グローバルガーベッジコレクションとその評価に関する考察

前田 宗則　小中 裕喜　石川 裕　友清 孝志　堀 敦史　関 進
技術研究組合 新情報処理開発機構 つくば研究センタ

本稿では，グローバルガーベッジコレクションアルゴリズムを評価するためのシミュレーションモデルを提案し，その予備評価を報告する．我々がこれまでに提案したGleanerアルゴリズムでは，サイクリックガーベッジの生成頻度やそれに含まれるオブジェクト数によって効率が大きく変化する．そこで，サイクリックガーベッジがどの程度生成されるかをシミュレーションによって予備評価した．シミュレーションでは，OO1ベンチマークモデルで採用されている参照の局所性を意識したランダムグラフを利用している．シミュレーション結果より，オブジェクト数nのサイクリックガーベッジの出現頻度$C(n)$は，$C(n) \sim n^{-\tau}$となることが分かった．これよりGleanerの戦略として，i) ローカルガーベッジコレクションを用いて着色範囲を減少させること，ii) 非常に大きな構造の塗りつぶしは放棄または中断したほうがよいことが導かれる．

A Preliminary Simulation Model of Global Garbage Collection

Munenori Maeda, Hiroki Konaka, Yutaka Ishikawa, Takashi Tomokiyo,
Atsushi Hori, Susumu Seki
Tsukuba Research Center, Real World Computing Partnership
16F Mitsui Bldg., 1-6-1 Takezono, Tsukuba-shi, Ibaraki, 305, JAPAN

In this paper, we presents a simulation model for global garbage collection algorithms and some simulation results. The efficiency of the Gleaner algorithm, we have presented so far, depends on how frequently cyclic garbage with inter-processor references is generated and how many objects are found in it. We simulate the cyclic garbage creation on a randomly connected graph based on OO1 benchmark model. The result shows an occurrence of cyclic garbage of n objects is proportional to $n^{-\tau}$. Thus, we may adopt two strategies to make less objects be reddened: i)incorporating with local garbage collector, and ii)discontinuing or pending when the number of reddened objects become larger.

1 はじめに

超並列マシンやワークステーションクラスタといったマルチコンピュータ上での高速，複雑，大規模な並列プログラムのための記述言語は，実行効率を重視しつつかつプログラムの保守や読みやすさからモジュラープログラミングを支援し，さらに安全なものであることが望まれる．自動メモリ管理は，ユーザーから繁雑で危険なメモリの獲得と解放という処理を隠蔽することができるのでこうした超並列言語の要請に答えることができる．

我々は，ゴミと疑わしきオブジェクトからマーキングを行ない，マーキングによって得られたオブジェクトの閉包が，求められた閉包に属さないオブジェクトによって参照されていないことを確認することでガーベッジを回収するグローバルガーベッジコレクションアルゴリズム [5–8] を提案している．Gleaner と呼ばれるこのアルゴリズムは，アクセス不能なオブジェクトを辿る点で従来のトレーシングガーベッジコレクションとは異なり，サイクリックなリモート参照を含むゴミオブジェクトを回収することができる．

さて，新しいガーベッジコレクションアルゴリズムについてどのように評価を行なうかという問題がある．これまでのガーベッジコレクションの研究においては，その評価は主にアプリケーションを規定した上で行なわれてきた．これは，ユーザープログラムの振舞いは多様であってシミュレーションそのものがあまり意味を持たなかったということである．しかしながら，GC に関するベンチマークプログラムの不在は，客観的な測定結果を提示できないというジレンマを持つ．Gleaner アルゴリズムにおいては，サイクリックなリモート参照を含むゴミがどの程度生成するかによって効率が変化する．サイクリックな生成がほとんど存在しない場合には，リモート参照のトレーシングは十分に長い時間遅延させることができる．また，サイクルをトレーシングした結果得られた閉包のオブジェクト数が小さければトレーシングがより少ないメッセージで行なえることが期待される．

我々は，ある手続きに基づいてある規模のオブジェクトからなるグラフを自動に生成し，その上で GC アルゴリズムをシミュレートすることで評価することを試みている．ユーザープログラムの振舞いを単純な処理で置き換えてしまうため特定の大きな構造が生じる可能性は少ないが，どの程度の規模のガーベッジが生じるかということを調べられる．

本稿の構成を明らかにする．まず，第 2 節で Gleaner アルゴリズムのアウトラインを述べる．ここではサイクル構造がどのように取り扱われるかを述べる．次に，第 3 節で，サイクルグラフの自動生成と生成されたサイクリックガーベッジグラフの持つノード数分布に関して議論する．これを踏まえて，第 4 節では Gleaner アルゴリズムの効率を改善する．第 5 節は今後の方向について触れる．

