分散メモリ並列計算機上での参照カウントと分散マーキングの実装および性能比較

山本泰宇[†,††] 田浦健次朗[†] 米澤明憲[†]

本論文では，分散メモリ計算機上での動的なメモリ管理手法である，参照カウント方式と分散マーキング方式を比較し，両者の得失を解析および実験によって示す．まず単純なメモリの回収モデルを考案し，以下の性質を導いた．すなわち，(1) 参照カウント方式の効率はほぼゴミの「深さ」に比例する．具体的には，分散した 2 分木のように，深い入れ子状のデータ構造を用いるプログラムでは，参照カウントの性能は大幅に悪化する．(2) 十分に大きなヒープが与えられる場合，分散マーキング方式の費用は，多くの追跡型のゴミ集めと同様，ヒープが大きくなるにつれて減少する．その一方，参照カウント方式の効率はヒープの大きさによらない項を持ち，ヒープを大きくしてもある一定以上には向上しない．さらに両方式に基づく分散ゴミ集めを実装し，3 つの並列プログラムを用いた実験の結果を通して両者の得失を確認した．

Comparing Reference Counting and Distributed Marking on Distributed-memory Parallel Computers

Hirotaka Yamamoto,[†] Kenjiro Taura[†] and Akinori Yonezawa[†]

We compare two common schemes of dynamic memory management on distributed-memory parallel computers, namely *reference counting* and *distributed marking*. We present a simple performance model to analyze the two schemes, and draw two important observations: 1) the performance of reference counting is proportional to the "depth" of garbage; specifically, it suffers severely when application programs use deeply nested data structures such as distributed trees. 2) Given sufficiently large heaps, the cost of distributed marking decreases as the heap size increases, as is the case in most tracing collectors. On the other hand, it is shown that increasing the heap size does not improve performance of reference counting as much, since its cost has a constant term against the heap size. By implementing distributed garbage collectors based on the two schemes, we confirmed our observations using three parallel applications.

1. はじめに

分散メモリ並列計算機を対象にした並列言語では，他のプロセッサ上にあるオブジェクトを参照することがある．この**遠隔参照**が存在する場合，オブジェクトがゴミとなったかどうかはオブジェクトのあるプロセッサ内の情報だけでは判断できない．そこで，分散メモリ並列計算機上でのゴミ集め（GC）は，あるオブジェクトがまだ他のプロセッサから参照されているかどうか調べる手段を必要とする．この手段を大別すると，参照カウント方式と追跡型の方式とに分けられる．

参照カウント 参照カウント方式は，各オブジェクトがいくつのプロセッサから遠隔参照されているかを数えておくものである．新たな遠隔参照を作る際，また遠隔参照を他のプロセッサにコピーする場合はカウンタを 1 つ増やす．オブジェクトを遠隔参照しているプロセッサが，もはやそのオブジェクトを参照していないことが分かると，カウンタを 1 つ減らす．このようにすると，カウンタが 0 であるオブジェクトは，オブジェクトの存在するプロセッサのルートから，到達可能でなければ回収可能と判別できる．

追跡方式 参照カウント方式以外の分散 GC の方式としては，プロセッサ間をまたがって遠隔参照をたどって生きているオブジェクトを検出する，追跡型の方式がある．追跡型の分散 GC としては，ヒープ管理方式によってコピー型とマーク型が考えられる．本論文における参照カウント方式との

† 東京大学理学部
Faculty of Science, University of Tokyo
†† 日本学術振興会特別研究員
JSPS Research Fellow

比較という目的からは，遠隔参照の追跡が重要であるので，以降では簡便のため，マーク型を前提に議論する．分散マーキングは，1 台のプロセッサで行うマーク&スイープ GC をプロセッサ間で行うように拡張したものである[1),2)]．各プロセッサは各々のルート（レジスタやスタック）から生きているオブジェクトをたどり，遠隔参照しているオブジェクトにたどり着くと，そのオブジェクトの存在するプロセッサに**マークメッセージ**を送る．マークメッセージを受け取ったプロセッサはそのオブジェクトから到達可能なオブジェクトもたどる．これを繰り返して，ルートから到達可能なオブジェクトをすべてたどり，たどられなかったオブジェクトをゴミとして回収することができる．

既存の研究ではしばしば，プロセッサ間の同期を必要としない参照カウントの方が，分散マーキングよりも優れていると主張されてきた．しかしそれらは後に 5 章で述べるように，分散 GC の方式の定性的な性質のみに注目しており，同量のゴミを回収するのにかかる仕事量の違いを明かにしてはいない．我々は参照カウントと分散マーキングの効率の違いを，GC の効率（回収量/仕事量）に注目して，単純なモデル上で考察した．さらにその考察を検証するため，並列オブジェクト指向言語 ABCL/f[3)] 上に参照カウントと分散マーキングの両方式を実装し，3 つの並列アプリケーションを用いた実験を行った．

以降では，2 章で我々の分散 GC の実装および特徴を説明し，3 章で参照カウントと分散マーキングの効率を単純なモデル上で示す．4 章では，3 章で導いた考察を検証する実験とその結果を述べる．5 章で関連研究にふれた後，6 章で我々の主張をまとめる．

2. 両方式の実装および特徴

2.1 局所 GC と輸出表

既存の研究と同様，我々は参照カウントと分散マーキングのいずれの方式でも，大域的な情報を用いない GC である**局所 GC** を併用している．オブジェクトの参照がはじめて他のプロセッサに渡されたときに，そのオブジェクトはそのプロセッサの**輸出表**と呼ばれる表に登録される．局所 GC は輸出表に登録されているオブジェクトは必ず生きているものとして，回収しない．すなわち，ゴミのうち，輸出表から到達不能であるものだけが回収される．

プロセッサ P が遠隔にあるオブジェクト O への参照を受け取ると，P 内に O に対する**スタブ**（proxy）オブジェクトが作られる．P から O への遠隔参照は実際にはこのスタブオブジェクトに対する参照である．各プロセッサはそのプロセッサ内のスタブオブジェクトを管理する表（**輸入表**）を持っている．

