愛知教育大学研究報告，55（自然科学編），pp. 9～13， March, 2006

# Common Lisp言語処理系による64ビット環境の評価

安本太一

情報教育講座

# Evaluation of 64 Bit Environments on a Common Lisp System

Taichi YASUMOTO

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

## 1 はじめに

64ビットCPUそのものは10年以上も前に発表されていたが[1, 2, 3]，CPUやメモリなどのハードウェアの価格が下がり，オペレーティングシステムなどのソフトウェアが64ビット環境に対応し，世間一般に普及し始めたのはごく最近である。

一般消費者向けのコンピュータにも，64ビットCPUが搭載されるようになってきており，64ビットCPUを搭載したコンピュータ本体を10万円以下で購入することも可能となった。

このような状況の中で，64ビットという名称に踊らされているだけで，真に対費用効果があるのかを確認せずに，64ビットCPU搭載のコンピュータを導入したり，64ビット環境対応のアプリケーションを開発するケースも少なくないと考えられる。

64ビット環境を導入するにあたっては，64ビット環境の導入が目的に対して十分な性能向上があるのかを評価することが重要である。64ビット環境について一般的な評価はいくつか行なわれているので[4]，本稿では，Lisp言語処理系からみた，64ビット環境の評価を行なう。筆者は既に64ビット環境に対応したCommon Lisp[5]言語処理系を試作しており，この64ビット環境版Common Lisp言語処理系の実行性能を，32ビット環境版と比較する。複数のプラットフォーム（異なるCPUやオペレーティングシステム）の上で実行し，プラットフォームによって，どのような傾向があるのかも比較する。

## 2 関連研究

### 2.1 過去の研究

佐藤らにより，DEC社（Compaq社を経て現在はHewlett-Packard社）の64ビットCPUであるAlphaを搭載したワークステーションを対象にPHL（Portable Hashed Lisp）というLisp言語処理系を移植した報告が行なわれている[6]。しかしながら，この報告は，ごみ集めの報告が中心であり，シェアが小さいAlphaやPHLを対象にしている。

一方，この佐藤らの研究から約15年後，筆者は，UltraSPARCという64ビットCPUを搭載したワークステーション上で，KCL (Kyoto Common Lisp)[8, 9, 10]という従来32ビット環境で動作していたCommon Lisp言語処理系を64ビット環境へ移植した。詳細は文献[7]に譲るが，64ビットCPUの特徴である64ビットアドレス空間や64ビット算術演算機能を生かせるよう，Lispデータオブジェクトの構造や関連するコードの変更を行なった。この際，実行効率やメモリ効率を向上させるために，オリジナルのKCLにおいて行なわれている工夫が損なわれないように配慮した。例えば，64ビット環境においては（32ビット環境の場合とは）基本データ型のデータアライメントが変わるが，異なるデータ型のLispデータオブジェクトでも関連のあるメンバのオフセットは一致させるようにした。

### 2.2 本研究の位置付け

佐藤らの研究も，上記の筆者の研究も，単一のプラットフォーム（一種類のCPUアーキテクチャ）の下で行なわれたものである上，Lispプログラムの実行性能についての言及がほとんどなく，64ビット環境下におけるLisp言語処理系の性能について情報が不足していた。そこで，今回は，UltraSPARCとPowerPCという2つのプラットフォームで，64ビット環境版KCLと32ビット環境版KCLを動作させ，その上でLispプログラムを実行して，64ビット環境になるとLispプログラムの実行性能がどのように変化するかについて性能評価を行なった。

### 2.3 PowerPC版64ビット環境対応KCL

PowerPC版のKCLは，アップルコンピュータ社のPower Mac G5上で動作する。オペレーティングシステムはMac OS Xバージョン10.4（通称Tiger）であ

