Vol. 39 No. 6 情報処理学会論文誌 June 1998

並列オブジェクト指向言語のためのガーベジコレクタ

八 杉 昌 宏†

並列オブジェクト指向言語処理系のための並列オブジェクトのガーベジコレクション（GC）方式を提案する．並列オブジェクトモデルを分析し，mutatorとセルを用いた単純なモデルに帰着させることで，ガーベジコレクタの設計を行う．提案するGC方式は，(1) マーク&スイープに基づき循環した参照を持つゴミも回収する，(2) 通常のプログラムの実行を長時間ブロックせず，並行して処理を行う，(3) 通常のプログラム実行に課するオーバーヘッドが小さい，という特徴を持つ．これらの特徴を満たすため，本GC方式では，オーバーヘッドの大きいリードバリア・ライトバリアを並列オブジェクトの内部に対して用いずに，個々の並列オブジェクトを単位として一括処理を行う．実装と評価は，電総研で開発された非共有メモリ型のデータ駆動並列計算機EM-4上で行った．

A Garbage Collector for Concurrent Object-oriented Languages

Masahiro Yasugi†

A garbage collection (GC) scheme of concurrent objects for concurrent object-oriented language systems is proposed. We make the GC design simplified and confirmed by decomposing the concurrent object-oriented computation model into a simple and essential model based on “mutators” and “cells.” The GC scheme features that (1) it can collect garbages which have cyclic references, (2) it runs concurrently and does not block the normal computation for a long time, and (3) it does not impose large overhead upon the normal computation. These are realized by using atomic processing for each object rather than using read-barrier/write-barrier inside the object. The implementation and evaluation were performed on the ETL’s distributed-memory data-driven parallel computer EM-4.

1. はじめに

本論文では，並列オブジェクト指向言語処理系ABCL/EM-4[8]における並列オブジェクトのゴミ集め（GC）方式[7]を提案し，その評価を行う．ABCL/EM-4[8]は，データ駆動並列計算機EM-4[3]を対象マシンとし，並列オブジェクト指向言語ABCL/ST[4]をソース言語とする処理系である．

並列オブジェクト指向言語では並列オブジェクトやメッセージを用いたモデルについてガーベジコレクションを考えねばならず，その設計を困難にしている．本論文では，並列オブジェクト指向言語モデルをmutatorとセルからなる単純で，本質的なモデルの視点から捉えることで，単純な基本アルゴリズムの並行化を応用して分散並行ガーベジコレクタの設計を容易に正しく行えることを示す．

提案するGC方式は，循環した参照を持つゴミも回収するため，参照カウント方式とはせず，マーク&スイープに基づくものとした．対象としたEM-4は分散メモリ型データ駆動計算機であるため，GC方式の設計においては次の点を考慮する必要がある．

- プログラムは実際に並列計算機により並列実行される．また分散GCが必要となる．
- データ駆動計算機ではハードウェアがスケジューリングを自動的に行う．このため，通常の計算機でしばしば採用される，通常の処理を止めてGCを行う方式が採用できず，通常の処理と並行してゴミ集め処理を行う並行GCが必要となる（ただし，提案方式では通常処理用計算資源の下限は保証しない）．例えば，一旦通常計算を止めて，ハードウェアのスケジューリングキューの内部にアクセスするといったことはできない．
- データ駆動計算機は，リモートメッセージ送信のオーバーヘッドが，その他のメッセージパッシング型並列計算機と比べてかなり小さい．このためオブジェクトの多くをリモートのノードに作って負荷分散させることが容易で，ノードを跨った参

† 京都大学大学院情報学研究科通信情報システム専攻
Dept. of Communications and Computer Engineering, Graduate School of Informatics, Kyoto University

照が多くなる傾向がある．実際，ABCL/EM-4 処理系ではノードを跨った参照関係とノード内部の参照関係を区別して扱わない．今回の GC 方式の設計においても，この扱いは同様とする．

