プログラミング 18－8
(1998. 3. 23)

Alpha-chipマシン上のPHL処理系について

佐藤 圭史　　青木 徹　　寺島 元章

電気通信大学大学院　情報システム学研究科

東京都調布市調布ヶ丘 1-5-1

{sato,aoki-to,terashim}@tera.is.uec.ac.jp

あらまし　PHL(Portable Hashed Lisp)は32ビット計算機を対象に設計されたLisp処理系である。本論では64ビットアーキテクチャであるAlpha-chip上のPHLのデータ表現とその実装法を述べ、新たに作成した圧縮型ガーベッジコレクションについての評価も述べる。ガーベッジコレクションは世代別管理方式を応用し、不使用セル全体の回収を行わず、最近に使用されたセルの局所性を高めることを実現する。このガーベッジコレクションによってPHL処理系の処理時間の短縮が実現されている。

キーワード　Lisp,64ビットアーキテクチャ,データ表現,ガーベッジコレクション

A new PHL system implemented on Alpha-chip

Keishi Sato　Tohru Aoki　Motoaki Terashima

Graduate School of Information Systems
University of Electro-Communications

1-5-1 Chofugaoka,Chofu-Shi,Tokyo 182 Japan

{sato,aoki-to,terashim}@tera.is.uec.ac.jp

Abstract　PHL (Portable Hashed Lisp) is a dialect of LISP and its system is originally designed for 32-bits machine implementations. In this paper, we present the design and implementation of new PHL on an Alpha-chip machine with 64 bits architecture, and the analysis of its compactifying garbage collection using a new scheme. The garbage collection is based on "generation" and focuses its scavenge on recently used cells rather than all used cells of a heap so that "young" cells being in use are highly localized in the heap. The new PHL with such garbage collection realizes the economy of the total execution time.

key words　Lisp,64bit-architecture,Data representation,Garbage Collection

1 はじめに

当研究室ではPHL(Portable Hashed Lisp)[1] と名付けられたLisp処理系が様々なハードウェア上(Sun Sparc、SONY NEWS、intel x86など) で、また様々なオペレーティングシステム上(SunOS、FreeBSD、Windows95/NTなど) で動作している。PHLはLispの一方言で、Standard Lispと、Common Lispの諸機能の融合を目指したLisp処理系であり、PHL処理系は使用計算機が自由に選択できるという可搬性実現のためにC言語で記述されている。しかし、PHLは32ビットアーキテクチャ計算機を対象に設計されていて、Cのソースコードにもそれが反映されている。

一方、今回PHLの移植を試みたAlpha-chipは64ビットプロセッサであり、その上で動作するDigital UNIXは64ビットアドレスを完全にサポートするオペレーティングシステムである。Cのコンパイル時にオプションを指定すると、ソースコードを変更することなく32ビットモードでPHLを実行することも可能である。しかし、ソースコードを変更せずにそのまま64ビットアプリケーションとしてコンパイルして動作させることはできない。これはPHLのデータ表現で用いるタグがポインタ値と競合するためである。

64ビットプロセッサはこれまでに比べて非常に大きな記憶空間を利用でき、実際に大容量のメモリを搭載した計算機が多い。こういう状況においてLispプログラムを実行させる場合、Lisp処理系が管理する記憶空間は十分大きく確保できるので、GC(ガーベッジコレクション) を行うことなくLispプログラムを実行することも十分可能である。これに伴って、今までのGCに対する考え方を変え、処理系の高速化に寄与するGCを提案する。

本論では、こうしたAlpha-chip上で動作するPHLのデータ表現とGCの実装法、及び評価について述べる。

2 64ビットプロセッサへの移植

2.1 アドレスの扱い

C言語においてアドレスが64ビットとなることにより最も影響を受けることは、ポインタである。C言語では、文字列、配列、関数アドレスなど様々な用途でポインタが利用される。

C言語の仕様ではポインタはlong int型と相互に代入が可能であるが、一般的な32ビットのC言語仕様ではintとlong intが同じであり、またこれらを明確に区別する習慣があまりなかったため、ポインタとint型の相互代入が日常的に行われてきた。これをそのままにして64ビットに移行すると、int型が32ビットであるための情報の欠落が生じ、正常に動作しなくなる[4]。

