計算機アーキテクチャ 83−36
（1990. 7. 18）

分散メモリ並列計算機上での実時間GC

小池 汎平, 田中 英彦
{koike,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学 工学部

概要

ガーベジコレクション(GC)をはじめとする自動的なメモリ管理は、記号処理を行なうために必要不可欠な基本技術である。分散メモリ構成の大規模並列計算機で記号処理を行なうためには、分散メモリ環境を考慮した効率の良いGCアルゴリズムを開発する必要がある。また、記号処理の応用範囲を広げるためには、GCによる停止時間を十分短くし、しかも停止時間の上限を保証する、GCの実時間化が是非とも必要である。本稿では、Bakerによって提案された逐次型計算機用の方法をもとにして、分散メモリ構成の並列計算機上で実時間GCを並列実行する方式についての基本的な検討を行なう。

A REAL TIME GARBAGE COLLECTION METHOD FOR DISTRIBUTED MEMORY PARALLEL COMPUTERS

Hanpei Koike and Hidehiko Tanaka
{koike,tanaka}@mtl.t.u-tokyo.ac.jp
Faculty of Engineering, The University of Tokyo,
Hongo 7-3-1, Bunkyo, Tokyo, 113 JAPAN

Abstract

Automatic storage management mechanism such as garbage collection (GC) is a fundamental technique for symbol processing. Symbol processing on a large scale distributed-memory parallel computer requires an efficient GC algorithm specially designed for distributed-memory environment. Realtime GC, which limits stop time to some short duration, is necessary to extend application areas of symbol processing. In this paper, a realtime GC method for distributed-memory parallel computers based on Baker's sequential method is proposed and discussed.

1 はじめに

記号処理言語の記述能力の高さは多様なデータを自由に動的に生成できる柔軟性によるところが大きい。プログラマはメモリ管理を気にすることなく複雑なデータ構造を駆使して高度なプログラムを作り上げることができる。ガーベジコレクション(GC)をはじめとする自動的なメモリ管理は、このような記号処理を行なうために必要不可欠な基本技術である。

いっぽう、並列計算機、特に大規模並列計算機では分散メモリ構成をとる必要があり、この上で記号処理を行なうためには、分散メモリ環境を考慮した効率の良いGCアルゴリズムを開発する必要がある。また、一般に並列処理では実行環境の管理が単純なスタック機構などと比べて複雑化するので、この点からもメモリ管理が重要となる。

従来より使用されている多くのGCの方式では、処理を一括して行なうため、GCのたびに本来の記号処理が長時間にわたって停止することが問題となっており、このことが、対話処理やロボット制御など、実時間での応答を必要とする多くの応用分野への高度な記号処理の適用を妨げる原因となっていた。記号処理の応用範囲をより多くの分野に広げるためには、GCによる停止時間を十分短くし、なおかつ停止時間の上限が保証される、GCの実時間化が是非とも必要となる。

著者らは、これまでに、分散メモリ用のGCの方法として、逆転ポインタ法を用いることで余分な領域を必要とせずにコンパクションが可能な、Morrisのアルゴリズムを用いたマーク&スキャン方式のGC、及び、オブジェクトの寿命の概念を導入することによりGCの効率化と停止時間の短縮化が実現されるジェネレーションスキャベンジングGCを、通信性能の低下につながるリモート参照の間接化などを行なわずに実現する方法を提案してきた[1, 2]。

本稿では、より広い応用範囲で分散メモリ並列計算機上での記号処理を可能とするために、分散メモリ並列計算機のための実時間GCの実現法について検討する。

2 実時間GC

従来より使用されている多くのGCの方式では、処理を一括して行なうため、GCのたびに本来の記号処理が長時間にわたって停止することが問題となっており、このことが、対話処理やロボット制御など、実時間での応答を必要とする多くの応用分野への高度な記号処理の適用を妨げる原因となっていた。

上記の問題点を解決し、記号処理の応用範囲をより多くの分野に広げるためには、従来の一括型GCに対して、(1)停止時間が十分短く、なおかつ、(2)停止時間の上限が保証される、という特性を持った実時間GCが必要となる。