局所 GC を併用する理由は，参照カウントの場合，局所的なオブジェクトの参照の管理をも参照カウントで行うのは，大きなオーバヘッドになるためである．分散マーキングの場合は，大域的なマーキングを行わなくても回収できるゴミがあるうちは，同期等のコストのかからない局所 GC を行う方が良い場合が多いという観測に基づいている．

今回我々が行った実験では，参照カウント・分散マーキングともに，局所 GC は全プロセッサがいっせいに行うようにしている．1 つのプロセッサが，GC の必要性を検出すると，あらかじめ合意されたマスタープロセッサにメッセージを送り，マスタープロセッサが全プロセッサに GC を開始するメッセージを送る．複数のプロセッサがほぼ同時にマスタープロセッサにメッセージを送った場合，マスタープロセッサはそれらすべてに対して 1 度だけ GC 開始メッセージを送る．

局所 GC を各プロセッサで独立して起こすようにすると，同期的な通信を行うアプリケーションでは，通信相手が GC 中である確率が高くなり，通信待ちの時間が大きくなり，結果的に性能が悪化する[4),5)]．一方，全プロセッサがいっせいに GC を行うことで不必要な GC を行う可能性もあるが，本論文で扱うアプリケーションにおいては，問題となるような性能の低下は存在しないことを確認している．

局所 GC には，Boehm らによる既存の 1 プロセッサ用の保守的 GC ライブラリ[6),7)] を変更して用いている．この GC ライブラリはマーク&スイープ方式によるメモリ管理を行う．GC ライブラリは通常徐々にヒープサイズを増加させていくが，ゴミ集めの効率の比較という点においてこれは挙動を分かりにくくする．そこで，後に 4.2 節で説明するように，ヒープサイズをある大きさに固定し，以降の拡張を禁止して実験を行った．

2.2 参照カウント

我々が実装した参照カウントは，**重み付き参照カウント**[8),9)] と呼ばれるアルゴリズムに基いている．重み付き参照カウントでは，遠隔参照を作る際，オブジェクトを持つプロセッサから重さ w (> 0) を一緒に渡す．遠隔参照を持つプロセッサが，他のプロセッサに遠隔参照をコピーする場合は，自分の持つ重さの一部を分け与える．

あるプロセッサが局所 GC を起こし，その結果遠隔参照していたオブジェクトをすでに参照していない

ことが分かると，現在保持しているそのオブジェクトに対応する重さを，オブジェクトの存在するプロセッサに返却する．重さを返却するためのメッセージ（**デリート メッセージ**）は GC の間は蓄積しておき，GC 終了後に各プロセッサ宛てのメッセージをまとめて送る．参照を渡したプロセッサは，他のプロセッサに渡した重さの合計を記録している．デリート メッセージで重さが返却されて，渡した重さと返却された重さが等しくなったオブジェクトは，輸出表から外される．

一度他のプロセッサに参照を渡したオブジェクトでも，ゴミとなった後にこうして重さが返却され，重さが 0 になると，次の GC で回収することができる．

2.3 分散マーキング

我々の分散マーキングは，開始する際にすべてのプロセッサで同期をとり，各プロセッサのルートからマークを始める．すべてのプロセッサがルートから，そのプロセッサ内でたどれるオブジェクトをマークし終えるまではマークメッセージは蓄えておき，マークし終えたら蓄えたマークメッセージをいっせいに送り出す．マークメッセージを受け取ったプロセッサは，対応するオブジェクトから再びマークを始める．このマークをして，蓄えたマークメッセージを送る，という操作を，全プロセッサ中のルートから到達可能なオブジェクトが，すべてマークされるまで繰り返す．

併用している局所 GC と分散マーキングのいずれを起動するかの戦略は，適応的なアルゴリズムになっている．4.2 節で述べるように，我々は実験を単純にする目的から，最初に大きさの固定されたヒープを与え，後はヒープがあふれるたびに局所 GC もしくは分散マーキングのいずれかを行っている（すなわち動的にヒープを拡張しない）．局所 GC か分散マーキングかの選択は，前回までの分散マーキングと局所 GC の費用（GC 時間/回収量）を比較し，優れている方の GC を行う．

3. 両方式の効率に対する考察

3.1 GC の効率

GC の目標は，少ない GC 時間でより多くのメモリをアプリケーションに供給することである．すなわち，

$$\text{費用} = \frac{\text{GC に要した時間}}{\text{割り当てられたメモリ量}},$$

を小さくすることが目標である．アプリケーションが十分長く実行されて，GC によって回収された領域を多数回再利用していると仮定すると，割り当てられたメモリ量は GC が回収したメモリ量と，ほぼ等しい．そこで上の式は，

$$\text{費用} = \frac{\text{GC に要した時間}}{\text{GC が回収したメモリ量}},$$

とも書ける．

比較の基準として 1 台しかプロセッサが存在しないときの単純な走査型の GC の費用 C_{local} を，ある 1 回の GC に着目して考えると，

$$C_{\text{local}} = \frac{\alpha_{\text{local}} L}{H - L},$$

となる．ただし，H はヒープの大きさ，L はその GC 時にルートから到達可能なオブジェクトの量，α_{local} は，GC がローカルメモリ上の 1 語を，走査するのに要する時間を表している．

注：上記の式は GC の中の，いわゆるマーキング処理，すなわち回収可能なメモリ領域を検出する処理の費用を表している．一方いわゆるスイープ処理，すなわち回収されたメモリを，実際に再利用可能にするための処理に要する時間は含まれていない．これにかかる時間は実際小さく，GC 時には 4 KB ごとのブロック単位で空き領域を含むブロックをつなぎなおすだけであり，より小さな単位でのフリーリストの再構築はメモリ割当て要求時に必要に応じて行われる（遅延スイープ）．いいかえれば回収されたメモリを，実際に再割当て可能にするための大部分の処理は，GC 時間には含まれていない．