2 Gleaner アルゴリズム

Gleaner アルゴリズムでは，各プロセッサごとに輸入されたリモートポインタを記録する外部参照表 (ERT) と輸出されたローカルポインタを記録するオブジェクトアドレス表 (ODT) を用意する．ローカルガーベッジコレクションは，ルートと ODT からトレースを行なうことで常に実施できる．また，ODT と ERT の各エントリは，オブジェクトの参照数を保持しており，ポインタの輸出に当たっては，重み付き参照カウント方式 [1] と類似の手続きを踏む．各プロセッサの ERT のエントリは他のプロセッサの ODT のエントリを指すことで，サイクリックな参照が生じうる（図 1）．

Gleaner アルゴリズムは，ローカルガーベッジコレクションでは取り除けなかったプロセッサ間を跨ぐリモート参照によって構成されたサイクリックなガーベッジを見つけて取り除くものである．図 2はその大まかな振舞いを示している．Gleaner アルゴリズムでは，オブジェクトは通常のローカルガーベッジコレクション用の色（BW ビット）に加えて RG ビットと呼ばれる 1 ビットの色を保持することが要請される．図 2-(a) は初期状態を表している．全てのオブジェクト，ODT や ERT の各エントリの RG ビットは緑 (g) である．さて，図 2-(b) に移って，+で示されたオブジェクトを起点として，そのオブジェクトから到達可能なすべてのオブジェクト，ODT，ERT のエントリを赤 (r) に塗る．これは RG ビットを反転させることに対応する．

Gleaner アルゴリズムでは，最後まで赤で残されたオブジェクトをゴミであるとして回収するので，図 2-(c) の時点での重要な仕事は，生きているオブジェクトを緑 (g) に塗る直すことである．緑への塗り直しの起点となるのは，i)

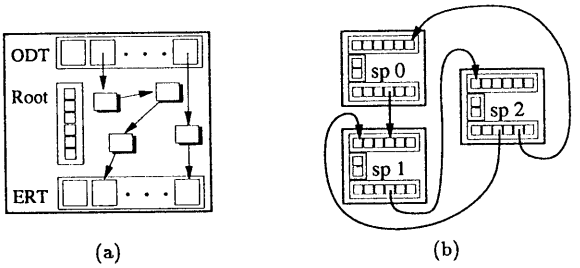

図 1: (a) 局所メモリ構造，(b) リモート参照によるネットワーク

緑の ODT エントリから指されている，または，各プロセッサでローカルにルートから到達可能な赤のオブジェクトと ii) 全ての対応する ERT エントリが見つからなかった赤の ODT エントリである．図 2-(c) は ii) のケースであり，その ODT から緑に塗られる．

図 2-(c) の処理が終了した時点では赤で残されたオブジェクトはゴミである．図 2-(d) でこうした赤オブジェクトはすべて回収される．

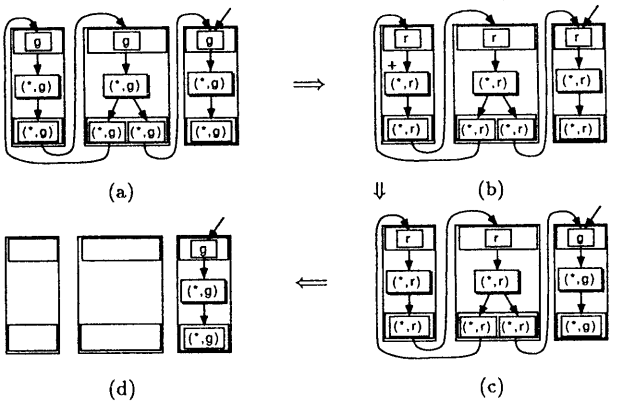

図 2: Gleaner によるサイクリックガーベッジの除去

経験的にガーベッジは局所的に存在している傾向がある．多くのガーベッジは，数プロセッサやあるいはプロセッサのクラスタといった単位の比較的狭い範囲に散らばっていることが多い（図 3）．Gleaner アルゴリズムは，そういった場合にトレースメッセージ数の減少や，トラフィックの局所化という利点が生じる．一方，ガーベッジが広範囲にばらまかれているならば，他のグローバルガーベッジコレクションよりも効率が悪化する．これは，色の塗り直し処理が余分なオーバーヘッドを生じること，またマーキングやスウィーピングのフェーズの終了に大域的な同期を必要とするからである．

