

使用中セルのちらばりに注目した実時間塵集めの提案

中澤 雅博、田内 康之、斉藤 宗昭
セコム IS 研究所 人工知能研究室

使用中のセルのちらばりに注目した実時間塵集め提案を行なう。アプリケーションを実行してセルの使用状態を見ると、使用中セルは一箇所に集中するよりも散らばる傾向があり、また塵がはるかに多いことがわかった。この性質を利用して塵集めで大きな比重を占める回収処理を無くし、コンス等がセルを必要とするときにセル領域を走査して塵を見つける方式を提案する。このため、常にセルに印を残すことで使用中セルと塵を区別する。

A proposal for a real time garbage collection system which considers the distribution of used cells

Masahiro Nakazawa, Yasuyuki Tauchi, Muneaki Saito

Artificial Intelligence Department,

SECOM Intelligent Systems Laboratory

This paper proposes a real time garbage collection system(GC) which considers the distribution of used cells.

Used cells which result from executing our application are widely distributed, not concentrated in the cell area. Using this property, this GC does NOT have a sweep phase which is an important part of the conventional GC. Every time CONS is executed, garbage is taken from the cell area and used. In order to achieve this, the MARK part of each cell is maintained throughout the execution and distinguished to be either a used cell or garbage.

§1. はじめに

近年、人工知能システム（AI）の研究の進歩にともない、AIを色々な分野に導入することが盛んに行なわれている。特に実時間処理の分野では、高い応答性が要求されることから、リスブ等のAI言語を利用する場合「塵集め」による処理の中断が大きな障害となっていた。そこで、我々は実メモリシステムを前提としたより中断時間の短い実時間塵集めを提案する。

§2. 基本アイデア

以下の章では、提案する塵集めアルゴリズムを説明するために、リスト処理で使用するデータ構造であるセルのみについて考えることにする。また、塵とは「未使用かあるいは現在使用されていず再利用可能なセル」と定義する。

塵集めは、使用中セルと塵を区別するための印付けフェーズと再利用可能な塵を回収する（世代別型の塵集めの場合は使用中セルの再配置に対応）¹回収フェーズとの二つに大きく分けることができる。塵集めによる処理の中断時間をできるだけ分散させることを考え、実行中に塵集めを少しずつ行なう実時間塵集めとして「印付け／回収フェーズをそれぞれ分割してコンス実行時に少しずつGCを進める」方法[1, 2]が報告されている。この方法をベースにしてもっと塵集めにかかる時間を分散させるにはどうしたらよいかを考える。

そこで、印付けフェーズと回収フェーズそれぞれの特徴を調べて可能性を探る。印付けフェーズによる処理の中断時間を短くするためにコンス実行時に少しずつ印付けを行なうことを考えてみる。セルを1個使うごとに印付けを実行するとした場合、

$$\left\lceil \frac{\text{（塵集め開始時に使用中セルの個数} + \text{残りのセル数）}}{\text{残りのセル数}} \right\rceil$$

個ずつ印付けを行なうこととなる。したがって、いつ印付けフェーズを始めるかで何個ずつ印付けをするかが決まる。印付けの個数を少なくしようとすれば塵集めの開始が早くなる。また、塵集めの開始を遅らせてセルを有効利用しようとすれば1回当たりの印付けの個数が増えるので1回当たりの処理の中断時間が長くなる。

回収フェーズはセル領域を端から端まで一ずつ走査して塵を回収していくため、セル領域に比例して回収にかかる時間が長くなる。印付けよりも非常に大きな領域を扱うことになるので1回当たりにかかる処理の中断も大きい。そこで、回収にかかる時間をどうにか塵集め期間中のみならず一個一個のセルの取り出し時に分散できないであろうか。これが、提案する塵集めのキーポイントになる。

アプリケーションを実行して得た感触であるが、使用中セルは全体の個数に較べてそれほど多くない。また、使用中のセルは、メモリ上極端に連続していない。この使用中セルの散らばりに注目すると、塵の1個ずつの取り出しではそれほどセル領域を走査する時間がかからないはずである。いつもセル1個ずつに使用中かどうかを示す印があれば、塵の1個ずつの取り出しが可能になる。すなわち、欲しい時にセル領域を順々に調べてとり出せばよいことになる。この考え（印を残す）を採用すると明示的な回収フェーズは消滅して一個一個の再利用可能なセルの取り出し時間に埋没する。これにより、塵集めによる処理の中断が印付けフェーズのみとなり一度に起こる中断時間を小さくすることができる。また、印付けフェーズを実行時に少しずつ行なうことで中断時間の分散がはかれる。