この 1 台のプロセッサ上での GC の効率の議論を基に，3.2 節では参照カウントおよび参照カウントで用いる局所 GC の，3.3 節では分散マーキングおよび分散マーキングで用いる局所 GC の費用をそれぞれ論じる．議論の要点を分かりやすくするため，以降の議論においては，以下で述べる周期的なアプリケーションの挙動を仮定する．これらの仮定が及ぼす影響は，適宜考察する．

仮定 1 アプリケーションは 1 周期の間に A $(> L)$ のオブジェクトを割り当てる．その周期が終わると割り当てられたオブジェクトはすべて捨てられる．アプリケーションはこれを十分な回数（$= n$ 回）繰り返す．

仮定 2 ヒープは A よりも十分大きく，したがって割り当てられたオブジェクトはほとんどが次の GC までにゴミになる．

仮定 3 複数回の GC にわたってルートから到達可能であり続けるようなオブジェクト（長い生存期間を持つオブジェクト）はわずかである☆．

☆ もしくは，世代 GC 等を用いることで，長い生存期間を持つオブジェクトに対する仕事量はごくわずかになる．

3.2 参照カウント

3.2.1 オブジェクトの深さ

参照カウントは,(1) 局所 GC によってもはや使われていない遠隔参照を検出し,(2) その情報を他のプロセッサに伝え,(3) それによって回収可能になったオブジェクトが次の局所 GC 時に回収される,というサイクルを繰り返してゴミを回収していく.ある局所 GC で回収されるのは参照数が 0 になったゴミだけであり,そうでないゴミは参照数が 0 になるまでの間,局所 GC を生き残ることになる.以下では議論を簡単にするために,局所 GC は全プロセッサでいっせいに行われるものとする(2.1 節を参照).

例として,プロセッサ間にまたがった,長さ l のリスト構造(隣り合った 2 個のセルはつねに別のプロセッサに割り当てられているとする)がゴミになったとする.各セルを指すポインタはこのリスト内のポインタしかないものとすると,ゴミになってから最初の局所 GC で,先頭のセルだけが回収される.その結果 2 番目のセルの参照カウントが 0 になり,次の局所 GC でそのセルが回収される.一般に i 番目のセルは第 i 回目の局所 GC で回収され,それまでの $(i-1)$ 回の局所 GC ではゴミであるにもかかわらずマークされることになる.

参照カウント方式による回収の費用を左右するパラメータとして,回収されていない全オブジェクト(ゴミも含む)からなるグラフに対する各オブジェクトの**深さ**を,以下を満たす最小の数として定義する☆.

- 参照カウントが 0 のオブジェクトの深さは 0.
- 深さ i のオブジェクトから,局所参照のみをたどって到達可能なオブジェクトの深さは i 以上.
- 深さ i のオブジェクトから,遠隔参照されているオブジェクトの深さは,$i+1$ 以上.

例として,**図 1** にオブジェクトの参照関係と,各オブジェクトの深さを示した.この定義の下で,深さ i のゴミはゴミになってからも,i 回の局所 GC を生き延び,$(i+1)$ 回目の局所 GC ではじめて回収される.

3.2.2 深さと参照カウントの費用との関係

G_i を 1 周期の間に割り当てられる深さ i のオブジェクトの量とする.するとまず,$A = \sum_{i\geq 0} G_i$ であるといえる.アプリケーションは計算を n 周期繰り返すので,その間に割り当てるメモリの量は,

$$nA = n\sum_{i\geq 0} G_i \ , \tag{1}$$

☆ この規則では互いに遠隔参照しあうようなオブジェクトの深さは決められないが,便宜上 ∞ としておけばよい.

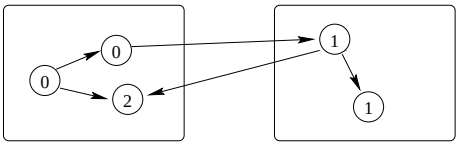


図 1 オブジェクトの深さ:四角がプロセッサの境界,丸がオブジェクト,矢印が参照関係を表す.オブジェクトの中の数字はその深さを表す.

Fig. 1 Depth of Objects: Squares, Circles, Arrows indicate processor boundaries, objects, references respectively; The number in an object is the depth of the object.

一方深さ i のゴミは,ゴミになってから回収されるまでに i 回マークされることに注意して,GC がこれらすべてのゴミを回収するために行う仕事は,

$$W_{\rm rc} = \alpha_{\rm local} mL + \alpha_{\rm local} n \sum_{i\geq 0} iG_i \ . \tag{2}$$

ただし m は n 周期の間に行われる GC の回数であり,これは以下で述べるように,ヒープの大きさおよび生きているオブジェクトの量により決定される.最初の項は生きているオブジェクトに対する仕事の量を表し,2 番目の項はゴミとなったオブジェクトに対してしなければならない仕事の量を表す☆☆.

上で m としておいた,GC の回数は次のように計算できる.深さ i のオブジェクトがゴミとなると,その後の i 回の GC を生きのび,また GC 間に割り当てられる深さ i のオブジェクトの量は $(n/m)G_i$ であるので,各 GC の直前に存在する深さ i のオブジェクトの総量 T_i は,

$$T_i = (i+1)\frac{n}{m} G_i \ . \tag{3}$$

ゆえに GC の直前に存在するすべてのゴミの量の合計 T は,

$$T = \frac{n}{m}\sum_{i\geq 0}(i+1)G_i \ . \tag{4}$$

さらに,T が $H-L$ と等しい事実から以下のように m が導ける.

$$m = \frac{(\sum_{i\geq 0} iG_i + \sum_{i\geq 0} G_i)n}{H-L} \ . \tag{5}$$

以上より参照カウントの費用は,

$$C_{\rm rc} = \frac{W_{\rm rc}}{nA} = C_{\rm local}(\bar{d}+1) + \alpha_{\rm local}\bar{d}. \tag{6}$$