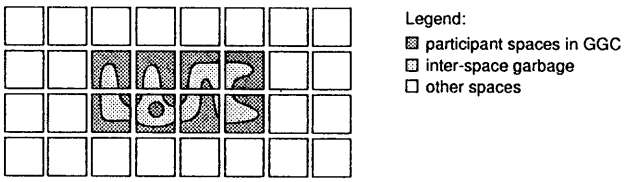

図 3: プロセッサクラスタに対する Gleaner アルゴリズムの適用

3 グラフの生成とサイクリックガーベッジの検出方式

3.1 OO1 に基づくグラフ生成と mutator モデル

エンジニアリングデータベースシステムを評価するために提案された OO1 ベンチマーク [2,3] のグラフの自動生成手法を取り上げる．OO1 ベンチマークでは，オブジェクトは定数個 c のスロットを持ち，そこに他のオブジェクトへの参照を格納する．参照先オブジェクトの決定は，乱数を用いて行なわれる．

すべてのオブジェクトはオブジェクト空間に配置される．物理的なアドレス空間を意識し，オブジェクト空間には 1 次元のインデックスが張られている．オブジェクトと参照の接続は，参照の局所性を考慮したものとなっている．90%の参照は，近接する直径 Rの距離にあるオブジェクトからランダムに選択されたオブジェクトに接続される．残りの 10%の参照は，全オブジェクトからランダムに選択されたオブジェクトに接続される．各オブジェクトの近接度は，アドレスに対応するオブジェクト id の近接度によって与えられる．

```
 1 for part in [1..N]
 2   for connection in [1..c]
 3     if rand[1..10] > 1 then
 4       let cpart = part + rand[1..R] - 1 - R/2;
 5       if cpart < R/2 then let cpart = cpart + R/2; /* 折り返し */
 6       if cpart > N - R/2 then let cpart = cpart - R/2; /* 同上 */
 7       connection(part, cpart) /* part --> cpart */
 8     else
 9       let cpart = rand[1..N]
10       connection(part, cpart)
11     fi
12   next
13 next
```

参照の局所性を規定するマジックナンバーである 90%は，実際，1%以上 99%未満の範囲であれば，ほぼ同じ結果が得られることが報告されている．

ルートに対応しオブジェクト空間の中のいくつかのオブジェクトは，他のオブジェクトと区別される．これらのオブジェクトは常に生きている．

OO1 はオブジェクト指向データベースのベンチマークとして提案されたものであるので，オブジェクト間の接続は定義されているが，オブジェクト間の接続の切断については言及されていない．一方，ガーベッジコレクションアルゴリズムにおいては，mutator がどのようにオブジェクト間の接続を切断するかが重要な問題となる．ここでは，以下のような非常に単純化された振舞いを行なうものと定義する．

mutator は，任意の生きているオブジェクトを起点として，そのオブジェクトに接続されたオブジェクトをトラバースする．

- トラバースにおいて訪問したオブジェクトの各スロットを確率 p で 0 に初期化（参照を消去）する．

- オブジェクトを訪問するたびに内部のカウンタを1減らす．カウンタが0ならばトラバースを終了する．

3.2 サイクリックガーベッジの検出方式

Cyclic Reference Counting[4](CRC) のアルゴリズムに従い，サイクリックガーベッジの検出を行なう．CRC の要請によって，オブジェクトには，2bit 色と参照カウントが与えられる．

ガーベッジの分離: オブジェクト空間を通常の mark-sweep アルゴリズムに従ってガーベッジコレクションする．最初全てのオブジェクトは白に塗られる．ルートから到達可能なオブジェクトは黒に塗られる．トレーシングが終了したときに白に残されたすべてのゴミオブジェクトを集めて，サイクル判定用のプールにそれらへの参照を格納する．

CRC マーキング: サイクル判定用のプールから参照を1つ取り出す．そのオブジェクトが黒でなければ赤にする．その赤オブジェクトを局所ルートとして，到達可能な全ての黒でないオブジェクトを赤にし，到達する度にその参照カウントを1減らす．

CRC 再マーキング: 前のステップで赤く塗られたオブジェクトのうち，参照カウントが0でないものを洩れなくピックアップする．このオブジェクトを緑に塗る．このオブジェクトから到達可能な全ての赤オブジェクトを緑にし，到達する度にその参照カウントを1増やす．

サイクルガーベッジの検出: サイクル判定用のプールから指されたゴミオブジェクトのうち，赤に塗られたまま残されたオブジェクトはサイクルを構成したオブジェクトである．このときのオブジェクト数をカウントする．赤のオブジェクトをすべてサイクル判定用のプールから取り除き，CRC マーキングからの処理を繰り返す．

