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

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。本稿では、オブジェクトの寿命の概念を導入することによって GC の効率化と GC 処理の疑似的な実時間化が可能とされる、ジェネレーションスキャベンジング GC を、分散メモリ上で並列実行する方式について検討する。

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。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複雑化するので、この点からもメモリ管理が重要となる。

本稿では、分散メモリ環境における GC では、予想される問題点を解決する鍵として、オブジェクトの寿命の概念を導入することがより一層重要となることを指摘し、寿命の概念を用いることによって GC の効率化がなされ、同時に GC 処理の疑似的な実時間化も可能とされる、ジェネレーションスキャベンジング GC を、分散メモリ上で並列実行する方式を提案し、その実現方法をハードウェア支援も含め検討する。

2　分散メモリ並列計算機における GC

本稿で前提とするのは以下のような状況である。

- 高性能な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) が高性能な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。
- 分散メモリ間に渡る単一のグローバルアドレス空間がある。
- 動的負荷分散が比較的高い頻度で行なわれ、プロセッサ間にわたる参照が多数生じる。
- 各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。
- ベクタなどの可変長オブジェクトの使用を許す。

メモリ参照が大きな割合をしめる記号処理を大規模並列計算機で行なうためには、多数の高性能なプロセッサが必要とする大きなメモリバンド幅を実現するためにメモリを分散化する必要がある。

また、記号処理が対象とする知識情報処理などの高度な応用は、単純な数値計算などと比べてプログラムの振舞いが不規則かつ動的であり、処理のローカリティを高め通信量を抑えるべく工夫することは当然必要であるものの [1]、高い並列処理効果を得るためには、積極的な動的負荷分散や動的データ配置が不可欠であり、その結果として PE 間に渡るデータ参照が数多く発生する。従って、このような状況に対応できる程度に十分に高い性能を持った PE 間通信機構は、いずれにしても用意しなければならない。

記号処理においてベクタなどの可変長オブジェクトをサポートすることは、実用的なプログラムを書く上で当然必要なことであり、このためには、ヒープ内に生じる様々なサイズの不要メモリ領域をフリーリストで管理するのでなく連続領域として回収するためにコンパクションを行い、その際、オブジェクトの移動に伴うアドレスの変更をそのオブジェクトを参照する全てのポインタに反映させる必要がある。

Generation Scavenging GC
on Distributed Memory Parallel Computers
Hanpei KOIKE, Hidehiko TANAKA
University of Tokyo, Faculty of Engineering.

以上のような条件のもとで、全処理の中に含まれるメモリ管理処理のオーバヘッドとこれに起因して本来の記号処理で発生するオーバヘッドの両方を最小にすることが目標である。もちろん、プログラムの静的な解析やデータそのものの性質によってあらかじめ回収時期が確定できるオブジェクトに関しては、メモリ領域の即時回収と再利用を最大限に行なうことも、このオーバヘッドを減らすために重要である [2]。

これまでに、逐次計算機のための GC として多くの方法が提案されているが [3]、分散メモリ構成の並列計算機で GC を行なう場合には、

- 分散メモリ間に渡るポインタ参照を効率良く取り扱うこと
- 全体としての処理性能の向上に見合うだけ、メモリ管理処理の性能も向上させること

などが新たに考慮すべき点である。

分散メモリ構成の並列計算機における GC を実現するにあたっての第一の課題は、分散メモリ間に渡るポインタ参照の取り扱いであり、

- あるメモリ内のオブジェクトが外部から参照されている時、このオブジェクトが参照されていることをどのようにして知るか。
- コンパクションによって外部から参照されている可変長オブジェクトのアドレスが変更された場合、これをどのようにして外部の参照元に知らせるか。

という問題を解決する必要がある。

第二の課題は GC 性能の向上である。プロセッサ台数が増加し処理性能が高まると、これに伴ってシステム全体でメモリを消費する速度が増加する。このため、並列処理による性能の増加に見合うだけ GC の性能の方も高めなければ、GC オーバヘッドの相対的な増加を招いてしまう。このためには、