ただしここで,

☆☆ 簡単のために,ここでは参照カウントがデリートメッセージを送受信するのにかかる費用は考慮していない.実際にはアプリケーションに依存する定数を参照カウントの費用($C_{\rm rc}$)に加えることになる.

$$\bar{d} = \frac{\sum_{i\geq 0} iG_i}{\sum_{i\geq 0} G_i} \ .$$

ただし，$\bar{d}$ は**ゴミの深さの重みつき平均**であり，$(\bar{d}+1)C_{\mathrm{local}}$ は生きているオブジェクトをマークするのにかかる費用，$\alpha_{\mathrm{local}}\bar{d}$ はゴミをマークするのにかかる費用となる．

式 (2) と同様，式 (6) でも最初の項は生きているオブジェクトをマークする費用を表し，2 番目の項はゴミに対する費用を表している．最初の項がヒープが大きくなるにつれ減少すること，および 2 番目の項がゴミの平均の深さ（$\bar{d}$）に比例していることは，深さ i のオブジェクトは i 回ゴミとなった後もマークされることから，直感的に理解できる．

また，最初の項も $\bar{d}$ に比例していることも興味深い．これは深いゴミが何度も局所 GC を生きのびることでヒープを圧迫するため，GC 後に割り当てられるヒープのサイズが減少してしまい，GC の回数を増やすことが原因である．

3.2.3 非同期 GC の場合

前項の議論の出発点は，

> 深さ l のゴミは，ゴミになってから回収されるまでに l 回マークされる

ということであった．この議論の仮定として，全プロセッサが「いっせいに」局所 GC を行うことを仮定したが，ここではこの仮定を緩めて，**全プロセッサが同じ周期で GC を行う**という場合の性能を考察する．すなわち，各プロセッサは 1 度ずつ，ある順番で GC を行い，全プロセッサが 1 度ずつ GC を行った後で再び各プロセッサが GC を 1 度ずつ行う，という周期を繰り返す．

今，$o_1 \to o_2 \to \cdots \to o_l$ という参照関係にあるオブジェクト群 $o_1, \cdots o_l$ がゴミになったとする（$\to$ は遠隔参照で，o_1 の参照カウントが今 0 になった）．o_i を保持するプロセッサを p_i と書く．各プロセッサが 1 度ずつ GC をおこすと，o_1 は必ず回収されるが，p_1 と p_2 がこの順序で GC をおこせば，o_1 に加えて o_2 も回収される．これは GC のおきる順番がどれも同様に確からしいと考えれば，確率 1/2 でおこると考えられる☆．同様に，p_1，p_2，p_3 がこの順序で GC をおこせば，o_1，o_2，o_3 が回収され，これは確率 $1/3! = 1/6$ でおこる．一般に，o_k が全プロセッサ 1 度ずつの GC で回収されるには，$p_1, p_2, \cdots, p_k$ がこの順序で GC をおこすことが必要でその確率は $1/k!$ である．これを基に，全プロセッサが 1 度ずつ GC を行うと回収されるオブジェクト数の平均を求めると，l によらず $e-1$（e は自然対数の底で，< 2.719）以下となることが示される（付録 A.1 参照）．

したがってこの場合も，深さ l のゴミが回収されるまでに，l に比例した回数のマークが必要であることにはかわりはない．そして，この利益は，2.1 節で述べた非同期 GC の欠点を被ったうえでのものであるので，実際にこれがそのまま性能向上となって現れるわけではない．

3.2.4 考　　察

式 (6) から以下の 2 つの重要な解釈を導ける．

- ゴミの平均の深さ（$\bar{d}$）が 0 に近い場合，$C_{\mathrm{rc}} \approx C_{\mathrm{local}}$，つまり参照カウントはほぼ単一プロセッサ上の GC と同程度に効率的である．しかし $\bar{d}$ が大きい場合，C_{rc} は単一プロセッサ上の GC の費用（C_{local}）よりもずっと大きくなる．特に，$\bar{d}$ が大きくヒープも大きい場合（それゆえ C_{local} はとても小さい）はその差は大きい．
- 参照カウントの費用は，いくらヒープを大きくしても $\alpha_{\mathrm{local}}\bar{d}$ より少なくはならない．いいかえればヒープを大きくするにつれて，費用はある定数（$\alpha_{\mathrm{local}}\bar{d}$）に近付く．

3.1 節における仮定 2 を満たさない，すなわちヒープが十分大きくない場合を考えると，生き残るオブジェクトの割合が大きいため，費用はこの C_{rc} よりも大きくなる．すなわち，ヒープサイズが大きくなるにつれて費用が減少するように見えるが，費用の下限は C_{rc} であり，それ以上には減少しない．

実際問題としては，このことがすべてのアプリケーションで問題となるわけではない．まず，多くのアプリケーションでは $\bar{d}$ は非常に小さい．なぜならば，2 分木のように深いオブジェクトが大部分を占めるデータ構造を使用する場合でも，多くの場合それに応じた量の浅いオブジェクトも生成され，結果として $\bar{d}$ が小さくなるからである．たとえば深さ n の 2 分木を作る場合，それを生成する最中にも，遠隔通信のためのチャネル等，深さが 1 や 2 にしかならないオブジェクトがそれにともなう量だけ生成される．生成された 2 分木を走査して計算を行う場合，典型的にはさらに浅いオブジェクトの割合が多くなる．

それでもなお，上のような性能モデルは，種々のアプリケーションに対する GC 方式の優劣を理解するう

☆ これは楽観的な見積りで，実際にはこれがおこるためには，p_1 の GC の**終了後**，デリートメッセージが p_2 に伝わった後で，p_2 の GC が**開始**される必要がある．すなわち両者の GC がオーバラップした場合を除外する必要があるが，簡単のため以降では考えない