る。Power Mac G5のCPUは64ビットに対応しているPowerPC970で，Mac OS Xバージョン10.4から64ビット環境が使用できるようになった。これを機に，KCLをMac OS X環境へ移植した。Mac OS Xには，BSD UNIX環境が装備されている。KCLはUNIX環境での動作実績が高いこともあり，Mac OS XへのKCLの移植は，32ビット環境版と64ビット環境版双方とも，比較的に容易に行なうことができた。オリジナルのKCLのコードから変更を要したのは，ファイルディスクリプタ（の標準Cに規定されていない部分），時間計測やアドレス空間の使い方といったオペレーティングシステムに固有な部分だけである。なお，本論文執筆時点では，PowerPC版KCLは一般ユーザ向けのコンパイル機能には対応していない。ただし，KCLの組み込み関数を使って対話的にLispプログラムをコンパイル実行できないだけで，開発者向けの手動手順であればコンパイル実行できる。オリジナルのKCLのコードを，MacOSXにおける実行ファイルのフォーマットや機械語ファイルの動的なリンクに対応させれば，対話的にコンパイル機能を使うことができるようになる。

## 3 評価の環境

評価のためのプラットフォームとして用意したものは，次に示すUltraSPARCIIi（以下単にSPARCという）とPowerPC970（以下単にPowerPCという）の2つである。この2つのプラットフォームは，32ビットバイナリと64ビットバイナリをネイティブに実行可能であるので，32ビット環境版KCLと64ビット環境版KCLをそのまま実行可能である。

- UltraSPARCIIi 333MHz
  メインメモリ：256MB，内部命令キャッシュ：16KB，内部データキャッシュ：16KB，外部キャッシュ：2MB，計算機本体：サンマイクロシステムズ社製Ultra 10，OS：Solaris9 (SunOS5.9)
- PowerPC970(PowerPCG5) 2GHz
  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，計算機本体：アップルコンピュータ社製Power Mac G5，OS：MacOSX 10.4 (Tiger)

SPARCの方は2000年頃発表された製品で，PowerPCの方は2004年に発表された製品である。アーキテクチャが異なるので，単純な比較はできないが，PowerPCの方がキャッシュ機能が充実している。図1は，配送元と配送先の配列を用意し，メモリ転送の速度を測定した結果である。おおよそ，それぞれのCPUのキャッシュのサイズを反映しているようである。

CPUのアーキテクチャもクロックも全く異なるのでメモリ転送速度の絶対値は別としても，メモリサイズの増加に伴うメモリ転送速度の低下の傾向（メモリ転送速度の低下が始まるメモリサイズ，転送速度と最低値と最高値の比など）を見る限り，メモリアクセスの性能は，SPARCは低く，PowerPCが高いようである。

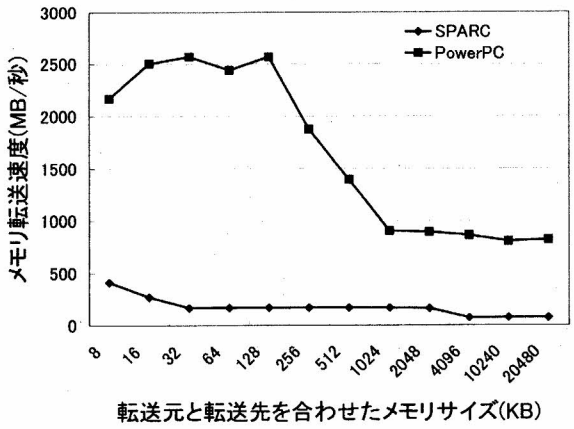


図1 SPARCとPowerPCのメモリ転送速度

## 4 評価用プログラム

評価は，同一のプログラムを32ビット環境版KCLと64ビット環境版KCLの双方で実行し，それぞれの実行時間を比較することによって行う。このための評価用プログラムとして，フィボナッチ数を計算する関数fibと階乗を計算する関数factを用意した（図2参照）。

フィボナッチ数を採用したのは，計算量が多いからである。短時間で実行が終了することがないよう，ここではフィボナッチ数の定義に沿って素朴な定義を

```
;フィボナッチ数の計算
(defun fib (x)
  (cond ((= x 0) 1)
        ((= x 1) 1)
        (t (+ (fib (1- x))
              (fib (- x 2))))))
(fib 30)
;階乗の計算
(defun fact (n)
  (do ((i 1  (1+ i))
       (p 1))
      ((> i n) p)
      (setq p (* p i))))
(fact 10000)
```