旧(32ビット版)PHLにおいてもこのことは例外でなく、ポインタとint型との間の相互代入を、ポインタとlong int型との間の相互代入とし、情報の欠落を防いだ。

2.2 型の変更

Alpha-chipは32ビット長(int型) を扱う命令と64ビット長(long int型) を扱う命令は別のものである。レジスタと同じサイズのデータを扱ったほうが高速であると考えられるので、Alpha-chip上へPHL処理系を移行するにあたり、処理系内部でint型として扱っていた変数は全てlong int型へと変更した。

int型を残したPHL処理系と、int型をlong int型に変更したPHL処理系では、実行速度に数パーセントの差が生じており、後者の方が速い。

2.3 データ表現の変更

2.3.1 旧PHLのデータ表現

旧PHLのデータは1語(32ビット) で表現されている。1語はタグ部とアドレス部に分けられ、いわゆるポインタタグと呼ばれるものとなっている。それぞれのポインタは、そのポインタが指す型を識別する情報をタグ部にもつ。タグ部は32ビット中の上位2 ～ 3ビットで構成されており、その内容は、000(CONSデータ)、001(シンボル)、010(ブロック)、011(浮動小数点数)、10(短整数)、11(文字) である(図1 (a))。ここで、短整数と文字のタグ部が2ビットとなっているが、これは短整数を$\pm 10^8$の範囲で、文字を4字詰めで実現するためである。

ブロックデータではベクトルや配列、文字列、長

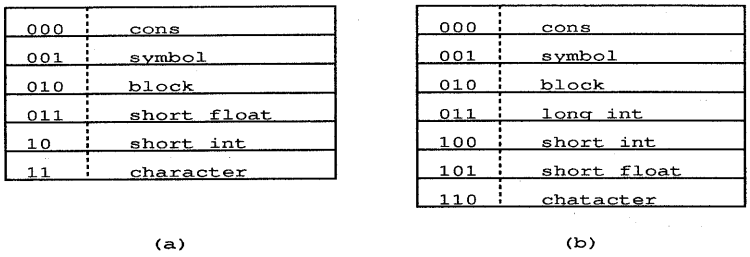

図 1: 新旧 PHL のタグ表現

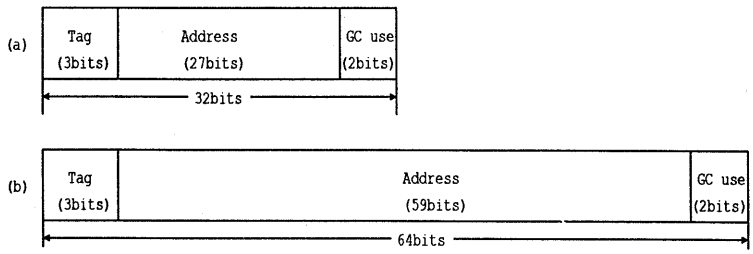

図 2: データ表現の変更

整数といった可変長セルを扱っている。ブロックデータでは、その先頭の 1 語 (ヘッダ部) がその属性 (図 3 Type 部)、その成分の個数や長さ (図 3 Length 部) を管理している。

シンボルのデータは図 4(a) に示す構造の中に保持されている。ここで、Value は変数の値などを保持するために使われ、 Function には関数定義が、Propaty List には属性リストがそれぞれ保持されている。これに続くアドレスには印字名が 7 ビットコードで 1 語に 4 文字ずつ保持されている。

2.3.2 新 PHL のデータ表現

従来と同様に 29 ～ 32 ビット目にタグを付けようとした場合、Alpha-chip ではアドレス表現が 64 ビットであるため、ポインタ値とタグとの間に競合が生じる。これを解消するため、以下に示す変更を行った。

タグとアドレス値の競合を避けるため、タグを 62 ～ 64 ビット目 (最上位) に配置した。これに加えて、新 PHL では全てのタグを 3 ビットで表現することとした。識別できるタグが増えたことを利用して、新たに長整数を識別するタグを設けた (図 1)。処理系内部で行われる数値、整数、長整数の型判別では、長整数がブロックデータとして扱われているために、ブロックデータを参照する手間を要していた。長整数のタグは、この手間を軽減させることが目的である。

