# Common Lisp 言語処理系における インテル Xeon 5100の評価

# 安本太一

情報教育講座

# **Evaluation of Intel Xeon 5100 on a Common Lisp System**

#### Taichi YASUMOTO

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

## 1 はじめに

筆者らは、Kyoto Common Lisp (以下 KCL という) [123]を対象に、Lisp言語処理系の性能向上や機能向上のため、即値データ[4]の実装や64ビット化(64ビット環境への対応)[56]をしてきた。即値データの実装は1989年、64ビット化は2004年であった。これらの対応をしてから、今日にいたるまでの間に、コンピュータのハードウェアは変化している。例えばメモリ転送速度や64ビット環境の性能は、これらの対応をした当時の前提は、今日のそれとは異なっている。

そこで、最近の一般消費者向けパーソナルコンピュータで使用されているプロセッサとして、IBM PowerPC 970と Intel Xeon 5100 (Woodcrest, Intel Core 2 Duo と同じ Intel Core マイクロアーキテクチャーを採用)を選び、KCLをポーティングし即値データや64ビット環境への対応を行い、その性能評価実験を行った。そして、プロセッサの性質によって、即値データの実装や64ビット化が、Lisp言語処理系の性能にどのような影響を与えるのかを考察した。

#### 2 過去の研究

過去に行われた即値データの実装と64ビット化について,簡単に説明する。

#### 2.1 即値データの概要

Lisp のオブジェクトは,通常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(図1参照)。

即値データの長所は,次のとおりである。

・実質上メモリを消費しないので,ごみ集めの対象と ならない。



図 1 KCL における Lisp のデータオブジェクトの表現 (32 ビット環境の場合)

- ・ヒープ割当が不要なので,オブジェクトの生成が高 速になる。
- ・オブジェクトを得るために、ポインタをたどる必要がなく、メモリアクセスが減る。
- ・コーディングされたそのままの形で等号や大小比較 ができる場合は,高速化が期待できる。

即値データの短所は,次の通りである。

- ・ポインタ内に,データ型チェックのためのタグを用意するので,そのタグのビット数分だけ,数値を表すデータの精度が落ちることがある。
- データ型チェックが複雑になるため,システム全体が遅くなる。
- ・ポインタ内におけるコーディング表現と,加減乗除 などの演算時における表現の間の変換と必要とす る。

#### 22 即値データの性能

サンマイクロシステムズ社の Sun 3( CPU は68020 16MHz, OS は BSD UNIX 系の SUNOS 4 ) や松下電器 産業社の Panasonic BE ( CPU は80386 16MHz, OS は System-V UNIX 系の BE OS ) を用いた性能評価実験で

は,固定長整数(fixnum)の即値データを多用する評 価用プログラムではオリジナルの KCL と比べて実行 時間が約3分の1に減少,即値データを全く使わない 評価用プログラムではオリジナルの KCL と比べて実 行時間が1.09倍から1.16倍になるという結果が得られ た。このときの評価用プログラムは,即値データによ る速度向上やオーバーヘッドを計測することに重点を おいたループのプログラムである。ループ変数に固定 長整数を使い,本体は変数に固定長整数の0を代入す る代入式だけといった簡単なものである。実行時間の 計測は, Lisp のプログラムをコンパイルしてから行 われた。生成されたコードは、ループはC言語のif 文と goto 文によって実現され,ループの制御変数に Lisp のデータ(即値データを実装した場合は即値デー タ,オリジナルの KCL の場合はポインタによって参 照されるオブジェクト)が使われる。

この結果からは,即値データ実装によるオーバー ヘッドはあるが,即値データを使用するときは(この オーバヘッドを補って)速度向上が期待できることが わかった。

#### 23 64ビット化の概要

64ビット化は、利用できるアドレス空間が64ビットになるように、Lisp オブジェクトを指すポインタが64ビットになるようにして行われた。あわせて、即値データも64ビットのポインタ幅を生かすように、拡張された(図2 図3参照)。これによって、大量のメモリを必要とするアプリケーションプログラムを Lisp言語で開発することが可能になった。大量のメモリを必要としない場合でも、KCL とリンクするライブラリが64ビット版しか提供されない場合があれば、KCLの64ビット環境への対応が必要となる。

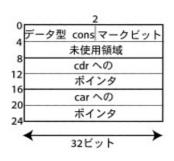


図2 64ビット環境下のコンスセルの構造

さらに,無限長整数(bignum)の演算の速度向上を目的とし,無限長整数の内部表現を,32ビットの整数を連結した表現から,64ビットの整数を連結した表現に拡張した。無限長整数の演算を64ビット単位で行うことにより,32ビット環境のときよりも,無限長整数の演算の計算量が減少するからである。



