C5-1

分散メモリにおけるガベージコレクション

Garbage Collection on Distributed Memories

小池 汎平, 許 魯, 田中 英彦
Hanpei Koike, Lu Xu and Hidehiko Tanaka
{koike,xu,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学 工学部
Faculty of Engineering, The University of Tokyo

記号処理言語の記述能力の高さはデータを自由に動的に生成できるという点によるところが大きい。ガベージコレクションをはじめとするメモリ管理は、このような記号処理を実現するために必要不可欠な基本技術である。いっぽう、並列処理計算機、特に大規模並列処理計算機では、分散メモリ構成を採る場合が多く、このような計算機で記号処理を行なうためには、分散メモリ環境での処理を考慮した効率の良いガベージコレクションアルゴリズムを開発する必要がある。本稿では分散メモリ環境におけるガベージコレクションの方式について検討を加え、これに基づいて、現在我々が開発中の並列推論エンジンPIE64で採用した方法について紹介する。

1 はじめに

記号処理言語の記述能力の高さはデータを自由に動的に生成できるという点によるところが大きい。ガベージコレクション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実現するために必要不可欠な基本技術である。いっぽう、並列処理計算機、特に大規模並列処理計算機では、分散メモリ構成を採る場合が多く、このような計算機で記号処理を行なうためには、分散メモリ環境での処理を考慮した効率の良いガベージコレクションアルゴリズムを開発する必要がある。本稿では分散メモリ環境におけるガベージコレクションの方式について検討を加える。

2 並列計算機におけるガベージコレクション

本稿で前提とするのは以下のような環境である。

- 分散メモリ構成の並列計算機によって記号処理を行ない、各メモリ上に独立してヒープが形成される。
- ベクタなどの可変長データの使用を許す。
- 動的負荷分散により比較的多くのプロセッサ間参照が生じる。
- 各要素プロセッサは高い通信能力を持った相互結合網で接続されている。

メモリ参照の局所性を得ることが難しい記号処理を大規模並列計算機で行なうためには、多数の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、記号処理においてベクタなどの可変長データをサポートすることは、実用的なソフトウェア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回収するためにデータのコンパクションを行い、その際、データの移動に伴うアドレスの変更をそのデータを参照する全てのポインタに反映させる必要がある。

更に、記号処理が対象とする知識情報処理などの応用は、プログラムの振舞いが不規則かつ動的であり、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が不可欠であると考えられる。そして、この結果としてプロセッサ間に渡るポインタ参照が数多く発生し、このような状況に対応するために、十分高いプロセッサ間の通信能力が要求される。

以上のような環境において、全処理の中に含まれるメモリ管理処理及びこれに起因したオーバヘッド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等によってあらかじめ回収時期が確定できる領域に対しては、メモリ領域の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[1]。

これまでに、単一プロセッサのためのガベージコレクションとして多くの方法が提案されているが[2]、分散メモリ構成の並列計算機でガベージコレクションを行なう場合には、

- 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

などの問題を新たに考慮する必要がある。

分散メモリ構成の並列計算機におけるガベージコレクションの第一の課題は、分散メモリ間に渡るポインタ参照の取り扱いであり、

- あるメモリ内のデータが外部から参照されている時、このデータが参照されていることをどのようにして知るか。
- 外部から参照されている可変長データのアドレスがコンパクションによって変更された場合、それをどのように

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。

いっぽう、プロセッサ台数が増加し処理性能が高まると、それに伴ってデータを動的に生成する頻度も高まり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、メモリ管理オーバヘッドの増加を抑えるためには、並列処理による性能の増加に見合うだけメモリ管理処理の性能の方も高める必要がある。このためには、

- メモリ管理処理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
- 分散環境ゆえの新たなオーバヘッドを極力増やさないこと

などが必要である。

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ガベージコレクションを行なう方法として次の二つが考えられる。

- グローバルガベージコレクション
- ローカルガベージコレクション

グローバルガベージコレクションでは、ある要素プロセッサがメモリを使い切ると全部の要素プロセッサが処理を中断してガベージコレクションを行なう。ローカルガベージコレクションでは、メモリを使い切った要素プロセッサだけがガベージコレクションを開始し、他の要素プロセッサは処理を続ける。

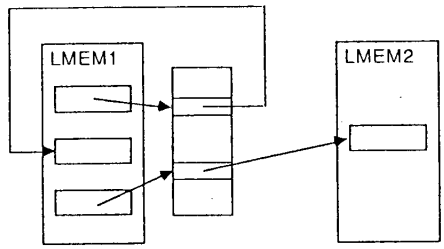
グローバルガベージコレクションは、

- ガベージコレクションの間隔が最も早くメモリを使い切ったプロセッサによって決められてしまい、残りのプロセッサはメモリが残っていてもガベージコレクションに参加しなければならない。
- ガベージコレクションの開始時や終了時などに全プロセッサが同期を取り合うためのオーバヘッドがある。これは実時間ガベージコレクションと併用するなどしてガベージコレクションによる停止時間が短いときに特に問題となる。
- ガベージコレクションの処理に高い並列性が得られることが必要である。