§3. 塵集め

3.1 アルゴリズム

実行中に少しずつ塵集めを行なう場合は、セルの状態は「使用中 (B)」、使用中か塵か「不明 (G)」、および「塵 (W)」の三種類になる。本方式のようにセルを必要ときにセル領域を走査して塵を取り出して再利用しようとする三種類の状態を区別できる必要がある。この三種類のセルの状態が塵集め (印付け) の開始から終了までにどう変化するかを図1に示す。印付け開始時は「使用中」の印を付け直すために「不明」と「塵」の二種類が存在する。印付け中は「塵」と「不明」と「使用中」が存在する。そして、印付け終了時には塵と使用中が存在する。使用中セルは、二通りのでき方がある。一つは「不明」状態のセルが使用中であるために印付けされたものである。もう一つは印付け期間中に塵の中から再利用され使用中として印付けされたセルである。印付けが終了すると「不明」のままの印を持つセルは使用されていないセルと見なせるので、すべて塵と見なすことができる。B、WおよびGの三種

W：塵 G：使用中か塵であるか不明 B：使用中

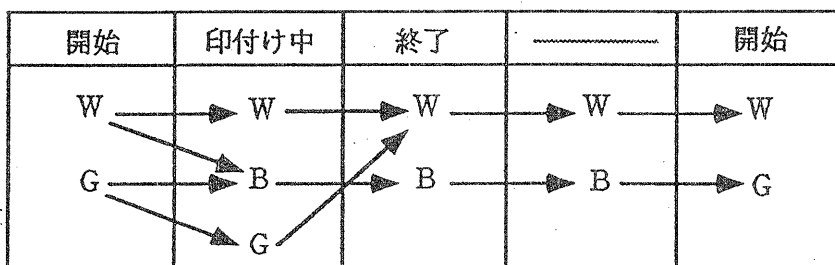


図1 三種類の状態の遷移

類のセルを簡単なビットの論理演算で高速に識別できるように印として三ビットを用いた例を示す。印として「001」、「010」、「100」の三種類を用いた場合の塵集めの実行における印の変化を、 $3n$ 回目の塵集め終了後の使用中を示す印を「100」、塵を「010」と仮定して $3n+1$ 回目の塵集めから $3n+3$ 回目の塵集めまでの計3回について印がどのように変化するかを示し、例として $3n+2$ 回目の処理手順について説明する。ここでの n は0以上の整数とする。

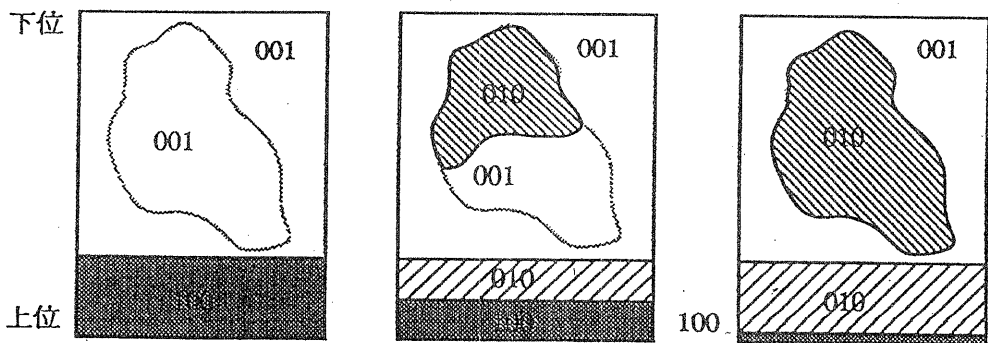


図2 $3n+2$ 回目のGC開始

図3 $3n+2$ 回目の印付け途中

図4 $3n+2$ 回目のGC終了

塵集め実行中の印の変化

- $3n+1$ 回目の塵集めでは使用中を示す印を「001」とする

前回使用中 (B) を示した印「100」の意味は、使用中から不明 (G) に切り替わる。確実に塵 (W) なのは「010」のセルであるから、「101」とビット論理積して偽ならば塵となる。終了後は「100」と「010」の印を持つセルが塵となるので、「001」とビット論理積して偽ならば塵である。