えで有益である．次章以降の実験では，参照カウント方式の性能がゴミとなるオブジェクトの深さにどれだけ依存しているかを見て，分散マーキングとの比較を行う．

3.3 分散マーキング

3.3.1 分散マーキングの費用

生きているオブジェクトのみをたどるという分散マーキングの性質から，その仕事量は L に比例したものになる．そこで分散マーキングの費用は，

$$C_{\mathrm{dm}} = \frac{\alpha_{\mathrm{dm}} L}{H - L} = \frac{\alpha_{\mathrm{dm}}}{\alpha_{\mathrm{local}}} C_{\mathrm{local}} \ . \qquad (7)$$

ただし，α_{dm} $(> \alpha_{\mathrm{local}})$ は比例定数で，遠隔参照をたどるコストを反映したものになる．α_{dm} は実際には，遠隔参照のみでつながるオブジェクトが最大何段あるか等によって変化する．しかし，たとえば最大段数をパラメータとしても，生きているオブジェクトの状態は参照カウントの仕事量に（分散マーキングと比較して）大きく影響しないため，参照カウントとの比較という観点からは得られる情報はない．そこで，本論文では α_{dm} をアプリケーションごとに一定とする．さらに分散マーキングでは，プロセッサの同期のコストや負荷の偏りからくる無駄な待ち時間等，様々なオーバヘッドが加わるが，本論文ではそれらも α_{dm} の増大という形で反映されているものとする．

すると分散マーキングの費用は確かに，1 プロセッサ上での単純な GC の費用より大きいことになるが，1 プロセッサ上での GC 同様に，ヒープを大きくするとともに費用が小さくなる利点があることが分かる．

3.3.2 分散マーキングにおける局所 GC の費用

分散マーキングにおける局所 GC は深さ 0 のゴミだけを回収し，そうでないゴミは分散マーキングを行うまで，決して回収されない．3.1 節のアプリケーションの挙動の仮定から，局所 GC を m 回行っては分散マーキングを 1 回行うものとする．すると m 回の GC の仕事量は，1 回あたり，$\sum_{i>1} G_i$ ずつマークしなければいけないオブジェクトが増加するので，

$$\begin{aligned} W_{\mathrm{lgc}} &= \alpha_{\mathrm{local}} \sum_{j=1}^{m} j \sum_{i>1} G_i \\ &= \alpha_{\mathrm{local}} \frac{m(m+1)}{2} \sum_{i>1} G_i \end{aligned} \qquad (8)$$

であり☆，費用は，

$$C_{\mathrm{lgc}} = \frac{\alpha_{\mathrm{local}}(m+1) \sum_{i>1} G_i}{2G_0}. \qquad (9)$$

参照カウント方式と同様，これはヒープの大きさによらない一定の値である．また，分散マーキングを行わないまま局所 GC を繰り返すと（m が大きいと），費用は増大する．

3.3.3 考　　察

実際には，分散マーキング 1 回に対して局所 GC をおこす回数（m）はアプリケーションごとに異なり，それゆえ費用も m によって異なる．しかし，2.3 節で述べた適応的なアルゴリズムにより，最悪の場合でも費用は C_{dm} であり，しかも C_{dm} はヒープが大きくなるに従って減少する．

4. アプリケーションを用いた性能評価

3 章で行った考察が現実のアプリケーションの挙動と一致することを確認するために，並列オブジェクト指向言語 ABCL/f を共有メモリ並列計算機 Sun Ultra Enterprise 10000 上で，分散メモリ計算機を模するようにして動作させ，実験を行った．

4.1 実験環境

ABCL/f　ABCL/f [3] は並行オブジェクトと関数やメソッドの非同期呼び出しをサポートするオブジェクト指向言語である．並行オブジェクトは遠隔参照を用いて複数のプロセッサ間で共有できる．また他のプロセッサへの関数やメソッドの呼び出しにおいては**通信チャネル**と呼ばれるオブジェクトを作り，その返り値を得るために用いる．主にこの並行オブジェクトと通信チャネルがプロセッサ間にまたがるゴミとなる．

Sun Ultra Enterprise 10000　250 MHz の UltraSparc I プロセッサ × 64 個と集中共有メモリを持つ並列計算機である．この上で複数のプロセスを立ち上げ，その間に共有メモリを介したメッセージ送受信層を用意して，分散メモリ型の計算機を模して ABCL/f を動作させている．共有メモリは通信以外の用途には用いていない．以降の実験では 48 台のプロセッサのみを用いている．

4.2 実験内容

すべてのアプリケーションはまったく同じか，ほぼ同じような計算を繰り返し行う．それぞれのアプリケーションに対し，まず以下に述べる手法でそのアプリケーションが最低限必要とするメモリの量 L を求めておく．そしてそれぞれに対し，L の 2〜8 倍のヒープサイズを与えて，アプリケーションを十分長い時間繰り返し動作させ，GC にかかる時間を測定する．測

☆ ここでは，ある回の局所 GC で回収できなかったオブジェクトを，以降の局所 GC でも毎回走査しなおすものとしている．冗長なマーキングは，輸出表からたどれるオブジェクトのマークを局所 GC を続けている間保存しておくことにより，防ぐことが可能である．しかし，我々がその方式を実装したところ，マークを保存することで回収できなくなるオブジェクトの量が多く，GC の頻度が増加して効率は上がらなかった．

定は最初の何回かの繰返し終了後，GCが一定の頻度でおこるようになってから開始した．またそれとは別に，1回の繰返しで生成されたゴミの深さを測定して，性能との関連を調べている．以下でそれぞれの測定方法を述べる．

アプリケーションが最低限必要とするヒープサイズ ヒープサイズを制限せずに計算させ，その間に分散マーキングを非常に細かく行うことで生きているオブジェクトの最大量を求め，必要とするヒープサイズの基準とした．

オブジェクトの深さの分布 3章で述べたように，オブジェクトの平均の深さが深いほど参照カウントの効率は悪くなる．そこでオブジェクトがどのような深さを持って分布しているかを知るために，GCがおこらぬよう十分なメモリを与えて計算を1度だけ行い，その後参照カウントで，局所GCを繰り返してとれるゴミの量を計った．N 回目の局所GCで回収できるゴミの量が，深さ $N-1$ のオブジェクトの量であると考えることができる．