図2 評価用Lispプログラム

し，実行時間の有効数字を稼ぐようにしている。計算量は多いが，整数演算は固定長整数（fixnum）の範囲に納まる。

階乗を採用したのは，大きな数を計算させて，無限長整数（bignum）の計算の性能を見るためである。64ビット環境では，無限長整数の計算を64ビット単位で行なうことから，32ビット環境より実行性能が向上するかどうかが関心のあるところであろう。階乗の定義は，(fact 10000) というように大きな引数をとったときにスタックオーバフローにならいように，再帰を使わず，繰返しの構文を使っている。

参考までに，フィボナッチ数を求める関数をC言語で定義し，その実行時間を計測した。その結果を，表1に示す。C言語による関数定義は，Lisp言語で定義したときと同様のアルゴリズムで行ったが，64ビット環境では整数はintではなくlong intを使用した。階乗については，C言語の基本データ型であるintやlong intでは階乗の計算をしてもすぐ桁溢れを起こしてしまうので，C言語による定義と実行時間の計測は行わなかった。

程度の差はあるが，SPARCとPowerPCともに，64ビット環境になると実行性能が悪くなっている（実行時間が増えている）。これは，次のような原因が考えられる。

- 64ビットの即値を扱うことに伴う命令数の増加
  RISCプロセッサにおいては，命令長が固定で，その長さが比較的短い。したがって，即値を扱う場合（整数データやメモリ上の番地をさす場合），一つの命令で即値を扱うことは無理で，複数の命令で分担することがある。あるプロセッサにおいて32ビットの即値をレジスタにロードする時は，レジスタの上位22ビットに即値をロードする命令と（残り10ビットを加算するための）or命令を組み合わせることが行なわれるといった具合である。
  したがって，64ビット環境になっても，32ビット環境のときと命令長が変わらないCPU（SPARCやPowerPC）では，64ビットの即値をレジスタにロードする時は，32ビット環境の場合と比べて命令数が増える。
- キャッシュの効果の半減
  64ビット環境になると，倍精度の整数（long int）を使う場合はもちろんのこと，変数，関数のアドレス，関数呼出しの戻り番地といったアドレスの表現が64ビットと倍増するので，単純に考えてキャッシュの効果が半減する。

## 5　インタプリタ実行における評価

フィボナッチ数および階乗を計算するプログラムを実行し，実行時間を計測した。その結果を表2に示す。

フィボナッチ数の計算は，無限長整数を使わないので，64ビット環境の優位性はない。一方，階乗の計算は，無限長整数を使うので，64ビット環境の優位性があることが予測される。64ビット環境では，64ビットの整数を用いて無限長整数を表現するので，32ビット環境に比べて，四則演算を行なう時の計算量が大幅に少なくて済むからである。

フィボナッチ数においては，SPARCとPowerPCの双方において，32ビット環境に比べて64ビット環境の方が実行時間が長い。しかも，SPARCの方が実行時間の増加の割合が大きい。

階乗においては，PowerPCの場合は，64ビット環境の方が実行時間が短くなる。SPARCの場合は，32

表1：C言語で定義したフィボナッチ数を求める関数の実行時間の比較

| プログラム | UltraSPARCIIi 333MHz | | | PowerPC 2GHz | | |
|---|---|---|---|---|---|---|
| | 32ビット | 64ビット | 比 | 32ビット | 64ビット | 比 |
| C言語fib(30) | 0.3 | 0.3 | 1.0 | 0.004 | 0.004 | 1.0 |
| C言語fib(40) | 40.5 | 41.8 | 1.03 | 14.477 | 17.015 | 1.18 |

（単位は秒）

表2：インタプリタにおける実行時間の比較

| プログラム | UltraSPARCIIi 333MHz | | | PowerPC 2GHz | | |
|---|---|---|---|---|---|---|
| | 32ビット | 64ビット | 比 | 32ビット | 64ビット | 比 |
| (fib 30) | 45.200 | 245.140 | 5.42 | 8.280 | 10.450 | 1.26 |
| (fact 10000) | 39.330 | 112.620 | 2.86 | 5.470 | 4.120 | 0.75 |

（単位は秒）

ビット環境に比べて64ビット環境の方が実行時間が長くなるが，実行時間の増加の割合はフィボナッチ数の場合より小さい。

上記のことから，次のことが考察される。

Lispのプログラムをインタプリタ実行する場合，64ビット環境にすると，一般に実行時間が増加する。C言語で同様なプログラムを実行した場合に比べて，実行時間の増加の割合が大きい。この実行時間の増加については，ポインタが32ビットから64ビットになったことに伴って，キャッシュメモリが実質半分になってしまうことが，大きな原因だと推測される。