- $3n+2$ 回目の塵集めでは使用中を示す印を「010」とする (図2、3、4)

前回使用中を示した「001」の意味は、使用中から不明に切り替わる。確実に塵なのは「100」のセルであるから、「011」とビット論理積して偽ならば塵となる。終了後は「100」と「001」の印を持つセルが塵となるので、「010」とビット論理積して偽ならば塵である。

- $3n+3$ 回目の塵集めでは使用中を示す印を「100」を印とする

前回使用中を示した「010」の意味は、使用中から不明に切り替わる。確実に塵なのは「001」のセルであるから、「110」とビット論理積して偽ならば塵となる。終了後は「010」と「001」の印を持つセルが塵となるので、「100」とビット論理積して偽ならば塵である。

- $3n+4$ 回目以降の塵集めでは上記の $3n+1$ 回目、 $3n+2$ 回目、および $3n+3$ 回目の印の変化を繰り返し使用する

処理手順

図2から図4に $3n+2$ 回目の塵集めの開始から終了までのセル領域中の印 (B、W、G) と塵の変化を示す。始め「001」が使用中 (B) を、「100」が塵 (W) を表す (図2)。「001」の中で点線で囲まれた部分は、塵集め開始時に使用中であったために塵集め中に使用中として印付けされる部分である。塵集め開始時に使用中を示す印を開始前の印「001」と区別するために1ビットシフトした「010」とする (図3)。

使用中のセルに「010」の印をつけるため印「001」の意味は、使用中から不明 (G) に切り替わる。塵集め終了後は、「001」の印の付いたセルを塵として再使用できる (図4)。再使用されたセルは印が1ビットシフトされ使用中を示す「010」となる。図1から図3において「100」から「010」に印が変わった部分は印付けを少しずつ実行している間に、通常の処理により使用されたセルを示す。僅かに残った「塵」を示す「100」を印として持つセルは、次の塵集めが開始されるまではすべて再利用されるはずである。次の塵集めでは1ビットシフトして、「100」で使用中を表す。これは、塵集めするとともに使用中を表す印が1ビットずつローテイトさせているためである。3ビット使用することで、使用中を示す印の変更や塵かどうかの判定が簡単なビットの論理演算で行なうことができる。

塵集めは以下の手順で進められる。

- 条件設定：塵集めの開始条件である塵セルの残り個数、印付けの間隔および一回当たりの印付け数を設定する。

- 条件確認：塵セルの残り個数が設定個数未満ならば、コンス等の塵セルの要求時に印付けを開始する。塵セルの残り個数が十分あるならば、なにもしない。
- 印付け処理：使用中であるにもかかわらず不明の印の付いたセルを、指定された個数ずつ取り出して印を付ける。印をつけるごとに使用中個数カウンタに1を加える。このようなセルがまだ残っていれば、印付け処理を一定回数の塵セルの要求後に実行するようにする。残っていないならば塵集めを終了する。塵の数は、セル全体の個数からこのときの使用中個数カウンタの値を引いたものとなる。塵集めは塵セルがなくなる前に終了しなければならない。
- 塵セル要求時の塵セルの走査：塵セルは、セル領域より走査して見つけることになる。走査は前回取り出した塵セルの次のセルより開始し、塵セルが発見されたときに終了する。最終セルまできた場合は、セル領域の先頭から開始される。従って、走査の開始位置が順次移動していくことになる。塵セルを発見した時点で塵セルの残り個数のカウンタを1減らす。

3.2 リスト処理操作

塵集め中に通常の処理で起こるリスト構造の変化に対して、どのように対応するかという問題がある。通常の処理を中断して塵集めを行なう一括方式では塵集め中には一切他の処理は実行されないためリスト構造の変化などありえなかった。ところが、塵集め自体を通常の処理の実行中に少しずつ分けて行なう場合には、リスト構造が塵集めの間に変更される可能性がある。したがって、中断していた塵集めを再開したときに正しく印付けができない可能性がある。これについては、文献 [1, 2] の方法を採用した。すなわち、リスト変更操作である `rplacd` および `rplaca` の操作に負担をかけるものである。