GCにかかった時間 実験を行った各ヒープサイズについて，参照カウントでは，局所GCにかかる時間とデリートメッセージの送受信にかかる時間を計測した．分散マーキングでは局所GCにかかる時間と分散マーキングにかかる時間を計測した．計測結果は48台のプロセッサそれぞれにおいてかかった時間を合計したものである．

4.3 アプリケーションの説明

2分木 プロセッサ間にまたがる13段（オブジェクトの深さはたかだか12）の2分木を60回繰り返し作るプログラムである．新たな2分木を作ると前回作った2分木はゴミとなる．したがって，主なゴミはプロセッサ間にまたがる2分木と，通信チャネルである．

8パズル 最短の解が17ステップである8パズル（4×4の15パズルを3×3にしたもの）を並列に20回連続して解く．プロセッサ間でのオブジェクトの参照は1段のものしか存在しない．つまり，深さ0と1のゴミのみが存在する．細かい通信を多数行うため，遠隔参照されるオブジェクト数は非常に多く，その大半はすぐにゴミとなる．

N体問題[10),11)] 8000体の並行オブジェクトを8分木で管理するN体問題を40ステップ繰り返し解くプログラムである．この8分木の深さはおよそ4〜5であり，毎ステップ新たな木を生成して以前の木はゴミとなる．2分木と比べて計算の途中でも多くのゴミを作るプログラムとなっている．

4.4 実験結果

図2 は各アプリケーションの生成するオブジェクトの深さの分布を表す．縦軸はゴミの量を示し，横軸はオブジェクトの深さである．図から，2分木におけるオブジェクトは，他のPEから参照されない深さ0のオブジェクトと深さ1の通信チャネルに加え，2分木のデータ構造が深さ12まで指数的に分布していることが分かる．また8パズルにおいても，深さ0のオブジェクトのみならず，相当数の深さ1のオブジェクトが存在することが分かる．対して，N体問題ではほぼすべてのオブジェクトが深さ0であり，1回の局所GCで回収可能である．

図3 がGCにかかった時間を示す．最低限必要なメモリ量 L の2，4，6，8倍のヒープを与えて計測した時間を左から並べている．並んだ棒の左が分散マーキング，右が参照カウントにかかった時間．LGCは局所GCにかかった時間，RRCは参照カウントにおいてデリートメッセージの処理にかかった時間，GGCは分散マーキングにかかった時間である．

GCにかかった時間をみると，2分木と8パズルでは参照カウントは分散マーキングよりも悪くなっている．ことに2分木ではヒープサイズが大きくなるほど性能差が目立つ．これは3章における我々の考察のとおり，参照カウントの効率はオブジェクトの深さの重み付き平均に比例するためである．さらに，オブジェクトの深さの重み付き平均がほぼ0であるN体問題では，参照カウントが分散マーキングよりも良い結果を示しているのも，我々の考察を裏付けている．

さらに分散マーキングでは，ヒープサイズが大きくなるにつれて一貫してGCにかかる時間が小さくなっている．それに対して参照カウントは，2分木と8パズルではヒープサイズが大きくなるにつれ，かかる時間が多少大きくなっている．3.2節における解析によれば参照カウントにかかる時間は一定となるはずであるが，実験結果ではそうなっていない．この原因の正確な究明はまだ行っていないが，以下のような原因が考えられる．我々の参照カウントの実装では，フロー制御のため，1つのデリートメッセージのサイズの上限を定めており，それを超えるデリートメッセージは複数回に分けて送信される．そのため，ヒープが大きくなり，それにつれて1回の局所GCで回収可能になるオブジェクト数が増えると，デリートメッセージの送受信にかかる時間が増え，それが全体の実行時間に影響しているものと考えられる．

分散マーキング全体は，ヒープサイズに反比例してGCにかかる時間が減少するのが理想的であるが，実

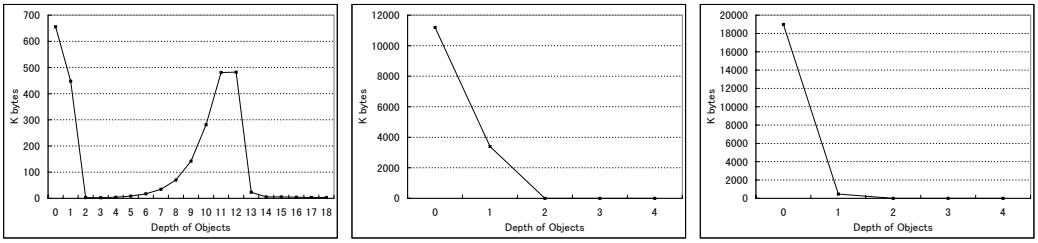


図 2 オブジェクトの深さの分布（左から 2 分木，8 パズル，N 体問題）
Fig. 2 Distribution of objects by their depth (from left to right, Binary-Tree, 8-Puzzle, N-body).

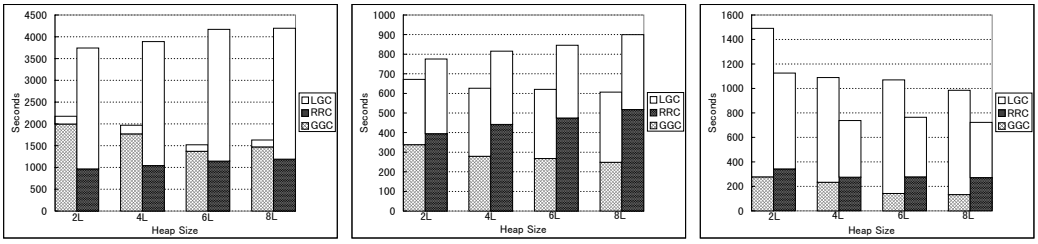


図 3 GC にかかった時間（左から 2 分木，8 パズル，N 体問題）
Fig. 3 Elapsed time of GC (from left to right, Binary-Tree, 8-Puzzle, and N-body).