- GC でもプロセッサ台数に見合うだけの並列性を取り出すこと
- 通信、同期、無駄な先行実行、などの並列環境ゆえの新たなオーバヘッドを極力抑えること

などが必要である。また、当然ながら GC 自体の効率化をできる限り図り、上記の問題が全体性能に及ぼす影響を相対的に抑えるという方策も必要である。

3 グローバル GC とローカル GC

分散メモリ構成の並列計算機において GC を行なう方法として次の二つが考えられる。

- グローバル GC
- ローカル GC

グローバル GC では、ある PE がメモリを使い切ると全部の PE が一斉に処理を中断して GC を行なう。ローカル GC では、メモリを使い切った PE だけが非同期に GC を開始し、他の PE は本来の処理を続ける。

グローバル GC は、

- GC の間隔が最も早くメモリを使い切った PE によって決められてしまい、残りの PE はメモリが残っていても GC に参加しなければならない。
- GC の処理自体が単独で、本来の記号処理における性能向上に見合うだけの高い並列性を持たなければならない。
- GC の開始時や終了時などに全 PE が同期を取り合うためのオーバヘッドがある。これは実時間 GC と併用するなどして GC による停止時間が短いときに特に問題となる。

などの点が問題である。

いっぽう、ローカル GC を採用した場合、PE 内の情報だけでデータが外部から参照されているかどうかを判定することと、PE 外にアドレスの変更を知らせずにコンパクションのためにデータを移動することが必要であり、このためには、個々の PE 毎に外部からの参照を管理するテーブルを用意し、外部からの参照を全てこのテーブルを介して行なわなければならない (図 1)。これには、

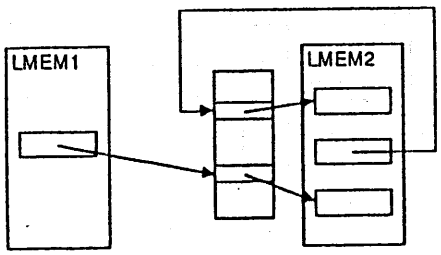
- 全てのポインタに対し参照管理テーブルを用意する (図 1(a)) [4]。
- ポインタをローカルポインタとリモートポインタに分け、リモートポインタだけ参照管理テーブルを介する (図 1(b)) [5, 6]。

という二つの方法が考えられる。しかし、このように参照管理テーブルを導入した場合には、次に述べる点が問題となる。

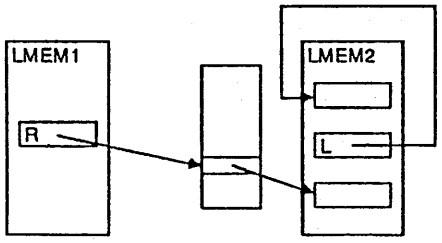
図 1(a) のように、全てのポインタに参照管理テーブルを用意する方法では、

- 間接参照を行なう分、ローカルメモリアクセスの手間までが増大する。

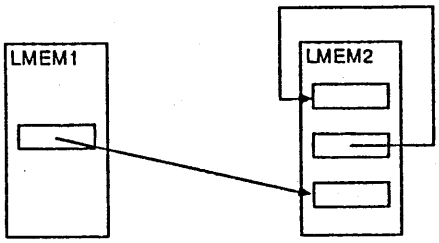
という点が問題となり、根本的な処理性能の低下につながりかねない。この問題を解決するために TLB のような高速小容量のアドレス変換バッファを用いることも提案されているが [1]、オブジェクトアドレス単位で参照の局所性が得られるとは考え難い。



(a) 全ての参照を参照管理テーブルを介する。



(b) 外部参照だけ参照管理テーブルを介する。



(c) 参照管理テーブルを全く使用しない。

図 1: 参照管理テーブルによる参照の管理

ローカル参照を高速化するために、図1(b)のように、ローカルポインタとリモートポインタを分離する方法を採用すると、

- ポインタを含むオブジェクトをプロセッサ間で移動させるたびに、その中に含まれている全てのローカルポインタとリモートポインタとの変換を行なわなければならない。特に、リモートポインタの一意性を保証するためには、ローカルポインタをリモートポインタに変換する際にハッシュなどの連想処理が必要であり、データ転送処理におけるオーバヘッドとなる。