3.3 動的特性

文献 [1, 2] に習って動的特性を見てみる。

図5では、横軸にセルの要求回数、縦軸はセル数を表す。 $F(t)$ と $A(t)$ は、 t 回目のセル要求時点での塵セルの数と使用中のセル数をそれぞれ表す。 N をシステムで利用できる全セル数、 M を塵集め開始時の塵の個数、 K を1回に印付けするセルの個数とする。

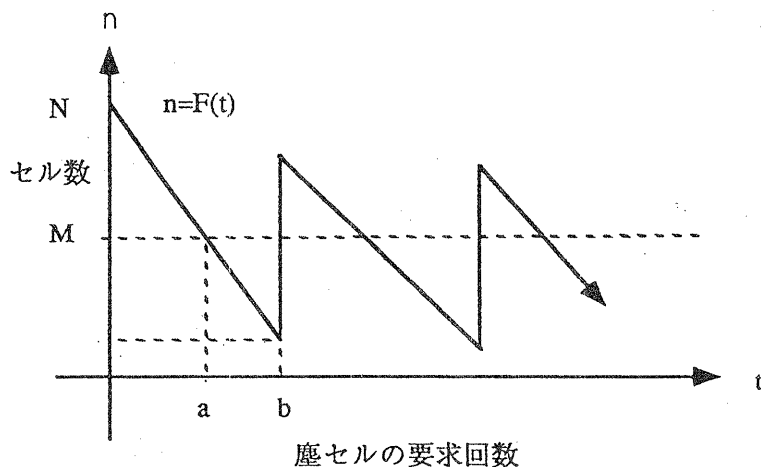


図5 塵セルの変化

セルを要求すると同時に塵セルの個数一つずつ減少する。この個数が M になると塵集めが始まるが、印付け中は塵セルの回収は行なわれないのでセルの要求ごとに塵セルが一つずつ減少する。このため $F(t)$ のグラフは鋸形波形となる。

いま、塵集めが $t = a$ で始まり、 $t = b$ で終了したとする。印付け中のセル要求では、塵集め開始時に使用中のセルを印付けするため合計 $A(t)$ のセルに印を付ける。一方、印付け中のセル要求が K 個なので、 $A(a)/K$ 回のセル要求で印付け（塵集め）が終了する。従って、塵集め終了時の塵セルの個数は

$$M + (N - A(a) - F(a)) - A(a)/K$$

となる。ここで、塵セルがなくならないための十分条件を考える。これは、すべての時点において $F(t) \geq 0$ となる十分条件を示せばよい。塵集め中には、 $A(a)/K$ の消費があるので

$$M \geq A(a)/K$$

である。ここで $A(a)$ は塵集め開始時に使用中のセル数であるが、これは塵集めをいつ開始するかに依存し、推定するのが困難である。そこで、あるリスト処理プログラムの実行中に、同時に使用されるセル数の最大値なら、推定が比較的容易であるので、その値 A_{max} とすると、上の条件が成り立つための十分条件は

$$M \geq A_{max}/K \quad (1)$$

となる。上の議論では、塵集め開始時に M 個の塵セルが残っていることを前提としている。しかし、状況によっては塵集め終了時の塵セルの個数が M 以下となり、ただちに次の塵集めが開始することもありえる。そこで上式が成り立つための十分条件には、さらに、 $F(b) \geq M$ がすべての塵集め終了時 $t = b$ について成り立つことが必要である。したがって、上式と同様の計算により

$$\begin{aligned} F(b) &= M + (N - A(a) - F(a)) - A(a)/K \geq M \\ M &= F(a) \end{aligned}$$

となる。この2式より全セル数 N は

$$N \geq M + (1 + 1/K)A_{max} \quad (2)$$

となる。したがって、式 (1) と (2) の両方を満たせば塵セルがなくなることはありえない。たとえば、 $K = 20$ であれば、この両式を満たす N の最小値は $1.1A_{max}$ である。一方通常の処理を中断して塵集めを一度に行なう一括方式の場合は明らかに $N \geq A_{max}$ であれば、なくなることはない。 $K = 20$ の場合を例にとれば、最悪の場合でも、本アルゴリズムを採用した場合は、一括方式の1.1倍のセルがあればセルがなくなることはないと保証される。なお、この場合、(1) と (2) を満たす M の最小値は