これまでに、逐次型計算機のための実時間GCのアルゴリズムが幾つか提案されている。その代表的なものとして、Bakerの方法[3]、湯浅の方法[4, 5]などがある。これらの方法では、コピー、または、マーク&スキャンというGCのための基本操作を、本来の記号処理の合間に一定量づつ時分割実行することにより、一回当たりのGCによる停止時間を一定値以下の短い時間に抑えて処理の実時間性を保証する。この分割されたGC処理は、ヒープメモリから領域を確保する操作(Lispのcons)のたびに起動される。

このように記号処理とGCのための処理を並行して実行させる場合に注意を要するのは、以下の条件を満足させる必要があるという点である。

- いかなる記号処理操作列が並行して実行されても、必要なオブジェクトのコピーまたはマークに失敗してその領域が誤って回収されてしまう、ということがないこと。

このために、通常の記号処理側で行なう操作にも何らかの処理を付加してこれを支援する必要が生じる。Bakerの方法では、記号処理側にはコピー済みのポインタしか見せないようにすることによって上記の条件を保証している。このために、ポインタの読み出し操作(Lispのcar/cdr)のたびに、ポインタの指す先が未コピーならばその場でこれを積極的にコピーする、という処理を付加している。いっぽう、湯浅の方法では、相対的に実行頻度の低い破壊的代入操作(Lispのrplaca/rplacd)において、上書きされる前の値にマークをつける、という処理のみを付加することにより、より少ない記号処理側のオーバヘッドで、マーク開始時に到達可能であったオブジェクトの全てにマークが付けられることを保証している。しかし、コンパクションを必要とする場合には、読み出し操作における何らかのチェックは本質的に不可避である。

一般に、実時間GCでは、一括型のGCと比べてメモリの回収効率が低下する。これは、実時間GCの処理を開始した時点ではゴミではなかったがGCを遅延させていればゴミとなって回収できたオブジェクト(フローティングガーベジ)の分だけ余分にメモリが必要となるためである。Bakerの方法では、セミスペースを使い切る毎にのみ実時間GCを起動すること、湯浅の方法では、ヒープメモリ残量を示すカウンタを用意し、これが一定値を割った時点で実時間

GCを起動することにより、このメモリオーバヘッドを抑えている。この場合、実時間GCで必要となるメモリ量は一括型GCと比べて、1.2ないし1.5倍程度であることが報告されている[6, 4]。

以上で述べたような実時間GCを、分散メモリ構成の並列計算機で実現する方法について検討するのが本稿の目的である。

本稿で前提とするのは以下のような状況である。

- プロセッサとローカルメモリからなる多数の要素プロセッサ(PE)が相互結合網で結合された分散メモリ構成の並列計算機上で記号処理を行ない、各メモリ上の独立したヒープにオブジェクトがアロケートされる。
- 要素プロセッサ間を結合する通信機構はネックとならない程度に高い性能を持つ。
- 分散メモリ間にわたる単一のグローバルアドレス空間がある。
- ベクタなどの可変長オブジェクトの使用を許す。このため、コンパクションが必要となる。
- 実現性、汎用性を著しく損なわない範囲でのハードウェアサポートは仮定する。

以下の節では、まず、逐次型計算機のためのBakerの実時間GCを紹介し、次に、出発点となる分散メモリでの一括型のコピー方式のGCの実現法を示す。そして、このコピー方式をBakerの方法をもとにして実時間化する方法を、コピー処理の時分割化、コピー中のデータアクセスの方法の順で検討する。

