3M−7

「集めない」塵集め

中沢 雅博、田内 康之、斉藤 宗昭
セコムIS研究所 人工知能研究室

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、リスプ等の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、著者らは実メモリシステムを前提としたより中断時間の短かい実時間塵集めを提案する。

2 塵集め

人工知能システムの開発に使用されるプログラミング言語にLISPがある。このLISPで主におこなうリスト処理では、セルと呼ばれるデータを使用して実行される。使用できるセルが有限であることと、リスト処理では再利用されない使用済みセル（通常、塵と言う）が実行するごとに発生することから、こうしたセルを回収して再利用することが必要となる。この作業を塵集めと呼ぶ。塵集めを行なうには、従来は実行中のプログラムを中断しなくてはならなかった。

中断時間を短縮する方法としてマーク・スウィープ方式を通常の処理中に分割して実施するものが提案されている[1]。そこで著者らは、プログラム実行の中断をさらに短くしかもスループットを低下させない、実メモリシステム向けの分割型印付け塵集めを提案する。

3 著者らの塵集め

リスト処理では、印づけされるセルはセル領域全体から比べると少ないと一般に言われている。これはほとんどのセルが途中結果の保存等の一時使用のせいである。未使用セルの使用要求があったときに、未使用セルが一つでもあればよい。すなわち、要求時に最低限必要な量があればよいわけであるから、回収が一度におこなわなければならない理由はない。またマーク・スィープ型の塵集めを用いた場合、塵はセル領域中にかなり均等に分布していることがわかった。

著者らはこれらの点に着目し、塵を回収する代わりにセル領域中の使用中セルと塵をつねに区別できるように印を残し、未使用セルの使用要求があったときにセル領域を走査して塵を回収する方式を考案した。この方式では、従来の塵の回収（スィープ）処理にかかる時間が、未使用セルの要求時毎に分割され埋没することになる。

3.1 塵集めの動作

塵集めの動作としては、使用中のセルを識別するための印付け処理のみを行なう。印付けは未使用セルがある一定個数以下になったときに開始し、通常の処理を中断して一度に実行するのではなく少しづつ実行していく。そして、塵集め開始時に使用中であったセルが全部印付けされたときに終了する。印付けを少しづつ実行していくため、印付け中のセルには、「塵」、「使用中」および「塵かあるいはまだ使用中か不明」の三種類のセルが存在することになる。そこで、印として3ビットを使用する。

図1から図3に塵集めの開始から終了までのセル領域中の印と塵の変化を示す。初め「００１」が使用中を表す印、「１００」が未使用セルである（図1）。「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印付けされる部分である。

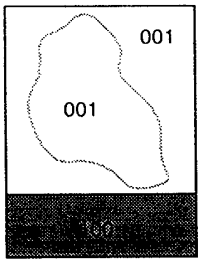


図1　ＧＣ開始

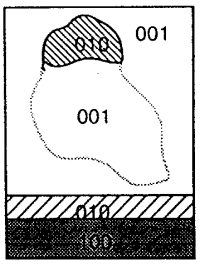
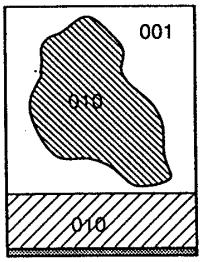


図2　印付け中



* 塵集めの開始から終了までの間に通常の処理で再利用されたセル
*
100
図3　ＧＣ終了

塵集め開始時に使用中を表す印を開始前の印「００１」と区別するために1ビット分シフトした「０１０」に変える（図2）。使用中のセルに「０１０」の印をつけるため印「００１」のセルは、「塵かあるいはまだ使用中か不明」の状態となる。

塵集めの終了後は、「００１」の印の付いたセルを塵として再使用できる（図3）。再使用するセルは印が1ビット分シフトされ使用中を示す「０１０」となる。図2、図3において「１００」から「０１０」に印が変わった部分は印付け処理を少しづつ実施している間に、通常の処理により使用されたセルを示す。

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、塵集めするごとに使用中を表す印が1ビットづつローテイトさせているためである。