という点が問題となり、オブジェクトを移動させる際の通信性能の低下を招く。更に、いずれの方法も、

- 外部から参照されなくなった時に参照管理テーブルエントリを回収する方法はいずれにしても別途用意しなければならない。このためには、外部参照を参照元で更に一段間接化し、参照の消滅を知らせ合うことなどが必要となり、更にオーバヘッドが増える。
- 既に外部からの参照がなくなっているオブジェクトが実際に不要であることが認識されるのは、少なくとも参照元のローカル GC が終った後の参照先のローカル GC においてとなる。これより前の GC では、実際は不要となっているオブジェクトが回収されない。これは、メモリの使用効率を低下させ、GC の頻度を高めてしまうとともに、GC の手間を増大させてしまう。
- ある PE がローカル GC を開始したあと、その PE のメモリ内のオブジェクトを他の PE からアクセスすることは難しく、結果的に他の PE の処理効率を低下させる。

などの点も問題となる。

以上の検討からわかるように、参照管理テーブルを必要とするローカル GC を採用すると、処理の複雑化、データ転送オーバヘッドの増大などを招いて、記号処理において必要とされる並列処理の柔軟性を阻害する可能性が高い。

並列記号処理言語の実装法の研究が進むにつれ、メモリ領域の再利用などの手法によって、メモリ使用効率の向上が達成されつつある。これにつれて、GC を起動する頻度自体は低くなっていくことが予想される。このような展望に立つと、GC のために参照管理テーブルを用意し、その結果として通信性能を低下させて本来の記号処理の性能を落とす方法は、問題があると考えられる。

いっぽう、図1(c)のように、管理テーブルによる参照の間接化を全く行なわずに分散メモリのグローバル GC を行なう方法が開発され、更に、本節の冒頭で挙げたような問題点が解決されるならば、処理の単純化、データ転送オーバヘッドの大幅な減少などがもたらされ、柔軟な並列記号処理の実現に大きく貢献すると考えられる。そこで次に、先に述べたグローバル GC の問題点を改善する方法について検討を行なう。

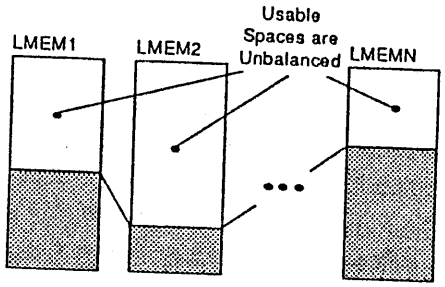
4 グローバル GC の問題点の解決方針

グローバル GC の様々な問題点を解決する目標は、

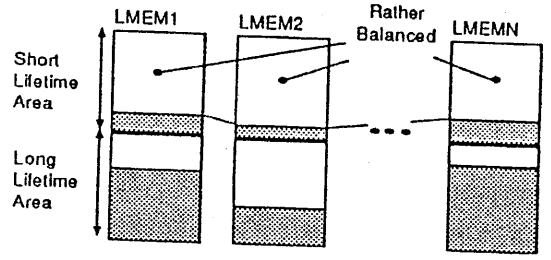
- 各 PE がなるべく同時にメモリを使い切ること
- GC の処理が単独で高い並列度を得ること
- 全 PE が同期を取り合うためのオーバヘッドを抑えること

