32は，単純な命令である大小比較vcgtq_s32や論理積vandq_u32などよりは，多くのサイクルを要していることが推測される．

並列版（2分割）と並列版（4分割）と進めると実行時間の短縮の割合が，cgt版の方がmax版に比べて

大きいのは，複数の単純な命令から構成されているため，同時実行できる命令の組み合わせが見つかる可能性が高いと推測される．これに対して，max版の`vmaxq_s32`は，命令が複雑で，同時実行できる命令の組み合わせが見つかる可能性は低いと推測される．ARM Cortex-A53（ARMv8-A）の方が，cgt版において，ARM Cortex-A7（ARMv7-A）より実行時間短縮の割合が明らかに高いのは，文献によるとA53の方がA7より2つの命令同時実行の性能が良い［12］からだと推測される．

7 まとめ

過去に提案したSIMD命令を使って組み込み関数`max`を高速化する手法を，ARMアーキテクチャのSIMD命令NEONにおいて適用し，約2倍強の実行効率の向上が得られた．実行効率の向上は，`max`の引数として与えられたLispの即値データをそのままSIMD命令ベクタとして扱えることが大きく寄与している．過去のPowerPCのAltiVec, x64（x86-64）のAVX2に加え，今回のARMのNEONで，計3つのアーキテクチャに適用できたことで，提案してきた手法は汎用性があるといえる．

本稿では`max`を対象に述べたが，NEONは，`min`はもちろんのこと，論理演算（`logand`, `logior`, `logxor`）や整数の算術演算（+, *）にも適用できる．即値データとして表現できるデータ型に対応するSIMD演算がNEONに存在すれば本稿で述べたような最適化が可能であり，単精度浮動小数点数の`max`と`min`，算術演算（+, *）にも適用できる．

SIMD命令の使い方には，大きな粒度の命令を1つ用いる方法と，粒度が小さい複数の命令を組み合わせて粒度の大きな命令と等価なことを実現する方法の2通りがあることがある．通常は，前者の方が後者より実行時間の短縮の効果は大きい．ループの回数を減らす代わりにSIMD命令が連続して実行されるように工夫した場合，工夫をしなかった時に対する実行時間の短縮の効果は，同時命令実行の効果によるものなのか，後者の方が高くなる．粒度の小さいSIMD命令の方が，命令同時実行ができる組み合わせが見つかる可能性が高いのだろう．`max`の事例では，後者が前者より実行時間が短くなるところまでは達しなかったが，粒度の小さいSIMD命令を組み合わせた方が1つのSIMD命令より実行効率が良い事例を探ることが今後の課題である．

Cortex-A53は，今回用いた32ビットモードAArch32の他に，64ビットモードAArch64があり，命令セットが異なる．Raspberry Pi 3の公式OS Raspbianは，現在32ビット版のみであるが，64ビット版が提供される機会があれば，提案している手法を適用したらどのようになるか検証したい．

参考文献

［1］ ARM: *Cortex-A7 NEON Media Processing Engine Technical Reference Manual*, ARM DDI 0462F ID051113 (2013).

［2］ ARM: *ARM NEON Intrinsics Reference*, ARM Document number IHI 0073A (2014).

［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］ NXP Semiconductors: *AltiVec Technologies*, http://www.nxp.com/pages/altivec-technologies:DRPPCALTVC.

［7］ Intel Corporation: *Intel Advanced Vector Extensions 2 and Bit Manipulation New Instructions*, ARCS005, Intel Developer Forum 2012 (2012).

［8］ 安本太一：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，愛知教育大学研究報告自然科学編，五十六輯，pp.13–17 (2007).

［9］ 安本太一：Intel AVXによるCommon Lisp言語処理系の最適化の一考察，愛知教育大学研究報告自然科学編，六十三輯，pp.25–30 (2014).

［10］安本太一：Intel AVX2によるCommon Lisp言語処理系の最適化の一考察，愛知教育大学研究報告自然科学編，六十四輯，pp.15–19 (2015).

［11］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).

［12］大原雄介：ARMの次世代64bitコアCortex-A57/A53はこんなCPUだ，ASCII.jp×デジタル ロードマップでわかる！当世プロセッサ事情第179回，http://ascii.jp/elem/000/000/745/745744/.

（2016年9月23日受理）