以下、個別的な変更について述べる。

短整数

タグの変更で、即値データ表現に使用できるビット数が旧 PHL の 29 ビットから 61 ビットとなった。短整数についてもこれは同様であり、$\pm 2^{58} - 1$ までの整数が短整数として扱えるようになった (下位 2 ビットは GC の制約により使えない)。

ブロック

ブロックデータは処理系内部で一つのセルとして扱われている。このためヘッダ部分は 32 ビット長のままで扱えなくなったので 64 ビット長で表現するように変更した (図 3)。

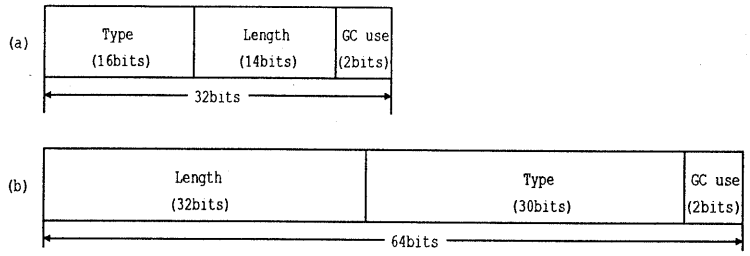

図 3: ブロックヘッダの変更

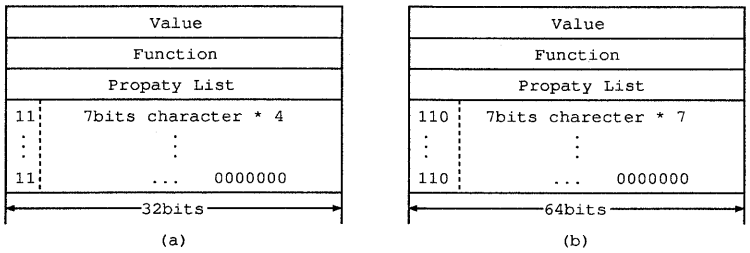

図 4: シンボルの変更

文字

文字は、即値データの 61 ビット表現に伴い、7 ビットコードで 8 字詰めとなるように変更した。これは、シンボルの印字名やブロックデータ表現による文字列に適用される。これによって、例えばシンボルの表現は、図 4(b) のようになる。

2.4 hash 表

PHL の処理系内部ではシンボルの一義性を実現するために hash 法を用いている。そのため PHL 実行時には hash 表が作成されるが、その成分は旧 PHL ではシンボルの heap アドレス (29 ビット) であった。新 PHL ではこれを相対アドレス (32 ビット) に変更した。このため hash 表は 32 ビット長になり、記憶空間の節約と局所性の向上が図られている。

これによって heap サイズは $2^{32}-1$ バイト以下という制約が付くが、実用的には十分な大きさといえる。

3 ガーベッジコレクション

3.1 設計方針

新 PHL の GC は、限られた記憶空間を効率的に使うことを目的とした GC という考え方から離れ、heap 内の使用中セルの局所性を高めることに機能を絞った。短期間で見ればキャッシング効果、長期間ではスワップといった、ハードウェア依存、OS の管理など、計算機の低レベルな部分に依存した高速化を行うことを目的としたものである。

このため GC は処理系に必要不可欠なものとしてではなく、どちらかと言えば付加価値的な位置付けを持ったものとなっている。

3.2 実装

GCは、基本的に圧縮型GCを採用している。このGCは、印付けされた(使用中)セルの絶対量に比例する時間で、処理を完了することができる。また、これまでの圧縮型GCとは違い、その対象領域はheapの高アドレス側(最近に確保された部分)にに限定される。それ以外の部分については使用済みセルの回収を行わない。これは以下のことを前提にしているからである。

- 関数定義で使用されるセルがheapの下位アドレス側に局所性を保って置かれることが多い。
- GC対象領域(高アドレス側)では、比較的多くのセルが計算で使用されている。
- 関数定義で使用されるheapの最下位部を除き、GC対象領域外に存在するセルは殆どが使用済みになると考えられる。

GC対象領域の設定は処理系起動時に行われる。なお、設定したGC対象領域の微調整、圧縮時のポインタ補正などには圧縮型の世代別管理方式GCアルゴリズム[2, 3]を利用している。