Lispのプログラムは，図3のように，Lispのデータ自身で表現されている。コンスセルが“つなぎ”となって，Lispの式を構成している。コンスセルが関数，変数や特殊形式の名前を表す記号，その他のデータを指し示している。関数の本体は，関数の名前を表す記号を表現している記号セルまでたどり，この記号セルの関数定義を格納しているスロットにアクセスすると得られるといった具合である。したがって，ポインタをたどることが多く，プログラムの実行そのものに際して，メモリへの頻繁なアクセスが避けらない。しかも，この頻繁なアクセスはさまざなアドレスに対して行なわれるので，局所性は低い。

その一方で，階乗のような無限長整数を扱うプログラムでは，実行時間の短縮が期待できる場合がある。無限長整数の演算は，64ビット環境では64バイト単位で行なうことができるからである。おおざっぱにいうと，64ビット環境では，32ビット環境の場合に必要な演算回数を n とすると，無限長整数の足し算では足し算の演算回数はn/2になり，無限長整数のかけ算ではかけ算の演算回数は $\sqrt{n}$ となる。無限長整数のかけ算では，ここに挙げた他に，足し算も行なわなければならず，その回数は64ビット環境の方がはるかに少なくて済む。

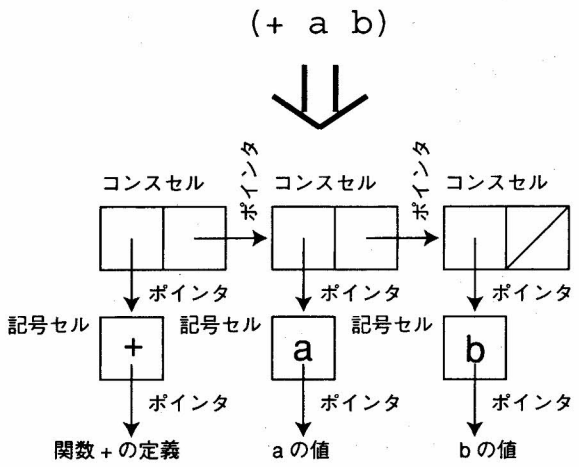


図3：Lispのプログラムの表現

PowerPCの場合は，階乗の計算において，64ビット単位の演算に伴う速度向上が，キャッシュメモリが実質半分になってしまったことにともなう実行効率の低下を補って，実行時間の短縮につながっている。また，SPARCの場合は，階乗の計算において，実行時間の短縮にはつながっていないが，実行時間の増加の割合がフィボナッチ数を求めるときよりは小さくなっている。

## 6 コンパイル実行における評価

KCLのコンパイラは，Lispのプログラムを C言語に翻訳する。例えば，関数factは，図4のように翻訳される。Lispの一つの関数が，C言語の一つの関数に翻訳される。C言語の関数の中では，KCLの内部ルーチンを呼び出して，Lisp言語で記述されたソースプログラムと等価な処理を行うようになっている。number_compareは二つの数の比較，number_timesは二つの数の積，one_plusは1を加算するKCLの内部ルーチンで，固定長整数と無限長整数のどちらも引数にとることができ，インタプリタ実行の時にも用いられている。

インタプリタ実行した時と同じプログラムを，コンパイルして実行した時の実行時間を，表3に示す。フィボナッチ数の場合は，インタプリタ実行の場合に比べ，実行時間が大幅に短縮される。64ビット環境で実行した場合の実行時間の増加の割合も減少している。一方，階乗の場合は，インタプリタ実行の場合と比べて（32ビット環境と64ビット環境の双方の場合において）実行時間にさほど違いがない。実行時間は

```
/*     function definition for FACT     */
static L1()
{     register object *base=vs_base;
      register object *sup=base+VM3;
      vs_check;
      vs_top=sup;
TTL:;
      base[1]= VV[0];
      base[2]= VV[0];
T2:;
      if(!(number_compare(base[1],base[0])>0
      )){goto T3;}
      vs_top=(vs_base=base+2)+1;
      return;
T3:;
      base[2]=
      number_times(base[2],base[1]);
      base[1]= one_plus(base[1]);
      goto T2;
}
```