である。さらに、記号処理の主要な対象である知識情報処理では人間との対話的な処理が大きな部分を占めるようになることが予想されるため、

- GC を実時間化すること

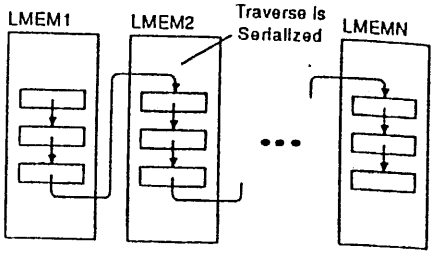


(a) 通常の GC の場合

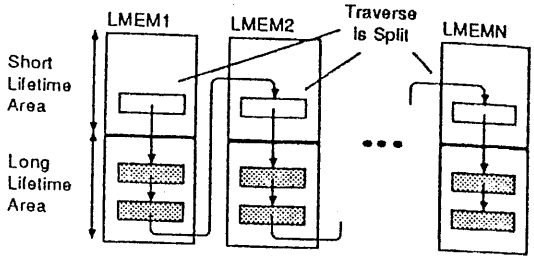


(b) 寿命を利用した GC の場合

図 2: 使用可能メモリ量のばらつきの改善



(a) 通常の GC の場合



(b) 寿命を利用した GC の場合

図 3: 処理の直列化の改善

も重要な課題である。以下で、これらの目標を達成する方針について検討する。

第一の目標は、メモリを使い切るまでの時間をできるだけ均一化することである。各 PE 間でメモリを使い切るまでの時間にばらつきが生じる原因は、

- PE 間のメモリ使用量がばらつくために使用可能メモリ量がばらつくこと
- PE 間のメモリ消費速度がばらつくこと

の二つに分けられる。後者は GC の間隔程度の長い期間に渡ってみればある程度平均化されることが期待され、大きな影響を及ぼすのは前者であると考えられる。

ここで、オブジェクトの寿命の概念を導入し、寿命の短いオブジェクトのみが通常の GC の対象となるような方法を採用することにすると、生成用の領域のサイズが十分大きいならば、GC の対象となる領域全体の大きさに占める使用メモリ量の割合は相対的に少なくなる。特に、固定サイズの生成専用の領域が用意されていれば全く問題はない。この結果、使用可能なメモリ量のばらつきが小さくなり、メモリを使い切るまでの時間が PE 間でばらつくという問題はかなり改善されることが期待される (図 2)。

第二の目標は GC の処理における並列性の向上である。GC の処理の基本はアクセス可能な全ての構造データを順にたどる処理である。このような処理において並列度を抑える大きな原因は、意味ネットワークなどの長い構造データの連鎖が多数の PE 間にまたがって存在した場合に、構造データをたどる処理が直列化してしまうことである。しかし、このような大規模な構造データの多くは、大部分が長い寿命を持つものであることが予想される。従って、寿命の概念を導入した GC を採用することによって、GC の処理のたびに、大きな構造データ全体を順に最後までたどらなければならないという状況は大幅に減ることになり、結果として GC の処理の並列度は高まることが予想される (図 3)。

PE 間で同期をとるためのオーバヘッドは、同期をとる回数自体を極力少なくすることと、同期信号をブロードキャストするために、同期信号線を PE 間に張り、専用の同期ハードウェアを用意することなどにより十分に小さくすることが可能である。これは、実時間型の GC を導入し、GC の起動される間隔が短くなった場合に特に有効となるであろう。

以上をまとめると、グローバル GC の問題点を改善するための鍵となるのは、GC にオブジェクトの寿命の概念を導入することにあるといえる。そして、これに実時間化が伴えばなお良い。以下では、このような特徴を持った GC について検討する。

5 ジェネレーションスキャベンジング GC

オブジェクトの寿命の概念を導入することによって、寿命の長いオブジェクトに対する GC の手間を軽減するとともに、GC 対象領域を限定することにより一回の GC での停止時間を十分に短くして疑似的な実時間化を実現する GC の効率化手法が提案さ

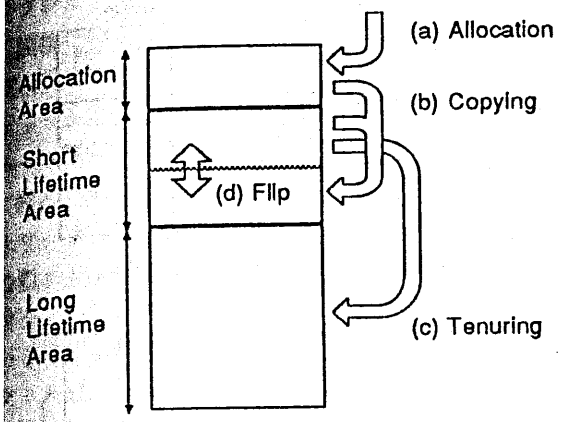


図 4: ジェネレーションスキャベンジング GC

している [7, 8]。このような方式のひとつに、ジェネレーションスキャベンジング GC [9] がある。

この方法では、ヒープ領域を、(1) 生成領域、(2) 二つの短寿命領域、および、(3) 長寿命領域に分ける (図 4)。新たなオブジェクトの生成は、生成領域に対して行なう (a)。生成領域を使い切ると、生成領域と短寿命領域中の生きたオブジェクトをもう一つの短寿命領域にコピーする (b)。また、このとき、何回かの短寿命領域の GC にわたり生き延びたオブジェクトは長寿命であることが仮定され、長寿命領域に移されて通常の GC の対象から外される (c)。コピーが終了すると 2 つの短寿命領域の役割を入れ換え (d)、本来の処理に戻る。

