分散メモリ環境でのガベージコレクションに関する考察

1Q-6

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

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点によるところが大きい。ガベージコレクション(GC)をはじめとするメモリ管理は、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場合が多く、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。そこで、本稿では分散メモリ環境におけるGCの方法について検討を行なう。

2 グローバルGCとローカルGC

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PE間参照が生じる。
- 各PEは通信能力の高い相互結合網で接続されている。

こ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に起因するオーバヘッドを最小にすることが目標である。もちろん、静的に回収が可能な領域に対しては、再利用を行なうことによってこのオーバヘッドを減らす必要がある[1]。

単一プロセッサのためのGC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GCを行なう場合、

- 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、どのようにしてそれを外部の参照元に知らせるか。
- どのようにして、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か。

などの問題を新たに解決しなければならない。

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は(1)グローバルGC、(2)ローカルGCが考えられる。グローバルGCでは、あるPEがメモリを使い切ると全てのPEが処理を中断して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GCを開始し、他のPEは処理を続ける。

グローバルGCでは、

- 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う。
- 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。

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し、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必要があり、このために外部からの参照を管理するテーブルを用意しなければならない。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[3]。
- 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[4]。

という2つの方法がある。いずれの方法も、

- あるPEがローカルGCを開始したあと、そのPEのメモリ内のデータを他の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。
- 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。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い、GCの手間を増大させるとともに、GC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必要となる。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。

という点が問題であり、更に前者の方法では、

- 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、この問題点を解決しようとした後者の方法では、

- ポインタを含む構造データをPE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。この変換にはハッシュなどの連想処理が必要である。
- 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。

という点が問題になる。

以上の検討からもわかるように、ローカルGCを採用して参照管理テーブルを用いると、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、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GC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。

---

[0] On Garbage Collection Method for Distributed Memories
Hanpei KOIKE, Lu XU, Hidehiko TANAKA
the University of Tokyo

3 PIE64のグローバルコンパクション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GCとして、まず手始めに、逆転リンクを用いて余分な領域を必要とせずにポインタの書き換えができ、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GCを開発した[6]。

GCにおいてリモートポインタを効率的に処理するために、IU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GC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性をなくした[7]。

シミュレーションによると64台のIUが並列に64台分のメモリ(256MB)のGC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、GCの処理で高い並列性が得られていることがわかる[6]。

4 PIE64におけるコピー方式のGCの実現

PIE64では、NIPの持つコマンドを利用してコピー方式のGC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IU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IUに対してコピーの継続を要求する。

リストアフェーズでは各IU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。

本方式を拡張し、寿命の概念を採り入れたGeneration Scavenging方式[8]を導入することにより、GC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もできるであろう。

5 グローバルGC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。

6 おわりに

本稿では、分散メモリ環境におけるGCの実現方法について検討した。まず、グローバルGCとローカルGC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GC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GC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] 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.

[7] 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ースプロセッサ", 並列処理シンポジウム予稿集, 情報処理学会, 1989年2月.

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