図4　関数factをコンパイルして得られるコード

表3：コンパイル実行時における実行時間の比較

| プログラム | UltraSPARCIIi 333MHz | | | PowerPC 2GHz | | |
|---|---|---|---|---|---|---|
| | 32ビット | 64ビット | 比 | 32ビット | 64ビット | 比 |
| (fib 30) | 1.930 | 2.210 | 1.15 | 0.3 | 0.32 | 1.07 |
| (fact 10000) | 39.020 | 112.490 | 2.88 | 5.34 | 4.04 | 0.76 |

(単位は秒)

無限長整数のかけ算が支配的であって，インタプリタ実行時にLispのデータで表現された階乗のプログラムを解釈実行するコストは，無限長整数のかけ算のコストと比べれば，比率としてはわずかであったことがわかる。

無限長整数のかけ算が支配的ならば，SPARCにおいても，32ビット環境よりも64ビット環境の方が速くなりそうであるが，実際には速くなっていない。SPARCのメモリアクセス性能の悪さが，足を引っ張っているようである。無限長整数は，整数をポインタで連結して表現されているので，Lispのプログラムをインタプリタ実行で解釈する時と同様に，無限長整数の演算においては頻繁なメモリアクセスが避けられない。

フィボナッチ数列の結果からは，コンパイルによって，プログラムがC言語に翻訳され，プログラムの解釈にともなうコストが 大幅に減ったことがわかる。特に，64ビット環境で実行したときの実行時間の増加の割合が，表2のインタプリタ実行のSPARCのところの5.42から1.15へ大幅に減っているのは，SPARCでは64ビット環境にするとLispプログラムの解釈に必要なコストが非常に高くなってしまうことの裏付けで，SPARCのメモリアクセスの性能の悪さが原因である。

## 7 まとめ

Lisp言語処理系によって，2つのプラットフォームSPARCとPowerPCの64ビット環境の実行性能を，32ビット環境と比較して評価した。

Lisp言語処理系を64ビット環境で動作させるにあたり，プラットフォームの選択に際して最も重要なのは，メモリアクセスの性能であることがわかった。Lispを象徴するコンスデータがポインタ二つから構成されていることや，Lispのプログラム自身がコンスデータがつなぎとなったLispのデータ自身で表現されていることから，メモリアクセスの頻度は，他のプログラミング言語で記述したプログラムの比ではないことは疑いの余地がない。

メモリアクセスが遅い環境下では，大容量のメモリが扱えるといったメリットがあっても，実行性能が大幅に低下してしまう。無限長整数の演算を64ビット単位によって行うことの速度向上も，損なわれてしまう。したがって，64ビット環境でLisp言語処理系を動作させるプラットフォームを選択する時は，メモリアクセスの性能を第一に考えるべきである。また，大容量メモリ利用や無限長整数の演算が必要なければ，無理に64ビット環境を利用する必要もなく，32ビット環境で十分である。

今後の課題としては，本論文で取り上げることができなかったプラットフォーム，例えばAMD Athron64プロセッサやIntel Pentium 4 600番台プロセッサにおいて，64ビット環境下におけるLisp言語処理系の性能を評価することである。多くのプラットフォームで評価を行った経験をもとに，64ビット環境におけるLisp言語処理系の性能向上に取り組む予定である。

## 参考文献


[1]清兼義弘：64ビットUNIX & CDE，共立出版（1997）.

[2]池井満：IA-64プロセッサ基本講座，オーム社（2000）.

[3]McDougall, R. and Mauro, J. : *SOLARIS Internals: Core Kernel Architecture*, Prentice Hall (2000).

[4]64ビットプログラミング，C MAGAZINE, Vol.16, No.10, pp.72-89 (2004).

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

[6]佐藤圭史，青木徹，寺島元章：Alpha-chipマシン上のPHL処理系について，情報処理学会プログラミング研究会研究報告，18-8，pp.57-64 (1989).

[7]安本太一：Common Lisp言語祖理系の64ビット化，愛知教育大学研究報告 自然科学編，第五十三輯，No.10，pp.22-32 (2004).

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

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

[10]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17年9月16日受理)