$$0.05A_{max}$$

である。つまり、塵セルの個数が全セル数の5%になったときに塵集めを開始すればよい。

次に、塵集めの回数を調べる。式 (2) を仮定すると、ある塵集めが $t = a$ で始まってから次の塵集めが始まるまでに、セル要求は

$$A(a)/K + (N - A(a) - F(a)) - A(a)/K = N - A(a) - M$$

回呼び出される。簡単化のために $A(t)$ を定数 A_{mean} とすると、セル要求が T 回呼び出される間に塵集めが起こる回数は

$$T/(N - A_{mean} - M)$$

で与えられる。上の $K = 20$ の場合の M と N の最小値 $M = 0.05A_{max}$ 、 $N = 1.1A_{max}$ を使い、 $A_{max} = 2A_{mean}$ として計算すると、

$$0.77T/A_{mean}$$

回の塵集めが起こることになる。

§4. 実験

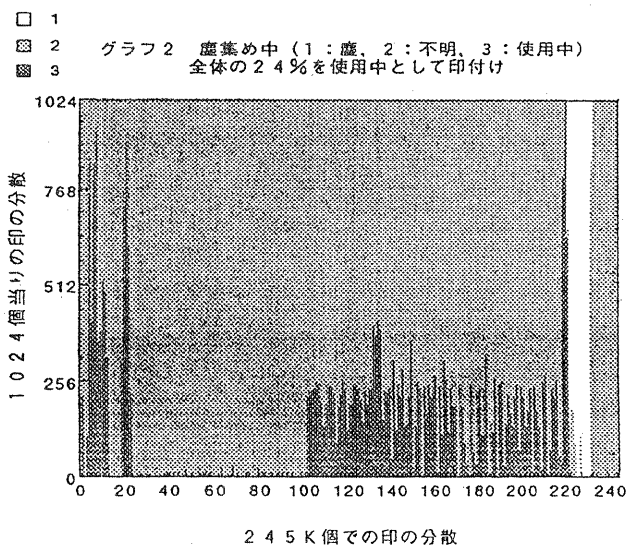
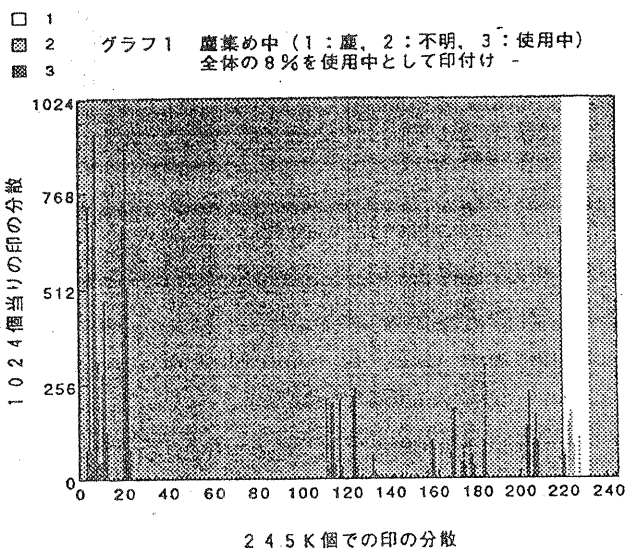
本方式の塵集めを実装した Scheme[3] ベースのリスプ処理系上に、エキスパートシェル (OPS5 ライクなエキスパートシェル) を載せ、この上でテスト用のプログラムを実行してセルの使用状況を調べた。ただし、処理系にはまだコンパイラがないのでインタプリタで実験を行なった。

4.1 実験内容

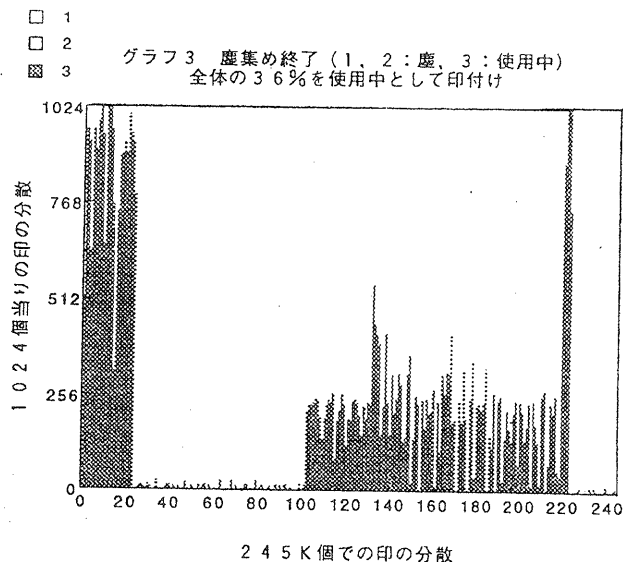
テスト用のプログラムとしてモンキー&バナナ問題 [4] を用いた。これは、エキスパートシェル自体も開発中であるため一般的で簡単な問題を選んだ。エキスパートシェルおよびモンキー&バナナ問題のルールのロードから、実行終了までに発生した塵集めによるセルの使用状況と塵セルを取り出すためのセルの走査数を調べた。