以上の処理は、生成領域と単寿命領域中の生きたデータに対してのみ行なわれるので、寿命の長いデータに対する GC のオーバヘッドを軽減することができる。また、GC による停止時間は通常十分短くすることができ、対話処理で必要とされる程度の実用上の実時間化が達成される。

ただし、GC の際に、長寿命領域内のオブジェクトから先はたどられないので、長寿命領域から短寿命領域のオブジェクトを指すポインタが生じた場合は、指されているオブジェクトがゴミと見なされて回収されないように、このようなポインタを管理して、ルートとして扱う必要がある。

この方法では、生成領域は固定長なので、分散メモリ間でメモリを使い切るまでの時間がばらつくという問題は改善されるであろう。また、寿命の概念が導入されることにより、たどりの直列化による並列度の低下の問題も改善されるであろう。更に、疑似的な実時間化も実現されている。

そこで、次に、ジェネレーションスキャベンジング GC を分散メモリ構成の並列計算機上で実現する方法を、分散メモリ環境でのコピー操作の実現、及び、分散メモリ環境でのルートポインタの管理方法に分けて検討する。

6 分散メモリでのコピー操作の実現

ジェネレーションスキャベンジング GC の基本となるコピー方式の GC を、分散メモリ環境で参照管理テーブルを用いずに高い並列度を引き出して実現する方法を示す。

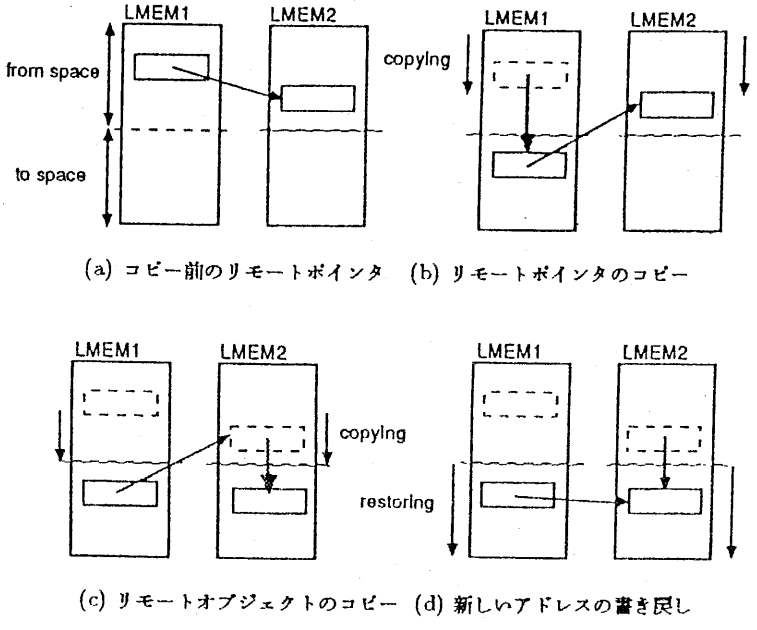
一般に、コピー方式の GC は、全メモリを旧空間と新空間にわけるため、一時には全体の半分しかメモリが利用できずメモリ利用効率が低いという問題点がある。しかし、ジェネレーションスキャベンジング GC においては、短寿命空間のみを二つの空間に分ければ良いので、メモリ使用効率は改善される。また、コピー方式の GC には、処理の手間がゴミの量によらず、生きたデータの量だけで決まるという特徴があり、大部分のオブジェクトが短期間でゴミとなる短寿命領域の GC として有利である。

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、いずれかの PE がメモリを使い切ると全 PE で同時に起動され、全 PE が同期して各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。処理を複数の大きなフェーズに分けることにより、細かな処理での待ち合わせが不要になり、オーバヘッドを減らせるとともに、並列度を上げ易くなる。