以下，2 章では，並列オブジェクト指向言語処理系 ABCL/EM-4 について述べる．3 章では，4 章で並列オブジェクト指向言語用の GC アルゴリズムについて述べるための準備として，単純で，本質的なモデルにおけるゴミの定義と並行 GC についてまとめる．4 章では並列オブジェクト指向言語用の GC アルゴリズムについて提案し，5 章ではその実装について述べる．6 章では，実際に実装を行い並列計算機 EM-4 で実験した評価結果について述べる．

2. 並列オブジェクト指向言語処理系

この章では言語の計算モデルと，EM-4 のアーキテクチャについて述べる．

2.1 並列オブジェクト指向言語 ABCL/ST

ABCL/ST[4] は，ABCL/1[5] に部分型を含む静的な型付けなどを導入した言語である．

ABCL/ST の計算/プログラミングモデルでは，並列の単位となるのは，独自のメモリと計算能力を持つ多数の並列オブジェクト（以下しばしばオブジェクトと略す）であり，これらはメッセージパッシングのみを通して互いに協調しながら並列に計算を行う．また，オブジェクトは動的に生成することができる．

ABCL では，オブジェクトは状態変数とメソッドを持つ．オブジェクトは休眠，待機，活性のいずれかのモードにあり，休眠モードでメソッド実行可能な要求メッセージを受け取ると，活性モードとなってメソッドを実行し，メソッド実行の完了とともに休眠モードに戻る．また，メソッド実行中に別のメッセージが必要な場合は，そのメッセージを受け取って実行を再開するまでの間，待機モードになる．

メッセージ送信には，past 型（非同期メッセージ送信）と now 型（非同期メッセージ送信+返答メッセージ待ち）がある．非同期に送られた要求メッセージは，受け手オブジェクトで，そのメッセージキューに入れられ，順番に処理される．また返答メッセージは，返答メッセージ待ちオブジェクトの実行を再開させる．

2.2 EM-4 のアーキテクチャ

ABCL/ST コンパイラは，細粒度ハイブリッド並列アーキテクチャEM-4[3] のアセンブリコードを生成する．電総研で開発され，稼働中の EM-4（プロトタイプ）は，80 個の要素プロセッサ（PE）からなり，12.5 MHz のクロック速度で動作し，細粒度の通信メカニズムを提供する．例えば，レジスタ上から FIFO 性を持つ高速なオメガ網にデータを直接送出する 2 ワードパケット出力命令などがある．EM-4 のハードウェアはまたそのデータ駆動（パケット駆動）の特徴と組み合わせることにより，複数個のスレッドの基本的スケジューリングメカニズムも提供する．ハイブリッドというのは，制御フローアーキテクチャとデータ駆動アーキテクチャの融合を意味する．

3. ゴミ集めの基本アルゴリズム

ここでは，4 章で並列オブジェクトの GC について説明する準備として，より基本的なモデルを用いて用語を整理する．

3.1 ゴミの定義

以下のような単純なモデルを考える．計算（通常のプログラム実行）はただ一つの **mutator** が，セルの集合を用いて進める．各セルは 0 個以上のスロットから成る．スロットには，セルへの参照（ポインタ）が保持される．そのセルを，そのスロットから参照されるセルと呼ぶこととする．mutator に属するスロット（レジスタ等に対応）をルートと呼ぶ．mutator は，ルートから参照されているセルのスロットの内容（参照）をルートにコピー（リード）したり，逆にルートの内容をルートから参照されているセルのスロットにコピーしたり（ライト）することができる．

このとき，ある瞬間に生きているセルはルートから参照を辿ることで到達可能なセルで，次のように再帰的に定義できる: (1) ルートから参照されているセルは生きている．(2) 生きているセル（のスロットのどれか）から参照されているセルは生きている．生きているセルでないなら，そのセルはゴミである．