4.2 セルの使用状況

塵集めの起動条件を次のようにして実験した。未使用セルの個数が全体 (245 K個) の 5% (12.25 K個) になったときに塵集めを開始し、100回コンス (0.1 K個) するごとに 1% (2.45 K個) づつ印付けしていく。この実験でのセル領域の状態変化をグラフ 1、2、および 3 で示す。グラフは 1 K 単位でのそれぞれの印の割合を示している。横軸は、セル領域での位置を示し (何 K 目か)、縦軸は 1 K 個内でのセルの割合を示す。内容は使用中の印が 2 から 3 に移り、いままでは使用中を示していた印 2 は、塵集め中「不明」を示すことになる。



グラフ1、2は塵を示す印1のセルを消費しながら塵集めを行なっているときのセル領域の様子を示している。グラフ1は、8回目（全体の8%の印を付け変えた）印付けを行なったときのセル領域の様子を示している。グラフ2は、24回目（全体の24%の印を付け変えた）印付けを行なったときのセル領域の様子を示している。グラフ3は塵集めが終了した（36回目、すなわち全体の36%の印を付け変えた）ときのセル領域の様子を示している。この時点で印2がすべて塵となるので、使用中のセルは印3で、塵セルは印1、2となる。



4.3 本実験で塵セルをとり出すために走査したセルの個数

本実験において1個の塵セルをとり出すために走査したセルの個数を見てみる。

平均	0.165
最大	4588
最小	0
1K以上アクセスした回数	14

表1 アクセス回数

1K以上アクセスした回数の部分は、シェルのソース・コードがある領域をアクセスしたものである。以上より、ほとんどが塵であり、使用中セルもかなり分散しており連続しているものは少ないことがわかる。印付けに関しては、湯浅氏の方法[1, 2]と同じがあるが、回収に関しては本方式は異なり必要なときに塵セルを示す印を持つセルを探すだけである。たとえば湯浅氏の方法[1, 2]では本実験のような場合は、回収フェーズで100回のコンス中に1回ずつ1%（2.45K個）の回収処理を100回行なわれることになる。これは、2K個以上のセル領域の走査を100回実行しなければならないことを意味している。これに対して本方式の場合は実験では未使用セルを探すために1K個以上走査した回数は14回と少なかった。

§5. 問題点および解決のアイデア

- 塵セルの使用要求時にセル領域を走査して塵セルを取り出すためにかかる時間が、最大「セル領域内の使用中セル数」走査する時間になる可能性がある。

極端に長いセル領域の走査を防ぐためと塵集め（印付け）時のセル領域の走査時間を軽減するために塵セル取り出し用にキャッシュ・リスト（CL）を作成する。CLは、セル領域を走査せずに塵セルを取り出せるように塵セルを連結してリスト構造にしたものである。CLの作成方法は、まずプログラムの起動時にセル領域の最後の方のセルをCLとして連結しておく。最初の塵集め以降は塵集め終了後もCLは残っている（多めに連結しておくため）ので、塵セル要求時に残りのCLから塵セルを取り出しながら次の塵集め用のCLをセル領域の走査方向と逆方向に走査（セルの取り出しで最後になる部分）して作成する。そこで極端に長いセル領域の走査を防ぐために、たとえば1K個セル領域を走査しても塵セルが発見できない場合は、このキャッシュ・リストを使用する。