コピーフェーズでは、各 PE が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並列に新空間にコピーする。この際に、旧空間のコピー済みのデータの領域の先頭には、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く。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)。

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)。

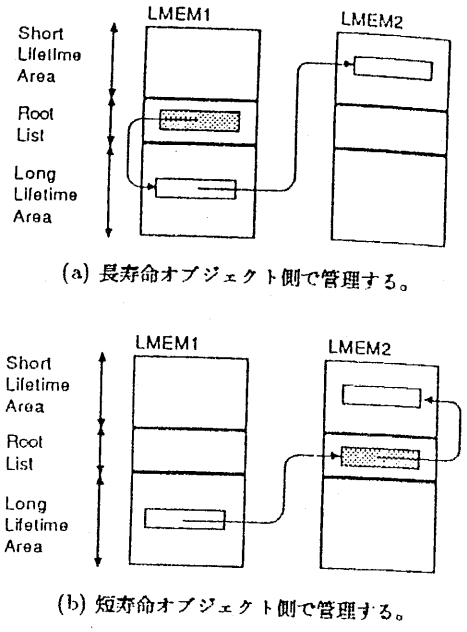
以上の方法により、リモート参照の間接化などを一切行なわずに、分散メモリ上でのコピー操作を実現できる。この結果、通常の処理でのメモリアクセスや



(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 5: 分散メモリでのコピー操作の実現



(a) 長寿命オブジェクト側で管理する。

(b) 短寿命オブジェクト側で管理する。

図 6: 分散メモリにおけるルートポインタの管理方法

データ転送は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、不要な待ち合わせ処理もないので、高い並列度が期待できる。

ジェネレーションスキャベンジング GC におけるコピー操作では、上記の基本操作に加えて、長寿命領域を指すポインタの先はコピーしないこと、及び、数回の GC にわたって生き延び、更に長い間生き続けることが予想されるオブジェクトを長寿命領域の方に移動させること、の二点の拡張が必要となる。しかし、これらはローカルな操作であり、分散メモリ環境ゆえの新たな工夫は必要ない。

7 分散メモリでのルートポインタの管理

ジェネレーションスキャベンジング GC を実現するにあたって検討を要するもう一つの課題は、ルートポインタの管理である。

ジェネレーションスキャベンジング GC では、長寿命領域中のオブジェクトから先をたどることはしないので、長寿命領域のオブジェクトから短寿命領域中のオブジェクトが参照されていた場合、そのオブジェクトがゴミとして回収されるのを防ぐために、このようなポインタ参照を全て把握しておき、 GC の際にルートとして扱う必要がある。このために、このようなポインタの書き込みの発生をハードウェアで検出してトラップをかける機構をプロセッサに用意する場合が多い [10]。また、短寿命オブジェクトが移動し、そのアドレスが変更された時に、この変更を長寿命オブジェクト中のポインタに反映しなければならない、という点にも注意する必要がある。

分散メモリ環境においては、検出されたルートポインタをどこでどのようにして管理するかが問題になる。ポインタの書き込まれた長寿命オブジェクトとポインタで指される短寿命オブジェクトが異なる分散メモリにある場合を考慮すると、上に述べた要件を満たす方法として、

- ポインタの書き込まれた長寿命オブジェクト側のメモリで長寿命オブジェクト内のポインタの位置を記憶しておく。(図 6(a))
- ポインタで指される短寿命オブジェクト側のメモリで短寿命オブジェクトのアドレスを記憶しておき、元のポインタは間接化する。(図 6(b))

の二つが考えられる。以下でこの二つを比較する。

前者の場合、このようなポインタ書き込みが検出された際の登録処理はローカルに行なわれる。しかし、後者の場合は、登録処理のためにネットワークを介した通信が必要になる。

前者の場合、管理されたポインタから指される長寿命オブジェクトをリストア操作の対象とすることにより、短寿命オブジェクトのアドレスの変更をポインタに直接反映させることが可能である。また、リストア操作の際に、指しているオブジェクトが長寿命領域に移動しポインタが短寿命オブジェクトを指さなくなったルートを順次削除することによって、ルートセットの単調増加を防ぐことができる。後者では、短寿命