図3 64ビット環境下のオブジェクトの表現

#### 2.4 64ビット化の性能

サンマイクロシステムズ社の Ultra 10( CPU は Ultra SPARCIIi 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 2 MB ,OS は Solaris 9 ),アップル社の PowerMac G 5 ( CPU は PowerPC 970 2 GHz × 2 ,メインメモリ 2 GB , 1 次命令キャッシュ64KB , 1 次データキャッシュ32KB , 2 次キャッシュ512KB , OS は MacOSX 10 4 Tiger )を用いた性能評価実験では ,フィボナッチ数の計算 (fib 30 )と階乗の計算 (fact 10000 )のプログラムを用いた。フィボナッチ数の計算では , 固定長整数の演算と関数呼出しを多数行う。一方 , 階乗の計算では , 無限長整数の計算が支配的になる。

これらのコンピュータの OS 上では,32ビット環境 用実行形式の KCL と64ビット環境用実行形式の KCL を,OS のモード切替なしに動作させることが可能で ある。フィボナッチ数と階乗の Lisp プログラムは, 実行時間を計測する前に,コンパイルした。

結果は、フィボナッチ数の計算では、64ビット環境は、Ultra 10と PowerMac G5の双方において、32ビット環境の場合より実行時間が10%程度増加し実行効率が低下した。一方、階乗の計算では、64ビット環境は32ビット環境の場合より、Ultara 10においては実行時間が2 88倍となり実行効率が低下し、PowerMac G5では実行時間が0.76倍と実行効率が向上した。

64ビット環境では、命令長が増加する、扱うデータやポインタの大きさが倍増するなどのオーバーヘッドにより、キャッシュの効果が半減するなどの理由で、実行効率が低下されることが予想されたが、フィボナッチ数の結果を見る限りUltra 10と PowerMac G5においてまさにそのとおりになった。一方、階乗では、PowerMac G5は実行効率が向上したが、Ultra 10は実行効率が大幅に低下した。Ultra 10はメモリアクセス速度が遅いため、無限長整数を構成する bignum セルへのアクセスに時間がかかるため、64ビット単位の演算を行っても速度向上が得られない。一方、Ultra 10より約5倍ほどメモリ転送速度が速い Power Mac G5は、64ビット単位の演算による速度向上の効果

が,64ビット環境におけるオーバヘッドを上回った。

#### 3 今回の性能評価実験の環境

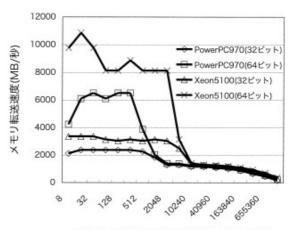
今回の実験に用いたコンピュータは,アップル社製の PowerMac G5と MacPro である。PowerMac G5は,CPUが PowerPC 970 2 5GHz×4,1次キャッシュが命令用64KBデータ用32KB,2次キャッシュが1MB,OSが MacOS X 10 4 Tiger である。MacProは,CPUは Xeon 5100 2 .66GHz×4,1次キャッシュが命令用32KBデータ用32KB,2次キャッシュが4MB,OSは MacOSX 10 4 Tiger である。

つまり、CPU アーキテクチャーが異なる他は、ほぼ同一の条件である。Xeon 5100はインテル社製のCPU であることから、PowerPC 970とあわせると、一般の利用者向けのコンピュータのCPU はほぼ網羅したことになる。KCL のコンパイルおよび、KCL のコンパイラで使用する C 言語コンパイラは、どちらもgcc バージョン4 01である。

これらのコンピュータ上での KCL の性能評価を行うにあたり,メモリ転送速度を計測した。その結果を,図4に示す。32ビットというのは,データの転送を int (32ビットの整数)の代入文の繰返しによって行う32ビット実行形式を用いて計測したものである。64ビットというのは,データの転送を long int (64ビットの整数)の代入文の繰返しによって行う64ビット実行形式を用いて計測したものである。メモリ転送速度を計測する C 言語プログラムを gcc でコンパイルするにあたって用いた最適化オプションは, O である。最適化オプションによってメモリ転送速度の実験結果は変化することから,今後の性能評価実験のことを考え,KCL をコンパイルするときの gcc や KCL のコンパイラが呼び出す gcc の最適化オプションも,この O に揃えた。

図4を見る限り、総じて、Xeon 5100の方がメモリ 転送速度は速い。64ビットの方が、32ビットより、メモリ転送速度は速い。PowerPC 970と Xeon 5100ともに、2次キャッシュのサイズを越えるあたりで、転送速度は落ち込んでいる。Xeon 5100の方が、メモリ転送速度が速いのは、スマート・メモリ・アクセスによるところが大きいのかもしれない。PowerPC 970と Xeon 5100ともに、64ビットにおいて、メモリサイズが少ないところでは伝送速度が安定していないのは、転送速度計測時の誤差が原因であると考えられる。メモリサイズが小さいところではメモリ転送時間が非常に小さくなっているため、メモリ転送速度を算出するときの分母が小さくなっている。