また決して塵とならない永久使用セルに特定の印（「１１１」）を付けることによって、印づけの対象セルを減らすことができる。使用中のセルの印は印づけが終る毎に1ビットずつシフトしていくのでビットの論理演算で判定できる。また永久使用のセルは全ビットを1にす

A Garbage Collection without Sweep Phase
Masahiro Nakazawa, Yasuyuki Tauchi, Muneaki Saito,
Artificial Intelligence Department, SECOM Intelligent Systems Laboratory

ることで常に使用中とすることができる。

3.2 塵集め中でのリスト処理操作

セル領域の印付けを分割しない場合は、塵集め中にはプログラムの実行が中断されるのでデータ構造の変化などありえなかった。ところが、塵集め自体を分割して実行する場合には、プログラムの実行がおこるので、データ構造の変更の可能性がある。これについては、リスト変更操作である rplaca および rplacd の操作の中で対処する方法 [1] をとった。

3.3 動的特性

図4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未使用セル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 を塵集め開始時の未使用セルの個数、K を一回で印づけするセルの個数とする。いま、塵集めが $t = a$ で始まり、$t = b$ で終了したとする。

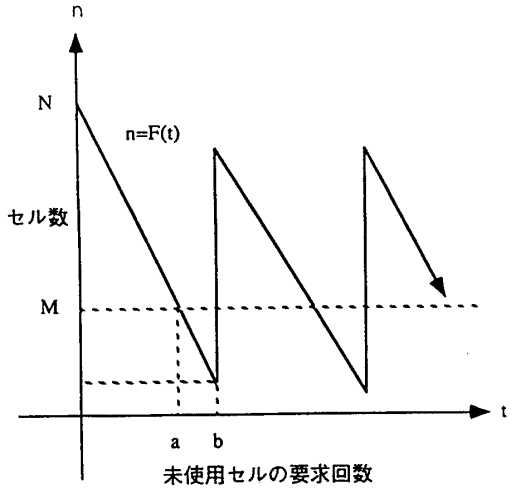


図4 未使用セルの変化

塵集め終了時の未使用セルの個数は $M + (N - A(a) - F(a)) - A(a)/K$ となる。未使用セルがなくならないための十分条件は、すべての時点において $F(t) \geq 0$ である。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。
使用されるセル数の最大値を Amax とすると、

$$M \geq Amax/K \quad (1)$$

となる。上の議論では、塵集め開始時に M 個の未使用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未使用セルの個数が 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、$F(b) = M + (N - A(a) - F(a)) - A(a)/K \geq M$、$M = F(a)$ により

$$N \geq M + (1 + 1/K)Amax \quad (2)$$

となる。式 (1) と式 (2) の両方を満たせば未使用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 1.1Amax である。一方、一括方式であれば、明らかに $N \geq Amax$ であれば、なくなることはない。したがって、$K = 20$ の場合を例にとれば、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。この場合、式 (1) と式 (2) を満たす M の最小値は 0.05Amax である。つまり、未使用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 Amean 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 $T/(N - Amean - M)$ で与えられる。$K = 20$ の場合の M と N の最小値 $M = 0.05Amax$、$N = 1.1Amax$ を使い、$Amax = 2Amean$ として計算すると、$0.77T/Amean$ 回の塵集めが起こることにある。ちなみに、[1] での同一条件下のそれぞれの値は、$N = 1.216Amax$、$M = 0.105Amax$、$0.82T/Amean$ となり、本処理方式のほうが値が改善されていることがわかる。

4 本方式での問題点

- 本方式は、未使用セルの使用要求時にセル領域を走査して未使用セルをさがすためにアクセス時間が最大「走査する領域内の使用中セル数」分だけ時間がかかる可能性がある。
- 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 3 ビットのフィールドが必要となる。
- セルのコピーによる領域の圧縮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。

5 まとめ

新しい実時間塵集めのアルゴリズムについて述べた。LISP 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を最小かつ平等に負担できることは、大きな意義がある。
今後、著者らが開発している Scheme[2] をベースとした実時間リスプ言語の処理系に本アルゴリズムを実装し、アプリケーションを作成の上有効性の確認を行ないたい。

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] H.Abelson et. al. Revised^3 report on the algorithmic language scheme. Technical Report Ai-Memo-848a, MIT Artificial Intelligence Laboratory, September 1986.