験ではそれよりも減少率が小さくなっている．主な原因は，現在の分散マーキングの実装中に以下のような，ヒープサイズに比例して時間のかかる処理が含まれているためと考えている．2.3 節で述べたように，分散マーキングにおいて，各プロセッサはプロセッサ内のポインタのみを追跡する，局所マーキングを行った後，その局所マーキングによって新たに到達した遠隔参照に対してマークメッセージを送信する．ここで，新たに到達した遠隔参照を局所マーキング後に見つけるのに，そのプロセッサの輸入表（2.1 節）を走査しており，その表の大きさは当然のことながらヒープサイズにともなって大きくなる．

この，遠隔参照を発見するのに輸入表全体を走査する方式は，既存の局所 GC ライブラリを利用した局所マーキング処理の変更を最小限にするために採用した方式である（局所マーキング処理は到達したオブジェクトが通常のオブジェクトであるかスタブオブジェクトであるかを区別する必要がない）．局所マーキング処理を変更して，マーキング中に発見されたスタブを列挙した表を管理すれば，輸入表を走査する処理を取り除くことができ，実験結果も改善するものと予想している．今後原因を確認し，改善する予定である．

5. 関 連 研 究

参照カウントおよび分散マーキングをそれぞれ実装・評価した研究はいくつかある[4),12)~15)]．しかし，これらの研究中でなされている評価は実装した方式内のみにとどまっている．

上記の研究を含む多くの分散 GC の研究で，参照カウントと分散マーキングの性質の差としてあげられてきたものは，

- 遠隔参照されたゴミを回収するのに，参照カウントは遠隔参照をしているプロセッサとオブジェクトを持つプロセッサだけが仕事をすれば済む．一方，分散マーキングはすべてのプロセッサが GC に参加しなくてはいけない．
- 参照カウントでは毎回の局所 GC で徐々にゴミがとれる．
- 参照カウントでは互いに遠隔参照しあうようなゴミは回収できないが，分散マーキングはそのようなゴミも回収できる．

等である[16),17)]．既存の研究の多くは，大域的な同期を不要とする参照カウントを優れているとしている．しかしこれらの特徴は，どちらの方式が同じ仕事量でより多くのゴミを回収できるかを明らかにするものではない．

小池らは文献 18) において，参照カウントの欠点として，ルートから到達不可能なオブジェクトが，参照元の局所 GC が起動されない限り回収されないという点を指摘している．本論文が参照カウントの欠点として考察しているのもまさにこの点であり，我々はそのことが GC 全体の効率にどのくらいの影響を与えるか

を，アプリケーションの作るデータ構造の性質から定量的に導き，かつ実験によって示している．

6. 結論と今後の課題

本論文で我々は，参照カウント方式の分散 GC の性能が，潜在的には非常に悪いものになりうることを示した．特に参照カウントの費用が，生きているオブジェクトをマークする費用も含めて，オブジェクトの深さの重み付き平均（3.2 節）に比例するというのは，従来明確に指摘されていなかった．

しかし現実のアプリケーションでは，N 体問題で見たように，分散木を用いていても計算の途中に生まれる浅い（ほとんどは深さが 0 の）オブジェクトがゴミの大半を占める場合も多い．

より正確に参照カウントの利点を考えると，以下のようにいえる．

- **ほとんどのオブジェクトの深さが 0 であることが保証されているなら**，参照カウントは 1 台のプロセッサ上で行う GC と同程度に効率的である．
- 参照カウントは単純な実装でも通信遅延に影響を受けにくい．なぜなら，デリートメッセージのやりとりはプロセッサ間の同期を必要としないため，通信遅延が隠れるからである．一方分散マーキングは明らかに遅延の影響を受けやすく，これを改善するためにはマーク処理とユーザ処理のインタリーブ等を行わなくてはならず，GC は複雑化しオーバヘッドも大きくなる．

たとえば単純なサーバ–クライアント型の計算のように，ほとんどの遠隔参照はクライアントからサーバに向かっている場合，ほとんどのオブジェクトの深さは 0 か 1 であると期待できる．また通信遅延も大きいと考えられるので，参照カウントを用いるのは良い選択であろう．しかし，分散メモリ並列計算機上で行う並列計算のように，しばしば深いオブジェクトを生成し，また通信遅延も LAN のそれよりずっと小さい環境であれば，分散マーキングを用いる選択もあってしかるべきである．

本論文では主に GC の効率にのみ注目した，参照カウントと分散マーキングの比較を行った．しかし，従来の議論にもみられるように，GC の性質として人々がしばしば注目することに，GC の停止時間があげられる．そこで今後は通信遅延に強い分散マーキング[19),20)]を実装し，その実用性を参照カウントと比較することを考えている．

謝辞 多くの貴重な御意見をくださった，東京大学の近山隆先生，小林直樹先生，そして査読者の方々に感謝いたします．特に本論文の 3.2.3 項の考察は査読者の方の指摘を基にしています．