3.3 シミュレーション結果

サイクリックなガーベッジオブジェクトのノード数がどういった分布をなすかをシミュレーションによって求めてみる．シミュレーションのパラメータとしては，a) 全オブジェクト数，b) オブジェクトのスロット数，c) ルートオブジェクト数，d) ミューテーション確率，e)mutator トラバースカウント，f) オブジェクトの近接距離，g) シミュレーション回数といったものが考えられる．本稿では，これらのうち，c) と f) を変更した場合の振舞いの違いについて報告する．

オブジェクト数100,000，スロット数4，ルートオブジェクト数を4, 16, 64 と変更し，ミューテーション確率0.5，トラバースカウント1,000,000，近接距離1,000に従つシミュレーションを testA(4), testA(16), testA(64) とする．繰り返し回数は1,000とした．図4に testA(4) におけるサイクリッククラスタの分布を示す．グラフは，縦軸が出現頻度，横軸がオブジェクト数である．縦横軸ともに対数スケールでプロットしてある．

testA(4) における対数プロットで Pearson の相関係数を求めた結果を表1に示す．これより，ノード数 n のサイクリックガーベッジの出現頻度 $C(n)$ は以下のように近似できる．

$$C(n) \sim n^{-\tau} \qquad \tau = 1.5 \tag{1}$$

testA(4), testA(16), testA(64) を比較した時，特に有為な差が認められない．ルートオブジェクト数は，生きているオブジェクトの割合を増加させるが，サイクリックガーベッジのオブジェクト数には影響を与えないようである．これは，次のシミュレーション testB(4), testB(16), testB(64) にも認められる．

次に近接距離を100に変更したシミュレーション testB を実施する．その他の条件は，testA と全く同様である．testB では，$\tau = 2.1$ となりノード数の分布が比較的早く減衰する．

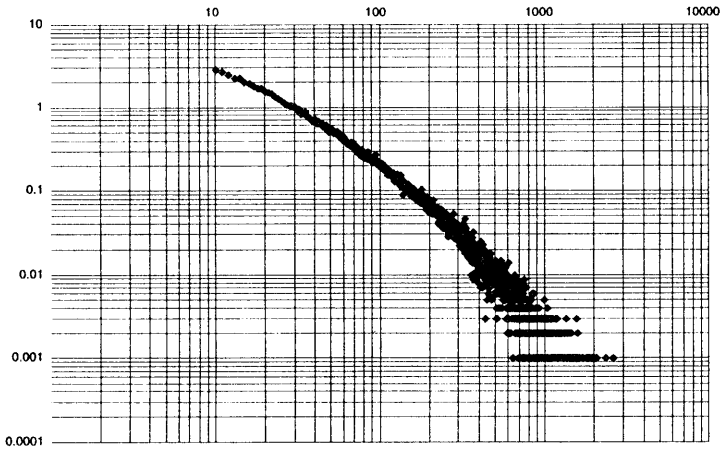

図 4: サイクリックガーベッジのオブジェクト数分布: testA(4)

表 1: 相関係数と傾き

	相関係数	傾き
testA(4)	-0.98924731	-1.4848902
testA(16)	-0.988162	-1.4788362
testA(64)	-0.9874045	-1.4975164
testB(4)	-0.979363235	-2.108050822
testB(16)	-0.975913254	-2.127649574
testB(64)	-0.97705964	-2.130133621

式 1に示されるようにシミュレーション中にかなり大きなオブジェクト数を持つサイクリックガーベッジが検出されうる．表 2によれば，ノード数の小さいもの，すなわち出現多い順にまとめた時に全体の半数の出現におけるノード数の平均は，11.5 程度であるが，一方 100%までの平均は 72.3 程度まで上昇する．test(A) におけるノード数の分布は比較的ゆっくりと減衰する傾向にあることがわかる．

表 2: 出現頻度と平均オブジェクト数

	50%	90%	100%
testA(4)	11.54989023	38.67675837	72.28567827
testB(4)	7.3593368	19.92552318	33.07360396

4 考察

オブジェクト数 n につき n^{-r} に比例した頻度でサイクリックガーベッジが生成されるとしたならば，大きなオブジェクト数のガーベッジもまれに生成されることを意味する．また，大きなオブジェクトの集合は，生きているオブジェクトを指している参照も多く存在していると思われるので，サイクリックガーベッジに属するオブジェクトが最初にピックアップされたとしても，そこから赤色が全オブジェクトに波及する可能性もある．