3 Bakerの実時間GC

逐次型計算機においてコピー方式のGCの実時間化を実現したBakerの方法について以下で簡単に説明する。

Bakerの方法では、通常のコピー方式のGCと同様に新旧2つのメモリ空間を用意する。メモリを使い切ると、新旧メモリ空間の切り替え(フリップ)を行なってから、その時点でのルートが、新空間に最初にコピーされてコピーの準備が行なわれる。実際のコピー処理は、通常の記号処理においてヒープメモリへの領域の確保を行なうたびに一定量づつ行なわれる。また、メモリからポインタを読み出す操作においてこのポインタの指す先が旧空間であった場合にもその場でコピーを行なう。これによって、記号処理側には、新空間にコピー済みのポインタしか出現しないことが保証される。

新空間へのコピーのためにS, B, Tという3つのポインタを用いる(図1)。Sはコピー済みのオブジェクトを順にスキャンして新たなオブジェクトのコピーを起動していくためのポインタ、Bはコピー処理のための領域を確保するためのポインタ、Tはコピー処理とは独立に新空間の反対側から新たなオブジェクトの領域を確保するためのポインタである。SとBとで挟まれた領域をキューとして用いて、広さ優先でコピー処理が行なわれる。また、記号処理側には、新空間にコピー済みのポインタしか出現しないことが保証されているので、新たに確保したオブジェクトの領域はスキャンの対象から外すことができる。

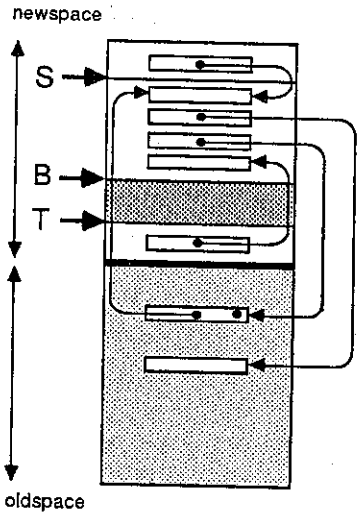


図1: Bakerの実時間GC

コピーの際に、旧空間のコピー済みのオブジェクトの先頭には、コピーの重複の防止と別の参照元への移動先の通知のために、コピー済みであることを示すマークをつけた移動先のアドレスを残す。記号処理側で他の参照元からのアクセスがあった場合には、コピー済みであることを検出して、この移動先アドレスを一段たぐる必要がある。

コピー処理はSがBに追い付くと終了する。いっぽう、コピー処理が終了する前にTがBに達した場合はメモリ領域不足のエラーとなる。

4 分散メモリでの一括型コピー方式GC

分散メモリ並列計算機上での実時間GCの実現方法を検討するに先だって、出発点となる一括型のコピー方式のGCを分散メモリ環境で効率的に実現する方法を示す。

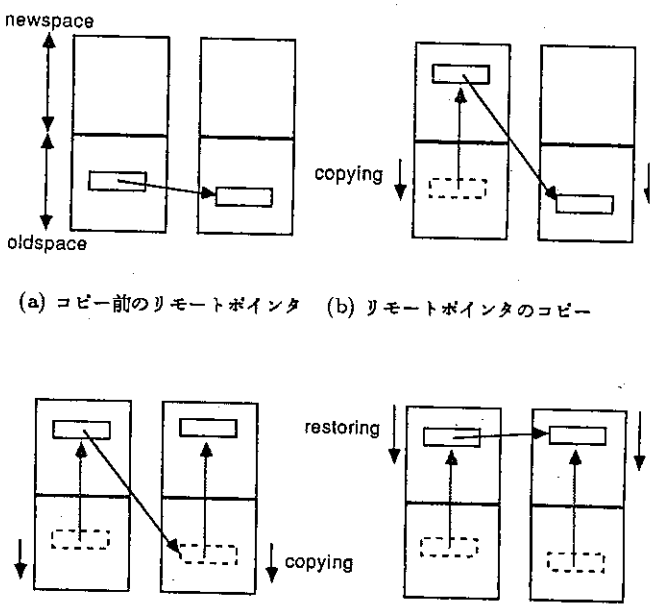
一般に、分散メモリ型の並列計算機で記号処理を行なう場合、(1)GCのルートをローカルに決定する、(2)コンパクションに伴うアドレスの変更をローカル