参 考 文 献

1) Hughes, J.: A Distributed Garbage Collection Algorithm, *Functional Programming Languages and Computer Architecture*, Lecture Notes in Computer Science, vol.201, pp.256–272, Springer-Verlag (1985).
2) Juul, N.C. and Jul, E.: Comprehensive and Robust Garbage Collection in a Distributed System, *Proc. International Workshop on Memory Management*, Lecture Notes in Computer Science, vol.637, pp.103–115, Springer-Verlag (1992).
3) Taura, K., Matsuoka, S. and Yonezawa, A.: ABCL/f: A Future-Based Polymorphic Typed Concurrent Object-Oriented Language – Its Design and Implementation, *Specification of Parallel Algorithms*, pp.275–291 (1994).
4) Taura, K. and Yonezawa, A.: An Effective Garbage Collection Strategy for Parallel Programming Languages on Large Scale Distributed-Memory Machines, *Proc. Principles and Practice of Parallel Programming*, pp.264–275, ACM (1997).
5) 山本泰宇，田浦健次朗，米澤明憲：分散メモリ並列計算機における Reference Count GC と Mark and Sweep GC の比較，*IPSJ SIG Notes 97-PRO-14* (*SWoPP'97 PRO*), Vol.97, No.78, pp.109–114 (1997).
6) Boehm, H.-J.: Space Efficient Conservative Garbage Collection, *Conference on Programming Language Design and Implementation*, pp.197–206, ACM (1993).
7) Boehm, H.-J. and Weiser, M.: Garbage Collection in an Uncooperative Environment, *Software Practice and Experience*, Vol.18, No.9, pp.807–820 (1988).
8) Bevan, D.I.: Distributed Garbage Collection using Reference Counting, *Parallel Architectures and Languages Europe*, Lecture Notes in Computer Science, vol.258, pp.176–187, Springer-Verlag (1987).
9) Watson, P. and Watson, I.: An Efficient Garbage Collection Scheme for Parallel Computer Architectures, *Parallel Architectures and Languages Europe*, Lecture Notes in Computer Science, vol.258, pp.432–443, Springer-Verlag (1987).
10) Barnes, J. and Hut, P.: A Hierarchical $O(N \log N)$ Force-Calculation Algorithm, *Nature*, Vol.324, pp.446–449 (1986).

11) 八杉昌宏，米澤明憲：N 体問題の並列オブジェクト指向アルゴリズム，日本ソフトウェア科学会第 8 回論文集 (1991).
12) Kamada, T., Matsuoka, S. and Yonezawa, A.: Efficient parallel global garbage collection on massively parallel computers, *Proc. SuperComputing*, pp.79–88 (1994).
13) Maeda, M. and Ishikawa, Y.: GLEANER-7: A Hybrid Distributed GC Algorithm, *Proc. OOPSLA Workshop on Garbage Collection and Memory Management* (1997). `ftp://ftp.dcs.gla.ac.uk/pub/drastic/gc/maeda.ps`.
14) 六沢一昭，市吉伸行：並列論理型言語 KL1 分散処理系の外部参照管理方式の評価，IPSJ SIG Notes 96-PRO-8（SWoPP'96 PRO 予稿集），Vol.96, pp.13–18 (1996).
15) 八杉昌宏：データ駆動並列計算機上の分散並行ガーベジコレクションの評価，並列処理シンポジウム JSPP'97, pp.345–352 (1997).
16) Plainfossé, D. and Shapiro, M.: A survey of Distributed Garbage Collection Techniques, *Proc. International Workshop on Memory Management*, Lecture Notes in Computer Science, vol.986, Springer-Verlag (1995).
17) Rodriguez-Rivera, G. and Russo, V.: Cyclic Distributed Garbage Collection Without Global Synchronization in CORBA, *Proc. OOPSLA Workshop on Garbage Collection and Memory Management* (1997). `ftp://ftp.dcs.gla.ac.uk/pub/drastic/gc/backtrace.ps`.
18) 小池汎平，田中英彦：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，並列処理シンポジウム予稿集，pp.273–280 (1990).
19) Hughes, J.: A distributed garbage collection algorithm, *Proc. Functional Programming Languages and Computer Architecture*, Lecture Notes in Computer Science, vol.201, pp.256–272, Springer-Verlag (1985).
20) Fessant, F.L., Piumarta, I. and Shapiro, M.: An implementation of complete, asynchronous, distributed garbage collection, *Proc. Programming Languages Design and Implementation*, pp.152–161, ACM (1998).

付　　録

A.1　3.2.3 項の証明

全プロセッサ 1 度ずつの GC で，o_1 から o_k まで，またはそれ以上のオブジェクトが回収される確率 P_k は，$P_k = 1/k!$ である．ここから，全プロセッサ 1 度ずつの GC で，o_1 から o_k までが回収され，o_{k+1} は回収されない確率 p_k を求めると，

$$p_k = \begin{cases} P_k - P_{k+1} & (k < n) \\ P_k & (k = n) \end{cases}$$

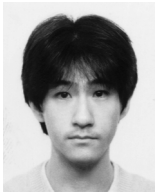
となる．ここから，全プロセッサ 1 度ずつの GC で回収されるオブジェクト数の期待値を求めると，

$$\begin{aligned} & \textstyle\sum_{k=1}^{n} kp_k \\ &= 1(P_1 - P_2) + 2(P_2 - P_3) + \cdots \\ &\qquad + (n-1)(P_{n-1} - P_n) + nP_n \\ &= P_1 + P_2 + \cdots + P_n \\ &= 1/1! + 1/2! + \cdots + 1/n! \\ &< e - 1 \end{aligned}$$

(平成 11 年 8 月 31 日受付)
(平成 12 年 3 月 2 日採録)

山本　泰宇

1974 年生．1998 年東京大学大学院理学修士取得，博士課程進学．並列分散システム，ゴミ集めに関する研究に従事．



田浦健次朗（正会員）

1969 年生．1996 年より東京大学大学院理学系研究科助手．1997 年東京大学大学院より理学博士取得．並列プログラミング言語の設計，実装に関する研究に従事．

米澤　明憲（正会員）

1947 年生．1977 年 Ph.D. in Computer Science（MIT）．1989 年より東京大学理学部情報科学科教授．超並列・分散ソフトウエアアーキテクチャ，等に興味を持つ．共著書「算法表現論」,「モデルと表現」（岩波書店），編著書「ABCL: An Object-Oriented Concurrent System」（MIT Press）等がある．1992〜1996 年ドイツ国立情報処理研究所（GMD）科学顧問，ACM TOPLAS 副編集長，IEEE Parallel & Distributed Technology および Computer 編集委員等を歴任，元日本ソフトウェア科学会理事長．