Gleaner アルゴリズムは，グローバル GC の実行終了のために，赤色が波及した全てのプロセッサで少なくとも一度はローカルガーベッジコレクションを実施しなければならない [7]．このときローカルガーベッジコレクションは赤色から緑色への着色のために用いられているわけであるが，生きているオブジェクトが無駄に赤色に塗られるのを防ぐこともできる．ナイーブには，赤色を塗るとき，同時にローカルガーベッジコレクションを実施する．まず Root から初めて黒に塗る．黒に塗られたオブジェクトはこのローカルガーベッジコレクションの間に生きていることが保証されるので赤に塗る必要はない．Root から到達可能なすべてのオブジェクトを黒に塗った後，赤の着色を行なう．その処理が終了したならば，ODT から到達可能なオブジェクトを全て黒に塗る．これによって，無駄に赤に塗られることをかなり防げると思われる．

より積極的には，黒白 (BW) のビットを 2 ビットに拡張する．これは過去に行なわれたローカルガーベッジコレクションの情報を保持しておくためである．BW ビットに新しく加えられたビットは，前のガーベッジコレクションにおいてルートから到達可能であったかどうかを保持する．こうすることで，着色範囲を最小化するためには，ガーベッジコレクションを行なわなければならないと制約を緩めることができる．過去いつ GC したのかというタイム情報をどこかにおいておけば，その時点でルートから到達不能であったオブジェクトのみ赤にぬることになる．

また，Gleaner アルゴリズムは，従来のトレーシングガーベッジコレクションと異なり，赤への着色自体は任意の時点で放棄してもよい．これは，次のフェーズで，生きていることが確認されたオブジェクトは必ず緑に戻されるからである．着色の範囲にあるプロセッサ数が非常に大きく増加し始めたら，どこか生きているオブジェクトから塗っていると判断してその着色は放棄した方がよい．もしオブジェクトの近接距離が予め見積もられているならば，どの程度のサイクリックガーベッジまで存在しやすいかによって，放棄する閾値となるオブジェクト数を決められる可能性がある．

5 おわりに

本稿では Gleaner アルゴリズムを評価する準備として，あるサイズのサイクリックガーベッジがどの程度出現するかをシミュレーションによって求めた．この結果，緩い局所性を持つグラフにおいては，比較的大きなサイズのサイクリックガーベッジが生じることがあることが示された．

こういった局所性を持つランダムなグラフが持つ性質はまだ多くのことが調べられていない．本稿でも一部のパラメータについてシミュレーションを実施しただけである．今後，残りのパラメータについて引続きシミュレーションを実施してみる予定である．

参考文献

[1] D. I. Bevan. Distributed garbage collection using reference counting. In *Lecture Notes in Computer Science*, Vol. 259, pp. 176–187. Springer-Verlag, 1987.

[2] R. G. G. Cattell and J. Skeen. Engineering database benchmark. Technical report, Database Engineering Group, Sun Microsystems, 1990.

[3] R. G. G. Cattell and J. Skeen. Object operations benchmark. *ACM Transactions on Database Systems*, Vol. 17, No. 1, pp. 1–31, March 1992.

[4] Rafael D. Lins. Cyclic reference counting with lazy mark-scan. *Information Processing Letters*, Vol. 44, No. 4, pp. 215–220, 1992.

[5] Munenori Maeda, Hiroki Konaka, Yutaka Ishikawa, Takashi Tomokiyo, and Atsushi Hori. An incremental global garbage collection to reclaim inter-space cycles of garbage. Technical Report TR–94016, RWCP, August 1994.

[6] Munenori Maeda, Hiroki Konaka, Yutaka Ishikawa, Takashi Tomokiyo, and Atsushi Hori. On-the-fly global garbage collection in a distributed environment. In *JSSST 11th Annual Conference Proceedings*, pp. 289–292. Japan Society for Software Science and Technology, October 1994.

[7] Munenori Maeda, Hiroki Konaka, Yutaka Ishikawa, Takashi Tomokiyo, Atsushi Hori, and Jörg Nolte. On-the-fly global garbage collection based on partly mark-sweep. In *Lecture Notes in Computer Science*. Springer-Verlag, September 1995. (To appear in 1995 International Workshop on Memory Management).

[8] 前田宗則, 小中裕喜, 石川裕, 友清孝志, 堀敦史. Incremental cyclic garbage collection for multi-computers. 情報処理学会第 49 回全国大会論文集, 第 4 巻, pp. 163–164, September 1994.