な操作にとどめる、等の理由により、分散メモリ間にわたるポインタ参照に間接参照テーブルを介させることが多い。しかし、この方法は、(1) メモリアクセスの手間を増大させる、(2) 通信の手間を増大させる、等の問題点がある [2]。このため、以下に述べる方法では、リモートメモリに対するポインタ参照もローカルメモリに対するポインタ参照と同様に間接化を行なわずにすむ方法を実現している。

通常のコピーアルゴリズムでは、オブジェクトをコピーするたびに、ただちに、参照元のポインタを新しく決定されたコピー先アドレスに書き換える。しかし、今回対象とするような分散メモリ並列環境では、コピーすべきオブジェクトがリモートメモリ上に存在する場合は、相手 PE に対してコピーの要求をする必要があり、リモートプロセッサの介在のために、コピーを要求してからコピー先アドレスが決定されて返されてくるまでの時間は長く、なおかつ、不確定なものとなる。このため、コピーのたびに結果が返ってくるまで処理を停止して待つか、非同期に返される値を待ち合わせる機構を用意することが必要となり、このままでは単純で効率的な実現は難しい。そこで、以下で述べるコピー方式では、全体の処理を全 PE の同期で区切られた複数のフェーズに分解して、コピー先アドレスの書き戻しを全てのコピー処理が終了するまで遅延させることにした。この結果、メッセージ送信、及び、リモートメモリの読み出しという単純な通信プリミティブのみを用いて効率的なコピー処理が実現可能となった。

処理は、(1) コピーフェーズ、(2) リストアフェーズ、の二つのフェーズからなり、ある PE でメモリを使い切ると全 PE で同時に起動され、全 PE が同期してそれぞれのフェーズを移行する。処理の手間はいずれのフェーズも生きているデータの量にのみ比例する。

コピーフェーズでは、各 PE が並列にルートから到達可能な自ローカルメモリ中の旧空間のデータを新空間にコピーする。この際に、旧空間のコピー済みのオブジェクトの先頭には、コピー済みであることを示すマークをつけた移動先のアドレスを残しておきコピーの重複を防ぐ。リモートポインタが出現すると、ネットワークを介してメッセージを送り、相手の PE に対してコピーの継続を要求する (b)。メッセージを送り付けられた PE は送られてきたリモートポインタをキューイングし、順次それらのポインタをルートと見なしてコピー処理を続行する (c)。1 つのオブジェクトが複数のリモートポインタから参照されていた場合は、複数のコピーメッセージが発行されることになるが、それらのメッセージはキューで直列化されて逐一処理されるので、重複したコピーが行なわれることはない。



(a) コピー前のリモートポインタ (b) リモートポインタのコピー

(c) リモートオブジェクトのコピー (d) 新しいアドレスの書き戻し

図 2: 分散メモリでの一括型コピー方式 GC

必要な全てのオブジェクトのコピーが終了するとリストアフェーズに移行し、各 PE が並列にルート及び新空間にコピーされたデータをスキャンし、リモートポインタが出現するたびに、ネットワークを介して、参照先のメモリの旧空間中に残された移動先アドレスを読み出して、参照元のリモートポインタに書き戻す (d)。

以上の方法により、分散メモリ上でのコピー操作を、リモート参照の間接化などを一切行なわずに、メッセージ送信、及び、リモートメモリの読み出しという単純な通信プリミティブのみを用いて効率的に実現できる。この結果、通常の処理でのメモリアクセスやデータ転送も極めて単純で高速なものとなる。また、コピー操作は、粒度の細かい処理に分けられ、待ち合わせ処理も必要ないので、高い並列度が期待できる。

5 分散メモリでのコピー処理の時分割実行

以上で述べた 分散メモリのためのコピー処理を、Baker の方法をもとにして実時間化する方法について述べる。いうまでもなく、ポイントとなるのはリモートポインタの取り扱いである。