- 通常のマーク・スウィープ方式と違い、印を残すためにセル内に印用の3ビット（最低2ビット）のフィールドが必要となる。

アドレスに3ビットの印（§3.の例）とセルかどうかを示すタグ1ビット（タイプ・チェックの高速化のため）を埋め込むことを考えてみる（組合せは「0010」「0011」「0100」「0101」「1000」「1001」となる）。たとえば、32ビットCPUで使用するメモリを16MB（24ビットで表現される0からFFFFFFF）に制限した場合に、上記の4ビットを最上位1バイトの下位4ビットに埋め込むと見かけ上アドレスは以下のように変化する。

埋め込みなし	0000000から0FFFFFFF
「0010」	2000000から2FFFFFFF
「0011」	3000000から3FFFFFFF
「0100」	4000000から4FFFFFFF
「0101」	5000000から5FFFFFFF
「1000」	8000000から8FFFFFFF
「1001」	9000000から9FFFFFFF

表2 上位4ビットを使用したときの見かけ上のアドレス

アドレスを指定するときに上位ビットの数ビットが実際にはデコードされない場合には、その部分を使用できる。たとえば、MC68000のように32ビットでアドレス指定できるがアドレスとして有効な部分が下位24ビット（16MB）なのは上位の8ビットを使用できる。

仮想記憶システムの場合は、仮想記憶から実メモリへの変換メカニズムを利用して見かけ上のアドレスから有効アドレスへの変換を行えばよい。すなわち、表3の各アドレスがすべて同じ実メモリ領域をマップする方法である。この方法ならば、データをアクセスするときに一々正当なアドレスをとり出すためにマスクを行なう必要がなく、必要となときに印あるいはセルがどうかのタグ・ビットを参照することができる。専用ハードウェアを用いずに仮想記憶のメカニズムを利用することでタグ・アーキテクチャを実現できることになる。

問題点としては、仮想記憶から実メモリへの変換テーブルが最低でも7倍になることである(実メモリ16MBを仮想メモリ112MBとして扱う)。セルかどうかを示すタグ1ビットのみを埋め込む場合は2倍で済む。

- コピーによるセルの再配置を実行しないために、仮想記憶システムでは、使用中セルの分散によるページフォルトによるディスク・アクセスの発生という問題が起こる。

しかし、本方式は実メモリシステムを対象にしているのでページフォルトによるディスク・アクセスという仮想記憶システム特有な現象を考慮する必要がない。

- 実際はセル以外のシンボル、文字、数字、ベクタ等を含む色々なデータを取り扱う。

処理の高速化のために、ほとんどのデータ・タイプをセルと同一サイズにしている。セルと同一サイズにできないベクタ等の可変長データは、連続領域に格納される可変長データ部と可変長データ部へのポインタを持つヘッダ・セル(セル領域を使用する)に分けてある。可変長データ部については塵集め時にヘッダ・セルを印付けすると共に、その可変長データ部をコピーして再配置することで可変長データ用の連続領域の圧縮を行なっている。

§6. まとめ

リスプ系の人工知能言語において、塵集めは非常に重要な問題である。塵集めが、システムのパフォーマンスを決定するといっても過言ではないからである。明示的な回収フェーズをなくすことで、全体の処理時間は変わらないにしても通常の処理の中断時間を細かく分散できる。また、特殊なハードウェアを使用せずに通常のCPUで安価に実現できるので、リスプ系の言語でアプリケーションを構築する場合に問題となる塵集めによる処理の中断に対して有効な方法であると考ええる。

今後、§5.で提案した解決方法について調査を行なって行きたい。

§7. 謝辞

この「塵集め」方式について有効な討論をしていただいたNTT基礎研究所の竹内研究グループの皆さんに感謝いたします。また、技術的なサポートをしていただいた田口利明研究員に感謝いたします。最後に、この研究の機会を与えて下さった橋本新一郎セコムIS研究所長に感謝いたします。

参考文献

- [1] 湯浅太一. 汎用計算機に適した実時間塵集め. 情報処理学会記号処理研究会報告, 41(4), 6 1987.
- [2] 湯浅太一. 汎用コンピュータでの実時間ごみ集め. サイエンス, September 1988.
- [3] H. Abelson et. al. Revised³ report on the algorithmic language scheme. Technical Report Ai-Memo-848a, MIT Artificial Intelligence Laboratory, September 1986.
- [4] Elaine Kant Lee Brownston, Robert Farrell and Nancy Martin. *Programming Expert Systems in OPS5*. Addison-Wesley Series in Artificial Intelligence. ADDISON-WESLEY PUBLISHING COMPANY, INC., 1985.