3.2 ゴミの検出

スロットに白，灰，黒の色のどれかを付けるものとして，ゴミの検出アルゴリズムを記述可能であるが，紙面の都合より割愛する．スロット毎にではなく，セル毎に白，灰，黒の色のどれかを付けるものとすると次のようになる．ただし，アルゴリズムの記述を容易とするため，検出アルゴリズムにおいては必要に応じてルートもスロット数が 1 のセルと考えることとし，通常のセルと同様に色を付ける．

この節では mutator を停止させて行う GC について考える．次の手順で **collector** は，セルをゴミとゴミでないものに分けることができる．

[ゴミ検出の基本アルゴリズム]

(1) ゴミ検出の開始時点においては，すべてのルートは灰，その他すべてのセルは白とする．

(2)　灰色のセルがなくなるまで，灰色のセルを一つ選び o とし，次の処理を繰り返す:
(i) o のすべてのスロットにつき，そのスロットから参照されているセルをマークする．ここでセルをマークするとは，セルが白だったら灰にすることをいう．白でなければそのままにする．
(ii) o を黒にする．

(3)　ゴミは白として残り，ゴミでないセルは黒となる．

ゴミを検出中，灰または黒のセルは生きており，白のセルは生きているかどうかまだわからない．上の手順では，黒のセルが直接，白のセルを参照することはない．つまり，白のセルは灰のセルから参照されることがあっても黒のセルから参照されることはない．このため，灰色のセルがなくなった時点で白のセルは生きているセルから参照されていない，つまりゴミであることがわかる．

上の手順では色は濃くなる方向に変化する．マーク&スイープの言葉を借りて，白だったなら灰へ変化させることをマークすると呼ぶ☆．一方，灰から黒への変化（上の手順における o に対する (i), (ii) の処理）については，一般的な用語がないようなので，ここでは調査すると呼ぶことにする．セルの灰から黒への変化において，そのセルから参照されるセルはマークされる．つまり，セルを調査することで，そのセルの内部にはマークされていないセルへの参照が存在しないことが保証される．

ここで述べた検出アルゴリズムにおいてはルートにも色を付けたが，ルートには色を付けずに，(1) と (2) の間でルートから参照されるセルを灰にする処理を行うほうが，どちらかというと一般的である．しかし次章以降で述べる並行 GC に，ルートから参照されるセルを灰にする処理も並行に行う場合も含めるにはルートにも色を付ける必要がある．

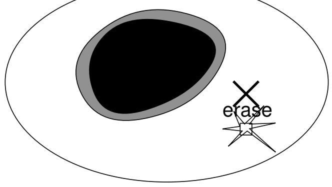
3.3　並行ゴミ検出

ゴミ検出が終了するまで mutator を停止させたままにしないで，ゴミ検出処理と並行動作させることができる．

並行動作させる場合，mutator の一部の操作に関して collector の処理への協力を得るためのバリアが必要になる．バリアには，(a) スナップショットを保存するものと，(b) 黒のセルが直接，白のセルを参照しないようにするものがある[9),10)]．従来，通常のセルからルートへの読み出し（リード）に関するバリアをリードバリア，ルートから通常のセルへの書き込み（ライト）に関するバリアをライトバリアと呼んで分類することが多かったが，本研究では，より本質的なバリアとして消去バリアとコピーバリアを提案する．

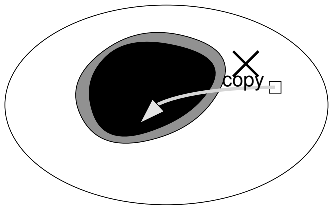
(a) は，次のような見方ができる．mutator が動作することで，参照のコピー操作やオリジナルの参照（＝ゴミ検出開始時に保持されていた参照）の消去が起こるが，このうち参照のコピー操作は自由とするが，まだその先のマークが行われていないオリジナルの参照を消去するのを防ごうというものである．このため，白または灰の領域（＝セルの集合）でのオリジナルの参照の消去においては，その消去しようとする参照の参照先をマークする（図 **1** のようなバリアを張る）．また黒の領域でのオリジナルの参照の消去はそれを辿るマークがすでに済んでいるので問題ない．このバリアにより，オリジナルの参照からなる参照関係のスナップショットに対して GC を行うことになる．