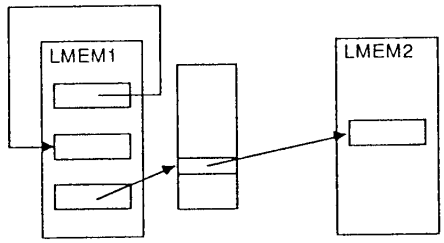
などの点が問題である。

いっぽう、ローカルガベージコレクションを採用した場合、プロセッサ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し、プロセッサ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必要があり、このためには、個々の要素プロセッサ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、

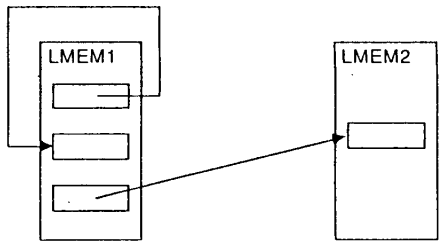
- 全てのポインタに対し参照管理テーブルを用意する(図1(a))[3]。



(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

- 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))[4]。

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。また、この問題点を解決しようとして、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、

- ポインタを含む構造データ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ず、この変換には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。
- リモートアクセスについては依然として間接参照を行なう分だけのオーバヘッドがあり、リモートアクセス機構を高速にした場合、このオーバヘッドも無視できない。

という点が問題となり、通信性能の低下を招く。更に、いずれの方法も、

- ある要素プロセッサがローカルガベージコレクションを開始したあと、その要素プロセッサのメモリ内のデータを他の要素プロセッサからアクセスすることは難しく、結果的に他の要素プロセッサの処理効率を低下させる。
- 既に外部からの参照がなくなっているデータが実際に不要であることが認識されるまでに時間がかかる。つまり、実際は不要となったデータが誤って必要とみなされてしまう。これは、ガベージコレクションの手間を増大させるとともに、メモリの使用効率を低下させ、ガベージコレクションの頻度を高めてしまう。
- 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に用意しなければならない。このためにオーバヘッドが生じる。
- 同一のデータを参照する値の異なるポインタが存在しうる。このため一般にはポインタの比較によってデータの同一性を判断することができない。このことは一部の記号処理操作の実現を複雑化する。

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用いてローカルガベージコレクションを採用すると、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、記号処理で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。

いっぽう、図 1(c) のように、間接参照テーブルを全く用いずに分散メモリのグローバルガベージコレクションを行なう方法が開発され、上で述べたような問題点が解決されれば、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ることになる。

4 PIE64 におけるコンパクション GC

現在我々が開発を進めている並列推論エンジン PIE64[5] は、64 台の推論ユニット (IU) がそれぞれローカルメモリを持ち、並列記号処理を行なう。PIE64 のためのガベージコレクションとして、まず手始めに、逆転リンクを用いることによって、余分な領域を必要とせずにポインタの書き換えを行なうことができ、更に、循環構造となった不要メモリ領域を回収することも可能な、Morris のアルゴリズムを用いたグローバルガベージコレクションを開発した [7]。

ガベージコレクションにおいてリモートポインタを効率的に処理するために、推論ユニット上のネットワークインタフェースプロセッサ (NIP) に mark, restore という 2 つのガベージコレクション支援コマンドを用意し、並列処理オーバヘッドを抑えることを目指すとともに、通常処理時における参照管理テーブルの必要をなくした [9]。これらのコマンドの動作の様子を図 2に示す。

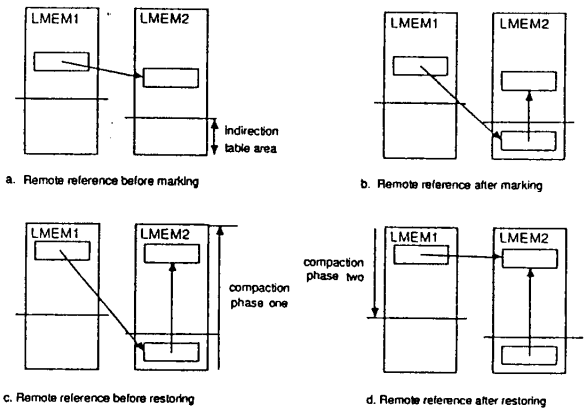


図 2: ガベージコレクションにおけるリモートポインタの書き換え

マーキングの際にリモートポインタが出現すると (a)、NIP に mark コマンドを発行し、リモートデータのある推論ユニット内に間接参照テーブルのエントリを作り、リモートポインタをここを指すように書き換える (b) とともに、相手 IU にマーキングの継続を依頼する。コンパクションの第 1 フェーズで間接参照テーブルの各エントリは新しい移動先に書き換えられる (c)。コンパクションの第 2 フェーズで間接参照テーブルのエントリを指すリモートポインタが出現すると NIP に対して restore コマンドを発行して、相手 IU 参照管理テーブル内にあるコンパクションで移動したデータの新しいリモートアドレスをリモートポインタに書き戻す (d)。

