5J-2

大域アドレス空間における GC とリージョンによるメモリ管理

藤田 晃史 † 田浦 健次朗 ‡ 近山 隆 ††
† 東京大学 工学部 電子情報工学科 ‡ 東京大学大学院 情報理工学系研究科 †† 東京大学大学院 工学系研究科

1 はじめに

1.1 背景

分散環境でプログラムを記述する手法として最も普及しているのは、通信を明示的に記述するメッセージパッシングモデルであるが、必要なデータやその所持者が動的に決定されるようなアプリケーションを記述するのは難しい。そこで、分散して存在するデータに一意の仮想的なアドレスを与えて大域アドレス空間を構成し、その読み書きという形でプログラムを記述する処理系が登場している [1,2]。

大域アドレス空間を用いるプログラムでは、大域アドレス空間上でメモリ領域の確保・解放を行うことになるが、逐次プログラムの場合と同様に解放のタイミングは誤りやすく、バグの原因となる。そこで、オブジェクトの参照関係を解析して不要なメモリ領域を自動で解放する、ガーベジコレクション (GC) の機能があると便利である。しかし、分散環境で GC を行うと、ノードをまたぐ参照をたどるのにノード間通信が発生し、GC にかかる時間が長くなる。これを高速化するには、ノード間通信の発生を抑える手法が必要である。

1.2 関連研究

分散環境の GC において通信を抑え、高速化する手法は既に研究されている [3]。これらの研究はオブジェクトは与えられたものとして、その参照関係を記述・解析する方法に注目している。

一方、オブジェクトの数自体を減らすことでも GC の実行時間を削減できると考えられる。これを実現する手法として、複数のオブジェクトのメモリ空間をまとめて確保・解放するリージョン [4] を用いる方法がある。本来、リージョンはメモリの確保・解放の高速化ために考案されたものだが、これは GC の高速化にも応用できる。GC ではオブジェクトの参照関係の解析と不要なメモリの解放を行うが、複数のオブジェクトをまとめてリージョンとすると、解析対象となる参照関係の数とメモリの解放回数が減り、その分 GC の実行時間が短くなると考えられる。だが、これを大域アドレス空間上の GC に応用した既存研究は無い。

1.3 本研究の貢献

本稿ではリージョンによるメモリ管理を利用し、高速に大域アドレス空間上の GC を行う手法を提案する。そしてこの手法を大域アドレス空間処理系の一つである DMI [2] 上にライブラリとして実装し、N 体問題を解くプログラムによって性能評価を行い、手法の有効性を示す。

2 DMI

DMI [2] は原によって開発された大域アドレス空間処理系である。DMI の大きな特徴は動的なノードの参加・脱退のサポートを標榜している点であるが、本研究ではその機能をサポートしない。他に大域アドレス空間に対するキャッシュ機能や、大域アドレス空間上のメモリ領域のオーナーを移動する機能などを備えているが、本研究では簡単のためにそのような機能は使用しないものとしている。

DMI は C ライブラリとして実装されており、プログラム中から DMI の API を呼ぶという形でその機能を利用する。例えば、DMI_mmap という API によって大域的に共有されるメモリ空間を確保し、その大域アドレスを得る。DMI_read,DMI_write という API によって指定した大域アドレスのデータを読み書きする。その他に mutex、バリア、条件変数など、並列プログラミングに必要となる機能を一通りサポートしている。

3 提案手法

本研究の提案手法は、大域アドレス空間上の GC を高速化するために、リージョン [4] を用いることである。一般にリージョンを用いると GC の対象となるものの数が減少するために実行時間が短くなるが、これは分散環境においては特に効果を発揮すると考えられる。というのも、分散環境の GC に時間がかかる原因は参照をたどる際のノード間通信であるが、リージョンを用いれば参照をたどる操作自体を減らすことができるからである。

4 ライブラリの設計と実装

4.1 API

本研究で実装するライブラリ (DGC) は、GC とそれに対応するメモリ確保の API を DMI [2] に追加するものである。基本的な API は以下の 4 つである。

DGC_malloc　大域アドレス空間上にメモリ領域を確保
DGC_newregion　大域アドレス空間上にリージョンを作成
DGC_ralloc　指定したリージョンに属するメモリ領域を確保
DGC_mark_sweep　以上 3 種の API で確保したメモリ領域を対象に GC を実行