本方式では、全 PE が、(1) アイドルフェーズ、(2) コピーフェーズ、(3) リストアフェーズ、という 3 つ

のフェーズを同期して移行し、(2) 及び (3) のフェーズで通常の記号処理と GC 処理が並行して実行される。

アイドルフェーズで、いずれかの PE がメモリを使い切ると、全 PE が同期してコピーフェーズに移行する。このとき、各 PE で新旧メモリ空間のフリップを行ない、その時点でのルートが、新空間に最初にコピーされる。

コピーフェーズでは、ヒープメモリへの領域の確保を行なうたびに、一定量づつのコピー処理を各 PE が非同期に行なう。ローカルなポインタ参照については、S, B, T という 3 つのポインタによって、Baker の方法と同様のコピー処理を行なう。

コピーすべきオブジェクトがリモートメモリ上に存在するものであった場合は、ネットワークを介してメッセージを送り、相手の PE に対してコピーの継続を要求する。メッセージを送り付けられた PE は送られてきたリモートポインタをキューイングし、領域の確保のたびに行なうコピー処理の際に、これらのポインタの指すオブジェクトのコピーも順次行なう。

リモートポインタの場合は、コピー先のアドレスを直ちに知ることができないので、Baker の方法とは異なり、リモートポインタは旧空間を指したままとなる。また、リモートオブジェクトのコピーが実際に完了する時期も不確定である。この様子を図 3に示す。そこで、コピーが完了するまでは、旧空間のオブジェクトがアクセスされ、コピー完了後は移動先アドレスを一段たぐって新空間のオブジェクトへのアクセスが行なわれる。

ここで述べた方法により、Baker の方法でコピーされるオブジェクトは全てコピーされる。従って、必要なオブジェクトをコピーしそこなって領域が誤って回収されてしまうことはない。

全ての PE で、S が B に追い付き、なおかつ、リモートコピーメッセージのキューが空になると、コピーすべきオブジェクトのコピーが全て終了したことになるので、コピーフェーズは終了し、全 PE が同期してリストアフェーズに移行する。

リストアフェーズでは、各 PE が並列に、ルート、新空間にコピーされたオブジェクトの領域 (すなわちメモリの上端から B までの領域)、及び、フリップ後に確保されたオブジェクトの領域 (すなわちメモリの下端から T までの領域) をスキャンし、リモートメモリの旧空間を指すポインタが現れるたびに、ポインタの指す先に書き込まれた移動先アドレスをネットワークを介して読み出して参照元のポインタに書き戻す。実時間性を保証するため、この処理も、ヒープメモリへの領域の確保を行なうたびに一定量ずつ時分割実行

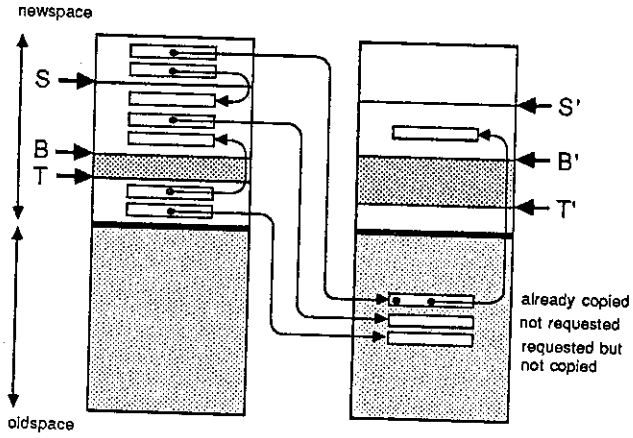


図 3: 分散メモリでの実時間 GC

する。

リストアフェーズにおいて、記号処理側でメモリからポインタを読み出す操作を行なう際に、このポインタの指す先がリモートメモリの旧空間であった場合には、その場で移動先アドレスの書き戻しを行なう。これによって、記号処理側には新空間にコピー済みのリモートポインタしか出現しないことが保証され、スキャン済みの領域に旧領域を指すリモートポインタが書き込まれてしまうことが防がれる。