一方 (b) は，次のような見方ができる．参照のコピー操作やオリジナルの参照の消去のうち，参照の消去は自由とするが，まだその先のマークが行われていない参照が既に調査したはずの黒の領域にコピーされるのを防ごうというものである．このため，白または灰の領域から黒の領域への参照のコピーにおいては，そのコピーされた参照の参照先をマークする（図 **2** の



白または灰の領域で参照を消去する場合，
消去しようとする参照の参照先をマークする

図 1　消去バリア
Fig. 1　Erase-barrier



白または灰の領域から黒の領域へ参照をコピーする
場合，コピーされる参照の参照先をマークする

図 2　コピーバリア
Fig. 2　Copy-barrier

☆ この論文で用いる「マークする」という用語は必ずしもマークビットを立てるという意味ではなく，新たに生きていることが判明したセルを collector が処理の対象として認識するという意味である．コピー GC では，コピーすることで「マーク」を行う．

ようなバリアを張る）．また黒の領域からの参照のコピーはそれを辿るマークがすでに済んでいるので問題ない．このバリアにより，変化していく参照関係に対して GC を行うことになる．

(a) のバリアを消去バリア，(b) のバリアをコピーバリアと呼ぶこととする．並行ゴミ検出を実現するにはどちらか片方のみを用いればよい（両者を異なるフェーズで組み合わせて用いる方式の提案[9]もある）．消去バリア・コピーバリアは参照数の増減に直接関係している点からリードバリア・ライトバリアより本質的なものである．リードはルートへのコピーとルートでの消去であり，ライトは通常のセルへのコピーと通常のセルでの消去であることから，GC 開始時に mutator の処理や collector の他の処理に優先してまずルートをすべて黒にすることで，(a) ではルートでの参照の消去に関するリードバリアをなくして通常のセルへのライトバリアのみとすることができ，一方 (b) ではルートへのリードバリアのみとすることができる．

新しいセルの生成では，セルの初期値をルートからコピーすることで初期化するとともに，新しいセルへの参照がルートにコピーされるものとする．ゴミ検出中に生成された新しいセルの色は次のようにする．(a) の消去バリアを用いる方式では，新しいセルは検出の対象外ですべて生きているとする．新しいセルはマーク処理や調査処理をすることなく黒とすればよい．これは，新しいセルの初期化に使われた参照のオリジナルは必ずマークに使われるからである．

一方 (b) では，新しいセルも検出やバリアの対象となるため，検出完了までにゴミになるかもしれない場合（そのためには新しいセルへの参照を保持することになるルートは黒でない必要がある），新しいセルは白としておくこともできる．また新しいセルをあらかじめ黒にしておいてもよい．その場合には，初期化のコピーに対してもコピーバリアが必要となるが，ルートが黒の場合など初期化に使われる参照のマークが済んでいればコピーバリアを働かせなくて済む．

4. 並列オブジェクトのゴミ集め

ここでは，3 章で述べた方式を用いて，並列オブジェクト指向計算におけるガーベジコレクションについて述べる．

4.1 ゴミの定義

並列オブジェクト指向計算においては，これまで見てきた単一の mutator とセル集合だけを扱う場合より複雑なモデルが必要になるが，以下では，並列オブジェクト指向計算モデルを解析して，mutator とセルからなるモデルの視点から捉えることで，3 章の考え方が適用できるようにする．その結果，ルートもしくは生きているセルから参照されるセルが生きていて，それ以外はゴミであるという 3 章での定義をそのまま利用できる．