3.2.1 リスト処理系側の管理

ここでは、リスト処理系側が行うポインタの管理について述べる。世代別管理方式では、ポインタ管理は新旧世代を跨る逆ポインタについての管理、及び死蔵セルの回収を目的としたものであった。本論ではこれをGC対象領域と非対象領域に跨るポインタの管理、及びGC対象領域の微調整に用いている。

処理系起動時

処理系起動時にはGCが起動されてから次にGCが起動されるまでに使用できるセル量(N)が設定される。処理系起動直後、この値はheap開始アドレスからtopまでの値として使用される。また、これと同時にpta(pseudo-target address)の値が設定される。ptaはGC対象領域の下端、topは上端であり、ptaはNに対する割合として設定される。

ポインタ書換え操作時

setf、rplace関数でポインタが変更されるとき、それ自身(ptaより下位のアドレス)とptaとの間を指すポインタが存在しないようなheapのアドレスをチェックしnta(nominee of target address)に保持する。また、ptaとtopの間を指すような逆ポインタがptaより下位のアドレスで発生した場合、これを表(Remember set)へと登録する。

3.2.2 ガーベッジコレクションの動作

以下は、GCが呼ばれてから終了するまでの動作である。図5は動作時のheapの様子を示したものである。

GC起動

GCが起動されるのはあらかじめ設定されているtopまでheapが使用されたときである。

印付け

印付けは、スタック、シンボル、Remember setを根として行われるが、ここで再度ntaの補正が行われる。ここでの補正はリスト処理系側が行ってきた管理に加え、ptaとtopとの間の使用セルにntaとptaとの間を指すポインタが存在しなくなるように補正が行われる(図5(a))。この補正によってptaとntaとの間に存在するのは、使用済みセルだけである。

GC対象領域の補正

印付け終了後、ptaの値が補正される。ptaは予め固定されているので、ptaを跨るセルの存在については一切考慮されていない。そこで、ntaをptaの値とすることによりGC対象領域の微調整を行う(図5(b))。