オブジェクトのアドレスをローカルに変更するためには間接化が必要となり、このようなデータに対する通常処理時のアクセスの手間が増大する。また、ルートセットから不要なエントリを取り除くことは難しく、ルートセットは単調に増加する。

次に、ルートポインタが原因となって発生するネットワークトラフィックを比較する。前者では、GC の際に長寿命オブジェクト側のメモリでローカルに管理していたルートが指すオブジェクトがリモートである時、これのコピーを要求するために、ネットワークトラフィックが発生する。これは、オブジェクトが短寿命領域にあるあいだ、GC が起きるたびに発生する。後者では、トラップが発生した時、オブジェクトのあるメモリにルートを登録するために一度ネットワークトラフィックが発生するだけである。

前者では、ポインタの管理のために新たな通信プリミティブを用意する必要はない。後者では、ルートポインタを登録するための通信プロトコルを新たに用意しなければならない。従って、実現は前者の方が容易である。

以上より、前者の方法は、指す先が短寿命であるあいだ GC のたびにネットワークトラフィックが発生するという点が問題であるものの、登録の処理がローカルにできること、指す先が短寿命でなくなった時にルートから削除することができること、実現が容易であることなどから、有利であると思われる。

8 通信機構によるサポート

並列計算機において、目的に合致した高機能な通信プリミティブを持つ高性能な通信機構を用意することは、性能向上の点からも、行なうべき処理とその効率化の方針を明確にする点からも、非常に重要である。これまでに述べた方式の GC を効果的にサポートするためには、以下のような GC 支援用の通信プリミティブを用意することが望ましい。

リモートメモリ上のオブジェクトのコピーをリモートのプロセッサに要求するために、

```
copy(remotepointer)
```

を用意する。ここで remotepointer はリモートメモリ上のオブジェクトのアドレスであり、該当するプロセッサに対し、コピーを要求するメッセージがポインタと共に送出される。メッセージを受け取ったプロセッサは送られてきたポインタをコピーすべきポインタのキューに追加し、順次コピーを行なう。コピーフェーズでリモートポインタが出現したときには、この通信プリミティブを実行すれば良い。また、ポインタのキューの領域があふれるとデッドロックの原因となるので、重複したコピーメッセージをハッシュ等により検出し、キューイングするメッセージ数の上限をプロセッサ台数に依らないようにする、などの工夫が必要である。

リモートメモリ上の移動先アドレスを読み出して、ローカルメモリ上のポインタに書き戻すために、

```
restore(remotepointer,location)
```

を用意する。ここで remotepointer はリモートメモリ上のオブジェクトのアドレス、location はローカルメモリ中のリモートポインタの置かれているアドレスであり、リモートメモリ上のオブジェクトの先頭に書き込まれた移動先アドレスがネットワークを介して読み出され、マークが落とされてローカルメモリに書き込まれる。リストアフェーズでリモートポインタが出現したときには、この通信プリミティブを実行すれば良い。

現在我々が開発中の並列推論エンジン PIE64 [11] の通信コプロセッサであるネットワークインタフェースプロセッサ (NIP) [12] は、並列記号処理向きの高機能な通信プリミティブを持つことを特徴としており、分散メモリの GC を支援するために上記の通信プリミティブが用意されている。ネットワークの回線接続等も含めて、copy コマンドは 16 クロック、restore コマンドは 15 クロック程度でメインプロセッサと独立に実行できる。

9 長寿命領域の GC

長寿命領域中のオブジェクトはゴミとなっても回収されない。長寿命領域は単調に消費され、やがて溢れる。従って、低い頻度ではあるが、長寿命領域の GC を行なう必要がある。このためには例えば、ローカルメモリ量に比例した手間を必要とするものの、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、循環構造も含めてすべての不要領域の回収が行なえる、Morris のアルゴリズムを用いた分散メモリ環境のためのマーク & スイープ型のコンパクション GC[13] などを用いることになる。