使用例を以下に示す。

```
global_address = DGC_malloc(size);
region_address = DGC_newregion();
global_address = DGC_ralloc(region_address, size);
DGC_mark_sweep();
```

4.2 メモリ確保時の動作とデータ構造

大域アドレス空間上のメモリ領域はどのノードからでもアクセスできるが、多くの場合はそのメモリ領域を確保したノードで使われる、すなわち局所性があると考えられる。よって、DGC_malloc や DGC_ralloc を呼び出したら、そのノード上にメモリ領域の実体を置くのが望ましい。そのため、本ライブラリでは各ノードごとにヒープとフリーリストを用意し、自ノードのヒープからメモリ領域を確保するようにする。

DGC_malloc の場合は、単に自ノードのヒープからメモリ領域を確保すればよいが、リージョンは複数のノードにまたがって存在するオブジェクト群を束ねて扱うので、ノードをまたぐデータ構造が必要となる。本ライブラリでは、図 1 に示すように、各ノードごとのヒープと、それをまとめるヘッダからなるデータ構造になっている。ヘッダは DGC_newregion を実行したノードに置かれる。DGC_newregion を実行すると図 1 のようなデータ構造が用意され、DGC_ralloc を実行すると指定したリージョンに属する自ノード用のヒープからメモリ領域が確保される。

4.3 GC のアルゴリズム

本ライブラリを用いるプログラムは C 言語で書かれることを想定しているが、C 言語で GC を行う手法としては保守的マークスイープ [5] が一般的であり、本ライブラリもこれを採用する。

マークスイープを行うためには各オブジェクトに付属するマークビットを用意する必要がある。DGC_malloc で確保するオブジェクトには、オブジェクトごとにマークビットを用

Garbage Collection and Region-Based Memory Management on Global Address Space

† Akifumi Fujita (Faculty of Engineering, the University of Tokyo)
‡ Kenjiro Taura (Graduate School of Information Science and Technology, the University of Tokyo)
†† Takashi Chikayama (Graduate School of Engineering, the University of Tokyo)

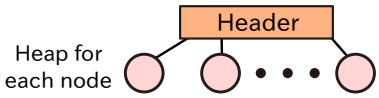

図 1　リージョンのデータ構造

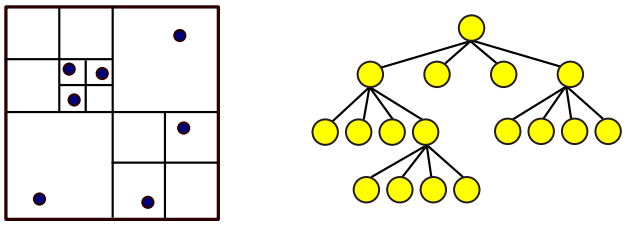

図 2　Barnes-Hut のアルゴリズムにおける空間分割 (左) と木構造 (右)

意する。リージョンについては、ヘッダと各ノード用のヒープのそれぞれにマークビットを用意している。こうすると、マークの際は各ノードに通信を行ってマークビットを立てる必要があるが、スイープ時に自ノードにあるヒープのマークビットを調べる際、ノード間通信を行う必要がなくなる。

また、保守的マークスイープではオブジェクトをマークした際にポインタが含まれているか検査を行うが、DGC_malloc や DGC_newregion の際に NOTRACE というオプションを指定すると、そのオブジェクト・リージョンは外部へのポインタを全く含まないとみなして検査を省略する。これを用いると GC をより高速に行うことができる。

5　性能評価

5.1　Barnes-Hut のアルゴリズム

通常、N 体問題は N 個の物体に対し残り N-1 個の物体との相互作用を計算するので計算量は $O(N^2)$ となるが、Barnes-Hut のアルゴリズム [6] では $O(N \log N)$ で計算できる。例えば重力 N 体問題の場合は、図 2 のように領域を再帰的に分割しておき、各領域が含む物体の質量の合計と重心を計算しておく。近傍の物体による重力を計算する際は直接計算を行うが、遠くの物体については領域の重心に全質量が存在するものとし、まとめて計算を行うことで計算量を削減する。

このアルゴリズムは、図 2 に示すように木構造を作ることで実現される。1 つのノードは 1 つの領域を表し、質量や重心などの情報と、その領域をさらに分割した領域へのポインタを持つ。粒子が移動したら木を作り直す必要があるので、各タイムステップごとに木の作成を行う。