全ての領域のスキャンを終了した PE は、アイドルフェーズに移行する。

6 コピーフェーズでのデータアクセス

本方式は、実時間でコピー処理を行なうため、旧空間を指すポインタが残されたままで記号処理が進められる。従って、このようなポインタを介して、コピー済みのオブジェクトをアクセスする場合には、旧空間に残されたコピー先アドレスをたどる必要があり、このための動的なチェックが必要である。

また、分散メモリ環境でコピー処理を行なうために、コピーを要求してから実際にコピーが行なわれるまでには時間の開きが生じる。従って、コピーが行なわれる前にオブジェクトがアクセスされる可能性があり、この場合には、旧空間のオブジェクトの方をアクセスすることになる。

さらに、並列環境でコピー処理が行なわれるため、コピー処理の実行中に、コピー中のオブジェクトが外部から通信プロセッサを介してアクセスされる可能性もある。

このような状況を考慮して、すでにコピー済みであるオブジェクトへの値の書き込みを誤って元の領域

に対して行なってしまう、などの矛盾が生じないように、データアクセス法を確立する必要がある。このためには、以下に示すような機構が必要となる。

コピー処理を行なう時にはコピーするオブジェクト単位でロックを掛け、排他的にコピー処理を行なうことにする。このために、コピー処理時にローカルメモリのバス上のアドレスを監視し、通信プロセッサによってオブジェクトのアドレス範囲のアクセスが行なわれようとした場合に、これをロックする機構を用意すれば良い。

いっぽう、コピーフェーズにおいて通信プロセッサを介してローカルメモリをリモートアクセスする場合には、オブジェクトがコピー済みかどうかのチェックを常に行なうことにする。このために、通信プロセッサのメモリアクセスの際には、オブジェクトの先頭を読み出してコピー済みかどうかのチェックし、このチェックと実際のオブジェクトの要素のアクセスとを排他的に行なうことにする。リモートアクセスで、この程度のオーバヘッドを付加することは、性能に大きな影響を与えることはないと考えられる。

7 おわりに

本稿では、分散メモリ環境において実時間 GC を実現する方法についての基本的な検討を行なった。

まず、逐次型計算機のための Baker の実時間 GC を紹介し、次に、検討の出発点として分散メモリでの一括型のコピー方式の GC の実現法を示した。そして、このコピー方式を Baker の方法をもとにして実時間化する方法を、コピー処理の時分割化、コピー中のデータアクセスの方法の順で示した。

今後、本方式の、より具体的な詳細化、特性の解析、効率化の方法についての検討を行なった後に、現在我々が開発を進めている並列推論エンジン PIE64[7] への具体的な実装法を検討して実験を行なうことを考えている。

なお、本研究は文部省の特別推進研究 No.62065002 の一環として行なわれている。

参考文献

[1] Xu,L., Koike,H. and Tanaka,H.: *Distributed Gabege Collection of the Parallel Inference Machine PIE64*, Proc. of the 11th Computer Congress, IFIP, Aug., 1989.

[2] 小池, 田中: 分散メモリ並列計算機上でのジェネレーションスキャベンジング GC, 並列処理シンポジウム '90 予稿集, 情報処理学会, 1990 年 5 月.

[3] Baker, H. G. : *List Processing in Real Time on a Serial Computer*, CACM, Vol.21, No.4, April, 1978, pp.280-294.

[4] 湯浅: *Realtime Garbage Collection on General-purpose Machines*, 日本ソフトウェア学会第 2 回大会, 日本ソフトウェア学会, 1985 年.

[5] 湯浅: 汎用コンピュータでの実時間 "ごみ集め", サイエンス, 日経サイエンス社, 1988 年 9 月号, pp.56-71.

[6] 小沢, 林, 服部: 実時間 GC の実現方式と評価, 情報処理学会論文誌, 第 29 巻 5 号, 1988 年 5 月, pp.465-471.

[7] 小池, 田中: 並列推論エンジン PIE64, 並列コンピュータアーキテクチャ, bit 臨時増刊, Vol.21, No.4, 1989, pp.488-497.