10 長寿命領域への移動

短寿命領域で数回の GC のあいだ生き続けたオブジェクトは、長寿命領域に移動され GC の対象から外される。これは、ある程度生き続けたオブジェクトはその後も生き続けることが期待されるという仮定に基づいている。しかし、これに反して、オブジェクトが長寿命領域に移動した途端にゴミになった場合、このオブジェクトはそこから指される単寿命オブジェク

トも含めて回収不能になり、GCの性能を悪化させる。このような事態が頻発しないように、オブジェクトが長寿命であるかどうかの判定方法は、データの性質、扱う問題の性質、プログラミング言語の性質などを考慮して、慎重に決定することが望ましい[14]。特に、並列記号処理言語では、プログラムの実行順序に不確定性があるため、データの寿命もより不確定になることが予想され、これを考慮する必要がある。

11 おわりに

本稿では、分散メモリ環境においてジェネレーションスキャベンジング GC を実現する方法について検討した。本稿で議論した問題は、複雑な構造を持った大規模なプログラムを実行した際に、はじめて顕在化するであろうものであり、小規模なベンチマークプログラム程度を用いてシミュレーションを行なっても、意味のある定量的な議論を行なうことはできない。従って、今後、厳密な定量的議論を行なうためには、

- 実機上での実際の応用プログラムを用いた評価

が重要であると考えている。本方式は、現在我々が開発を進めている並列推論エンジン PIE64 に実装し実験を行なう予定である。そして、これに基づき、実際の問題の性質を利用しつつ、

- 効果的な長寿命オブジェクトの判定法の検討

を行なう必要がある。

また、今回提案したコピー処理は、オブジェクトのコピーを同一PE内でのみ行なうものであるが、NIPの持つリモートヒープアロケーション機能を利用することによって、これをPE間にわたるコピーの可能な方法に拡張し、並列記号処理における大切なテーマである負荷分散 / データ配置の問題と統合して、

- GC時のオブジェクトマイグレーションによるオブジェクトの最適再配置の実現

の方法を検討することも重要な課題である。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] 日高, 小池, 舘村, 田中: 実行プロファイルに基づくPIE64 の負荷分散方式, 並列処理シンポジウム '90 予稿集, 情報処理学会, 1990 年 5 月.

[2] 小池, 田中: 単一参照アノテーションを用いた論理型言語の最適化コンパイル, 情報処理学会第 38 回全国大会予稿集, 6Q-1, 1989 年 3 月.

[3] Cohen, J.: *Garbage Collection of Linked Data Structures*, ACM Computing Surveys, Vol.13, No.3, 1981.

[4] Mohamed-Ali, K.A. and Haridi, S.: *Global Garbage Collection for Distributed Heap Storage Systems*, TRITA-CS-8502, Dept. of Computer Systems, Royal Institute of Technology, Stockholm, April,1985.

[5] Rudalics, M. : *Distributed Copying Garbage Collection*, Proc. of ACM Conf. on Lisp And Functional Programming, 1986, pp.364-372.

[6] Ichiyoshi, N., Miyazaki, T. and Taki, K.: *A distributed implementation of flat GHC on the Multi-PSI*, Proc. of the 4th International Conference on Logic Programming, 1987.

[7] Lieberman, H. and Hewitt, C. : *A Real-Time Garbage Collector Based on the Lifetimes of Objects*, CACM, Vol.26, No.6, June, 1983, pp.419-429.

[8] Moon, D. A. : *Garbage Collection in a Large Lisp System*, 1984 ACM Symposium on Lisp and Functional Programming, Aug., 1984.

[9] Ungar,D.: *Generation Scavenging: A Non-disruptive High Performance Storage Reclamation Algorithm*, SIGPLAN Notice, Vol.19, No.5, May, 1984.

[10] Ungar,D. et al.: *Architecture of SOAR: Smalltalk on a RISC*, Proc. 11th Int'l Symp. on Computer Architecture, June, 1984, p.1887.

[11] 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.

[12] 清水, 小池, 島田, 田中: 並列推論マシン PIE64 のネットワークインタフェスプロセッサ, 並列処理シンポジウム '89 予稿集, 情報処理学会, 1989 年 2 月.

[13] 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.

[14] Ungar,D. and Jackson, F.: *Tenuring Policies for Generation-Based Storage Reclamation*, Proc. of OOPSLA'88, Sept., 1988.