シミュレーションによると 64 台の推論ユニットが並列に 64 台分のメモリ (256MB) のガベージコレクションを実行する時間は 1 台で 1 台分のメモリ (4MB) を GC する場合と比べ 1.2 倍程度の時間に抑えられており、ガベージコレクションで高い並列性が得られていることがわかる [7, 8]。

5 コピー方式のガベージコレクションの実現

PIE64 の NIP に用意したコマンドを用いてコピー方式のガベージコレクションを実現することも可能である。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利用可能でないという問題点があるが、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ので、ゴミの発生率の高い言語の処理系の実現において速度的には有利である。

処理は、コピーフェーズ、リストアフェーズの 2 つのフェーズからなり、64 台の IU 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。

コピーフェーズでは、各推論ユニットが並列に旧空間のルートから到達可能なデータを新空間にコピーする。リモートポインタが出現すると、NIP に mark コマンドを発行して、相手の推論ユニットに対しコピーの継続を要求する。

リストアフェーズでは各推論ユニットが並列に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、NIP に restore コマンドを発行して、相手のメモリの旧空間中に残された移動先アドレスを書き戻す。

6 ジェネレーションスキャベンジング方式の実現

上で述べたコピー方式を拡張することにより、寿命の概念を採り入れたジェネレーションスキャベンジング方式[10]を導入することも可能である。このためには、パーマネントオブジェクト空間からテンポラリオブジェクト空間をさすポインタを管理する機構を用意すれば良い。これにより、ガベージコレクションの効率を高め、利用可能なメモリ量を拡張し、疑似的な実時間化を図ることが可能となる。

7 グローバルガベージコレクションの改良

以上で述べたように、グローバル GC を採用することにより、ローカル GC を用いた場合に生じる様々な問題点を解決することができるが、

- 各 PE がなるべく同時にメモリを使い切ること
- 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
- GC の処理で高い並列度を得ること

などの点に注意することも必要である。また、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。このため、

- GC を実時間化すること

も重要な課題である。

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、(1)PE 間のメモリ使用量のばらつきに起因する使用可能メモリ量のばらつき、(2)PE 間のメモリ消費速度のばらつき、に分類される。

後者は GC の間隔程度の長い期間に渡ってみれば平均化されることが期待されるので、大きな影響を及ぼすのは前者であろう。しかし、寿命の概念を導入した GC を採用すれば、通常 GC の対象となる領域の大きさに占める使用メモリ量の割合は少なくなり、その結果使用可能メモリ量のばらつきが小さくなるので、ここで述べる問題は大幅に改善されると予想される。

同期のためのオーバヘッドは、専用の割り込み回路を用意することにより解決できる。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。

グローバル GC の処理において並列度を抑える原因となるのは、複数の PE 間を順に渡る長いデータのリンクのために処理が直列化されることである。しかし、このようなデータの多くは、長い寿命を持つものであることが予想され、寿命の概念を導入した GC を採用することにより、このようなデータを扱う状況を大幅に減らし、処理の並列度を高めることができるであろう。

8 おわりに

本稿では、分散メモリ環境におけるガベージコレクションの実現方法について検討した。まず、グローバルガベージコレクションとローカルガベージコレクションを比較し、後者で必要となる参照管理テーブルの問題点について論じた。次に、並列推論エンジン PIE64 で用いたグローバルコンパクションガベージコレクションを紹介し、コピー方式への拡張法を示した。最後にグローバルガベージコレクションで生じる問題点の解決法を検討した。

参考文献

[1] 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.

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

[3] 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).

[4] 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.

[5] 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, Vol.21, No.4, 1989.

[6] Morris, F.L.: *A Time- and Space-Efficient Garbage Compaction Algorithm*, CACM Vol.21, No.8, Aug., 1978.

[7] 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.

[8] Xu,L., Koike,H. and Tanaka,H.: *The Gabege Collection System for Parallel Inference Machine PIE64*, Proc. of NACLP 89, Oct., 1989.

[9] 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予稿集, 情報処理学会, 1989 年 2 月.

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