試しに,データの転送を long int (64ビットの整数) の代入文の繰返しによって行う"32ビット"実行形式を用いてメモリ転送速度計測した場合は,PowerPCとXeonともに,データの転送を int (32ビットの整数)



転送元と転送先を合わせたメモリサイズ(KB)

図 4 PowerPC 970と Xeon 5100のメモリ転送速度

の代入文の繰返しによって行う32ビット実行形式を用いた場合と同様のメモリ転送速度を示した。

#### 4 KCLの Xeon 5100への対応

PowerPC 970への対応は過去の研究[5]で既に行われているが, Xeon 5100で KCL を動作させるためには KCL のソースコードを変更する必要があった。

Xeon 5100はリトルエンディアンであるため,エンディアンに依存する部分は,ビックエンディアン PowerPC 970用の KCL から変更する必要があった。64ビット環境への対応,すなわちポインタの拡張,アラインメントの変更などは,PowerPC 970における経験を生かすことができた。Xeon 5100固有の事情で特に変更を要したのは,ヒープの管理に関する部分であった。MacOSX の Xeon 5100の64ビット環境下では,ヒープがおおよそ100000000番地(16進数表記)からと高位

図5 評価用プログラム

から始まる。オリジナルの KCL はヒープが 0 番地近くから始まることを仮定していたので,メモリ管理関係のコードを修正する必要があった。

PowerPC 970の場合とオペレーティングシステムが 同じであるので, Xeon 5100になってもシステムコー ルやライブラリを使用している部分の修正は必要な かった。

#### 5 性能評価実験

同じ Lisp プログラムを, PowerPC 970上と Xeon 5100上の KCL 上で実行し, その実行時間を計測した。 PowerPC 970と Xeon 5100のそれぞれには, さらに, 即値データあり32ビット版,即値データなし32ビット版,即値データあり64ビット版を用意した。したがって,全部で6つの KCL の実行ファイルを用意した。即値データなしの64ビット版は,用意しなかった。 2 バイト表現の文字データを処理系にあらかじめ全て登録するかどうか(KCLのソースプログラムでいう character\_table をどうするか), 絶対値が比較的小さい整数をどの範囲まであらかじめ登録しておくか(KCLのソースプログラムでいう small\_fixnum\_table をどうするか)といったことを,検討する必要があったからである。

評価用プログラムは 図 5 に示すように ,フィボナッチ数を求めるプログラムと階乗を求めるプログラムである。フィボナッチ数を求めるプログラムは , 固定長整数 (fixnum)を多く使い , 関数呼出しが多く行われる。無限長整数は使用しない。階乗を求めるプログラムは , 無限長整数を多く使うプログラムである。これらの Lisp プログラムは , 実行を行う前に , コンパイルを行った。

PowerPC 970における実行時間を表 1 に ,Xeon 5100 における実行時間を表 2 に示す。

表 1 PowerPC 970 2 5GHz における実行時間の比較

プログラム	即値データ なし	即値データあり			
	32 ビット	32 ピット	64 ピット	比	
(fib 30)	0.303	0.253	0.317	1.25	
(fact 5000)	1.023	0.997	0.693	0.70	

(単位は秒)

表 2 Xeon 5100 2 .66GHz における実行時間の比較

プログラム	即値データなし	即値データ あり			
	32 ピット	32 ピット	64 ピット	比	
(fib 30)	0.150	0.186	0.167	0.89	
(fact 5000)	0.323	0.323	1.017	3.15	

(単位は秒)

#### 6 実験結果の考察

PowerPC 970の結果をみると,フィボナッチ数の計算では,32ビット版においては,即値データを使う場合は,実行時間が0.83倍ほどになっており,実行効率が向上している。以前の即値データの研究[4]の実験より,実行時間短縮の割合が少ないが,今回の実験では関数呼出しなどの時間が含まれていて相対的に即値データが寄与する割合が減っていることを考慮する必要がある。それでも,実行時間の短縮に結び付いているので,即値データの効果は実用的であると考えられる。

即値データを使う場合において,64ビット版は,32ビット版に比べて実行効率が低下しているのは,関数呼出しなどプログラムの制御に関わるコストが64ビット環境になって増えていることが影響しているものと思われる。

32ビット版の階乗の計算では,即値データありの場合は即値データなしの場合と比べて実行時間が0 97倍と実行時間が若干減少しているが,フィボナッチ数の場合ほど実行時間に減少がさほどみられず,ほぼ同じである。これは,無限長整数の掛け算が支配的だからであろう。