今回は Barnes-Hut のアルゴリズムを分散環境で実行するので、木構造を複数のノードに分散させて保持する。

5.2　GC の動作

Barnes-Hut のアルゴリズムにおいて GC を実行する場合、図 2 の木構造が GC の対象となる。粒子の場所など、他のデータは静的に確保しておくことができるからである。

リージョンを使わない場合は図 2 に示した木構造の参照をすべてたどってマークする必要がある。本ライブラリでは GC は並列に実行するが、自ノードがマークしたオブジェクトが参照するオブジェクトは自ノードがマークするようになっている。よって、木の根をマークしたノードが木構造全体のマークを行うことになり、ノード間通信が多く発生して GC の実行時間が長くなると考えられる。

リージョンを使う場合は各タイムステップで作成する木構造 1 つを 1 個のリージョンとし、リージョンに属する各ノードごとのヒープに木構造のデータを分散させて格納する。リージョンをマークする際は、リージョン内のオブジェクトのデータ構造に関わらず、図 1 に示したヘッダとヒープにマークを行えばよい。これは図 2 の木構造のノードをそれぞれマークするのに対してはるかに処理が少なく、GC の実行時間が短くなると考えられる。

また、リージョン内部にはリージョン外を指すポインタは無いので、NOTRACE オプションを使用できる。これを使用するとリージョンのヒープからポインタを探し、その参照先をマークするという処理を省略できるので、さらに GC の実行時間を削減できる。リージョンを使わない場合は、図 2 に示すように各オブジェクトは他のオブジェクトへのポインタを持つので、NOTRACE オプションを使用することができない。

表 1　GC の実行時間 (秒)

ノード数	1	2	4	8	16
w/o Region	0.795	21.131	30.131	36.119	40.867
w Region	1.176	1.505	1.572	1.994	3.093
NOTRACE	0.001	0.005	0.006	0.010	0.014

5.3　実験環境

実験は Intrigger プラットフォーム† の hosei ノード (Xeon E5530 (2.4GHz) × 2 CPU × 22 Node, 10GbE) で行った。$N = 16384$ の N 体問題を Barnes-Hut のアルゴリズムで解き、各タイムステップの最後に GC を実行する。11-20 タイムステップ目の GC の実行時間の平均を記録した。一回 GC を実行するたびに木構造を 1 つマークし、1 つスイープしている。

5.4　実験結果と考察

実験結果を表 1 に示す。表中の w/o Region は木構造の作成に DGC_malloc を使った場合、w Region はリージョンを使った場合、NOTRACE は NOTRACE オプションをつけたリージョンを使った場合の GC の実行時間である。

リージョンを使うことで GC の実行時間が減少し、NOTRACE オプションによってさらに減少していることから、提案手法の有効性が確認できる。もっとも、このプログラムでは非常に大きいリージョンを 1 種類使用するという単純なもので、リージョンによる高速化の効果が現れやすいと考えられる。また、木構造のマークは逐次的に行うようになっているが、これを並列化することでリージョンを使わない場合の GC が高速になると考えられる。

6　おわりに

本稿では、大域アドレス空間における GC をリージョンを用いることで高速化する手法を提案し、N 体問題を解くアプリケーションによってその有効性を示した。しかし、GC のアルゴリズムや評価に用いたプログラムは非常に単純なものだったので、より複雑な場合において提案手法が有効か調査する必要がある。

謝辞　本研究の一部は文部科学省科学研究費補助金特定領域研究「情報爆発に対応する新 IT 基盤研究プラットフォームの構築」の助成を得て行われた。

参考文献

[1] Berkeley Unified Parallel C Project. http://upc.lbl.gov/.
[2] 原健太朗. 再構成可能な高性能並列計算のための PGAS プログラミング処理系. Master's thesis, 東京大学, 2011.
[3] D. Plainfosse and M. Shapiro. Survey of distributed garbage collection techniques. Technical report, 1994.
[4] David Gay and Alex Aiken. Memory management with explicit regions. *SIGPLAN Not.*, Vol. 33, pp. 313–323, May 1998.
[5] Hans-Juergen Boehm and Mark Weiser. Garbage collection in an uncooperative environment. *Softw. Pract. Exper.*, Vol. 18, pp. 807–820, September 1988.
[6] Josh Barnes and Piet Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, Vol. 324, No. 6096, pp. 446–449, December 1986.

† https://www.logos.ic.i.u-tokyo.ac.jp/intrigger/