圧縮

このようにして設定されたGC対象領域に対して圧縮を行う。また、これと同時にポインタの補正を行う。(図5(c))。

使用領域の割当て

GC終了後、Nの値から新しくtop、ptaが設定される。また、新しく生成されるセルはGC終了後の下端にあるセル以降に格納されて行く(図5(d))。

なお、本GCは圧縮型GCに世代別管理を融合した高速なアルゴリズムを使っているのに加え、GC

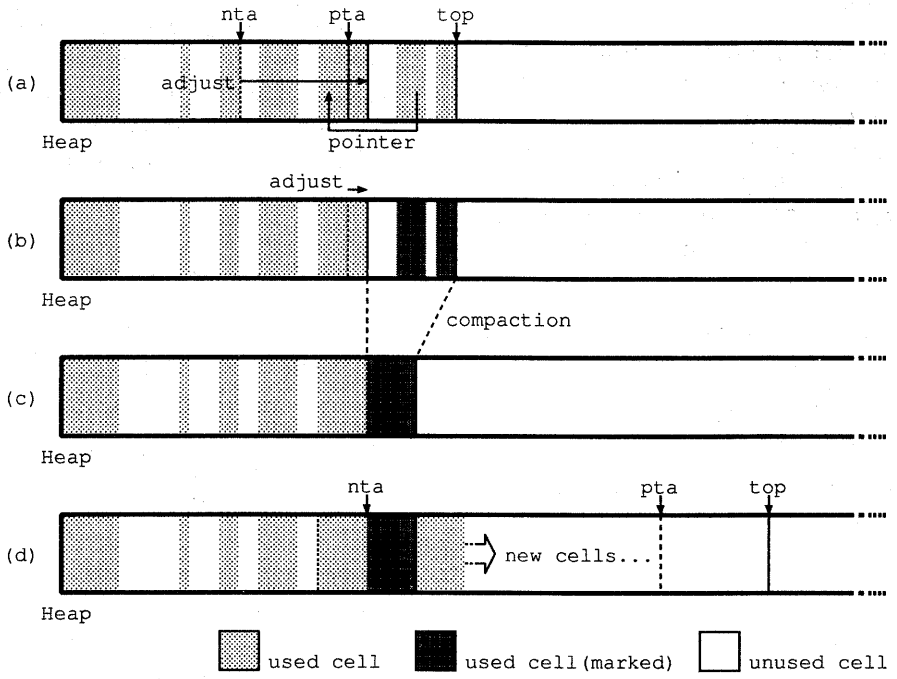

図 5: ガーベッジコレクションの動作

対象領域が小さく抑えらているので、これまでのGC の数倍の速度で処理が可能である。

4 評価

4.1 データ表現の変更

int 型の変更、データ表現の変更、及び拡張、hash 表の変更による効果を見るため、処理系の純計算時間の比較を行った。比較の対象として用いたのは旧PHL であり、alpha-chip マシン上で 32 ビットプログラム用コンパイルオプションを指定してコンパイルしたものである。テストプログラムとして用いたのは TPU、Boyer である。また、実行に当ってはGC が起動しないよう十分大きな heap を確保した。この結果を表 1 に示す。TPU では約 5.1% 、Boyer では約 7.7% の処理速度向上が見られた。

	TPU	Boyer
32bit	1713(ms)	2561
64bit	1626	2365

表 1: データ表現変更の評価

4.2 ガーベッジコレクション

4.2.1 処理時間

本研究では GC を処理系全体の高速化を目的として実装した。この効果を見るために、GC が呼び出される場合と、GC が全く起動しない場合とで、処理系全体の処理速度を比較した。Reduce をテストした結果を、表 2、3 に示す。表 2 は、GC 対象領域の大きさを変化させたものであり、表 3 は格納領域の大きさを変化させたものである。この結果から、GC を呼ばない場合よりも、格納領域、GC 対象領域をうまく設定して GC を起動させた場合の方が

|gc| (KF)	50	60	70	80	90
time(ms)	3367	3256	3269	3273	3304

表 2: GC 対象領域に対する処理速度変化 (Δtop=100K)

Δ top (KF)	80	100	150	200	300	700
time(ms)	3314	3256	3268	3309	3269	3270
GC 回数	8	6	4	3	2	0

表 3: GC の回数、格納領域に対する処理速度変化 (|gc| =40KF)

高速に処理可能であることが示される。この結果はGC処理時間を含んでいるので、処理系の純計算時間では更に違いが出て来るものと思われる。

4.2.2 評価

新PHLのGCを実装するにあたって前提となっていた事項、及びGCの妥当性を検討するため、heapを小区画に分割し、その小区画に含まれる使用セルの割合をGC起動時毎に調べたものを図6に示した。

TPUの結果からはセルの寿命が短く、GCの対象領域に入っている使用セルもその殆どが次回のGCまでに使用済みセルとなっていることが分かる。使用セルの局所性が十分に保たれており、この結果は本GCが有効に働く例となった (図6(a))。

Reduceの結果からは長期間使用されるセルが存在するため、GCの対象領域外に、不使用セルとなることを期待していたセルが生き残っていることが読み取れる (図6(b))。これに対し、GC対象領域を2倍にするとある程度使用セルの局所性が改善され、また純計算の処理時間も短縮されていると考えられる (図6(c))。なお、実際の処理時間は図6(b)で3287(ms)、図6(c)で3282(ms)であった。

5 おわりに

本論ではPHLのalpha-chip上での実装、使用セルの局所性を高めることを目的としたGCの実装およびこれらの評価を行った。

今後の改良点として挙げられるのは、現在よりも多くのビットをタグに割り当てられることが分かったので、これまでブロックデータとして扱っていたデータを全てタグ表現で表し、高速化を図ることである。また現在のGCにおいては世代別管理型GCをそのまま流用しているため、ntaの管理機構に処理系が大きな手間を要している。ここに新たなntaの管理機構を適用することによって処理系全体の速度向上を図ることも可能と思われる。またheapの状態から、処理系がpta、ntaの値を適切に管理することによって高速化を図ることも可能と思われる。

今回提案したGCは処理系の計算を高速化させる効果を持っている。しかし、本GCは停止回収型であるため、GCの起動に関わらず、処理系全体の速度はそれほど変わらない。今後、GCの並列化を行えば処理系の全体の高速化が実現できるものと思われる。

参考文献

[1] 寺島元章:PHLの新インタプリタ, 記号処理研究会資料,SYM 73-5,pp.33-40(1994).