64ビット版の階乗の計算では,整数演算を64ビット単位で行うことの速度向上が,プログラムの制御に関わるコスト上昇(速度低下)を上回って,大幅に実行時間が短縮されてる。今回の実験で用いた PowerPC970は,過去の研究[6]のときとは CPU のクロックやキャッシュの容量が異なっているが,過去の実験結果と傾向は変わりない。

一方, Xeon 5100の場合は, PowerPC 970の場合とは様相が異なっている。

- ・32ビット版のフィボナッチ数の計算では,即値データありの方が,即値データなしの方より,実行時間が124倍と長くなっている(実行効率が低下している)
- ・フィボナッチ数の計算では、即値データありの場合は、無限長整数の恩恵がないのにもかかわらず、64 ビット版は32ビット版と比べて実行時間が0.89倍と 短くなっている(実行効率が向上している)。
- ・階乗の計算では,64ビット版は,32ビット版より, 実行時間が3.15倍と大幅に増加している(実行効率 が低下している)。これは,PowerPC 970の場合よ り,実行時間を要している。
- ・32ビット版の場合は、階乗の計算は、即値データありと即値データなしでは違いはない。これは、階乗の計算が、即値データの関与は少なく、無限長整数の計算が支配的であるからであろう。事実、PowerPC 970と Xeon 5100の場合双方とも、階乗のプログラムをインタプリタ実行しても、実行時間はあまりか

わらない。コンパイルして機械語に翻訳されるのは プログラムの制御であり、最終的にはインタプリタ 実行とコンパイル実行の双方とも KCL 内部の無限 長整数乗算ルーチンを呼び出しているからである。

結果を一部説明できないところもあるが,今回の結果をみるかぎり,Xeonの状況については次のように考えられる。

- ・メモリ転送速度が十分に早くなっているので,メモリアクセスを抑えることができるという即値データの長所が,生かせない。
- ・Xeon は,32ビットのプログラムを特に効率良く実行できるように設計されている。例えば,2つの命令を1つに融合して処理するマイクロフュージョンの機能は,EM64T Long Mode では機能しない。Xeon 5100において,64ビットのプログラムの実行性能が著しく悪いというわけではなく,32ビットのプログラムが効率良く実行できたときの実行性能が際立っているということであろう。したがって,64ビット版では,32ビット版に比べて階乗の計算の実行時間が著しく増大してみえるのは,実行時の最適化機能が利用できないことが原因だと考えられる。

試しに, PowerPC 970と Xeon 5100において,32 ビット版の即値データなしで,インタプリタ実行でフィボナッチ数 (fib 30)の計算を行ったところ,それぞれ9.184秒,6 297秒を要した。Xeon 5100は PowerPC 970に対して,コンパイル実行のときは約50%の実行時間であったのに,インタプリタ実行の時は約69%の実行時間となって優位性が低下していた。インタプリタ実行のときは,Lispプログラムを構成するセルをたどることになるので,スマート・メモリ・アクセスといった実行時最適化機能が効きにくいことが推測される。

#### 7 ま と め

過去に研究が行われた即値データの実装と64ビット

化を, Xeon 5100プロセッサに適用してみた。

今回の実験結果の範囲では,過去の実験結果とは異なり,即値データによる実行効率の向上や,64ビット演算による無限長整数演算の高速化が得られなかった。原因は,メモリ転送速度の著しい向上や,32ビット環境に焦点をあてた最適化が挙げられる。

過去に効果が認められた手法であっても,前提としていた CPU のアーキテクチャの状況が変わると,期待した効果が得られないことある実例を示した。

本論文で行った実験は、フィボナッチ数と階乗の計算だけなので、Xeon 5100プロセッサの Lisp 言語処理系における性能を述べるには十分ではない。最近のプロセッサは得意とする実行時最適化が適用できる場合は劇的な実行効率向上をもたらすが、さもなければ期待したほどの性能が得られない場合があるので、今後さらに他の実験を重ねてデータを得て、プロセッサの特徴にあわせて Lisp 言語処理系をチューニングしていく手法を探っていくことが今後の課題である。

## 参 考 文 献

- [ 1 ] Steele, G. L. Jr.: Common Lisp the language, Digital Press (1984).
- [ 2 ] Yuasa, T. and Hagiya, M.: Kyoto Common Lisp Report, Teikoku Insatsu Publishing (1985).
- [ 3 ] Yuasa, T.: Design and Implementation of Kyoto Common Lisp, *Journal of Information Processing*, Vol. 13, No. 3, pp. 284–295 (1990).
- [4] 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
- [5] 安本太一: Common Lisp 言語処理系の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, pp 22 32 (2004).
- [6] 安本太一: Common Lisp 言語処理系による64ビット環境 の評価, 愛知教育大学研究報告自然科学編,第五十五輯, pp 9 13 (2006).

(平成19年9月18日受理)