[2] 寺島元章:古典的ガーベッジコレクションからの話題, 記号処理研究会資料,SYM 68-5,pp.31-38 (1993).

[3] 新田寛:世代型ガーベッジコレクションの研究, 修士論文, 電気通信大学 情報システム学研究科 (1997).

[4] 清兼義弘:64 ビット UNIX&CDE, 共立出版 (1997).

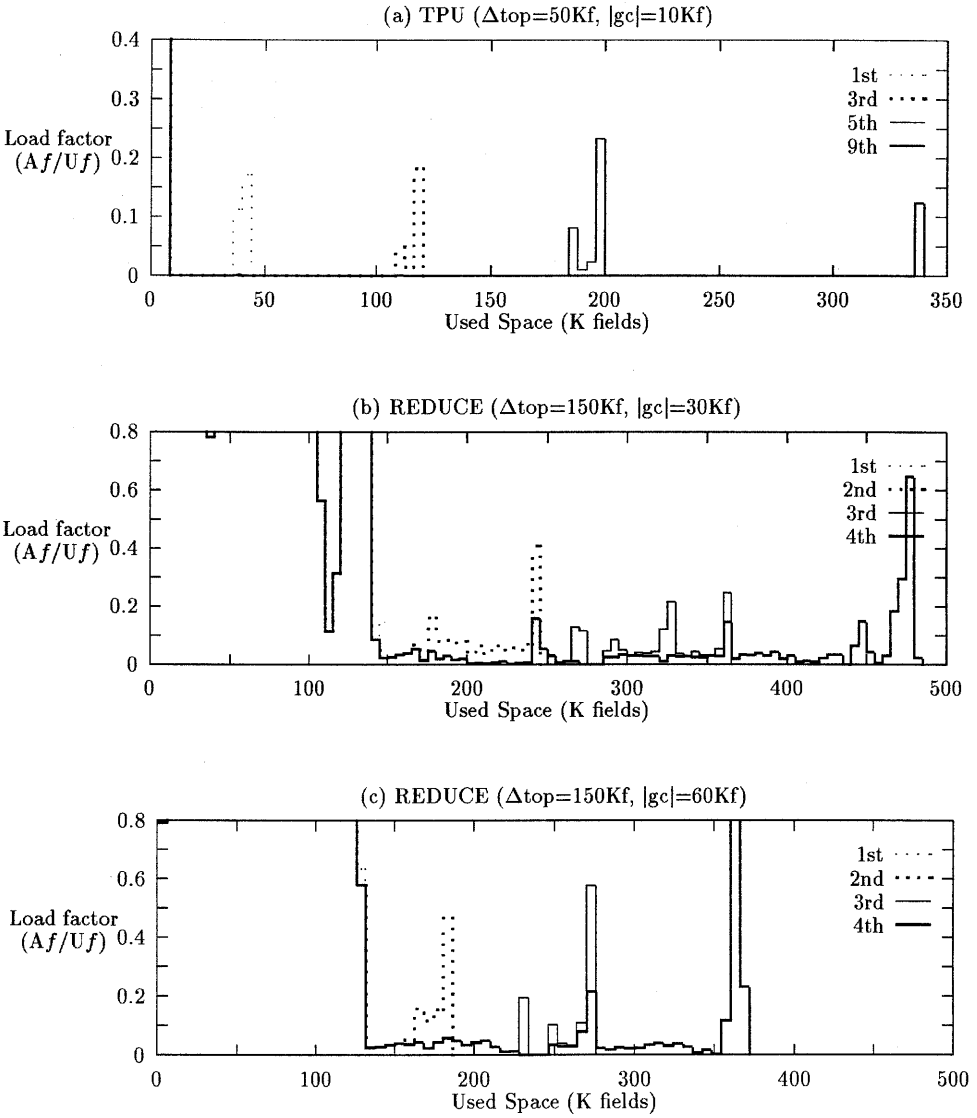

図 6: heap 内の使用セル