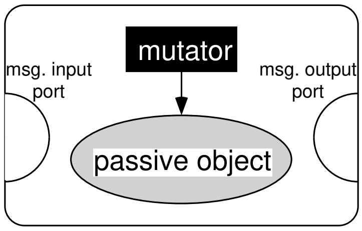


図 3 並列オブジェクトのモデル
Fig. 3 Model of Concurrent Object

モデルを並列オブジェクト空間とメッセージ空間（ネットワーク）に分けて考える．

並列オブジェクト空間には並列オブジェクトが存在する．各並列オブジェクトは受動的オブジェクト（セル）と (virtual) mutator の組であると考える（そのほかのメッセージキューなどは受動的オブジェクトの一部とみなして以後無視する）．ここで，オブジェクトの数だけの mutator を考えていることに注意してほしい．この mutator は，組となっているセル以外の他のオブジェクトのセルに直接アクセスすることはない（カプセル化）．また，この mutator はメッセージを作ってネットワークに送り出したり，ネットワークからメッセージを取り込んだりできる（**図 3** 参照）．また，並列オブジェクトへの参照は，受動的オブジェクト（セル）への参照と mutator への参照の両方と考える．

並列オブジェクトが，メソッド実行や次のメッセージの取り出しなどの処理を行なっている間をアクティブな状態と呼び，逆にメッセージ待ちなどで実行が続けられない状態（＝実行がサスペンドされている状態）を非アクティブな状態と呼ぶ．従来の並列オブジェクトの GC では，並列オブジェクトをそのまま扱うもの[1]も多く，collector の処理の正しさを明確に示すことを困難にしていた．本研究では，並列オブジェクトがアクティブか否かを，次のように mutator と受動的オブジェクトの参照関係の有無に翻訳することで，並列オブジェクトをセルと mutator からなる単純なモデルに分解する．つまり，並列オブジェクトには mutator に属するスロット（ルート）が存在し，アクティブな並列オブジェクトでは，受動的オブジェクトは mutator（のルート）から参照されていると考える

(図 3 参照).逆に非アクティブな状態になっている間は,mutator からその受動的オブジェクトへの参照は存在しないものとする.もし mutator からその受動的オブジェクトへの参照が常に存在しているものとしてしまうと,そのままでは,全ての受動的オブジェクトがルートから参照されていて生きていることになってしまうためである.また,mutator(のルート)と受動的オブジェクトは独立に色を持つものとする.

メッセージ空間(ネットワーク)には,飛行中メッセージが存在する.各飛行中メッセージは,受動的メッセージ(セル)とそれを運ぶためのそれぞれの mutator の組になっていると考える.そして,並列オブジェクトの場合と同様に,それぞれのメッセージでは,mutator (のルート)から受動的メッセージへの参照があるものと考える.また飛行中メッセージはそのメッセージがオブジェクトに取り込まれた時点でメッセージ空間からは消滅するものとする.

以上まとめると,たくさんの mutator が存在しているわけだが,ゴミの定義は,3.1 節の場合と同じで,ゴミとは(いずれかの mutator に属する)ルートから参照を辿ることで到達可能な受動的メッセージ・受動的オブジェクト以外である.すべての飛行中のメッセージ,およびアクティブなオブジェクトは生きており準ルートと考えることもできる.

4.2 分散並行ゴミ検出

ゴミ検出も基本的には 3 章と同様である.つまり,4.1 節で述べたようにオブジェクトまたはメッセージの mutator に属するルートを灰(ルートではない)受動的オブジェクトや受動的メッセージを白のセルとして [ゴミ検出の基本アルゴリズム] を用いる.

ただし,オブジェクト空間やメッセージ空間は,collector にとっても分散しているので,collector も複数個で分散して処理を行うものとする(それぞれが灰色のセルを選んで調査していく).また,オブジェクト空間は分散しているので,受動的オブジェクトの現在の色というものを系全体で同時に観測することはできない.提案方式では受動的オブジェクトの色を次のように定める.まず,受動的オブジェクトと同じ場所に受動的オブジェクトが黒であることを示す情報は存在することとする.また,collector が系のどこかで,オブジェクトをマークしようとしたときには,オブジェクトが存在する場所の collector にマークを依頼するためのマークメッセージを送出するものとする.提案方式ではマークメッセージ送出時点において論理的なマークが行われた(つまり受動的オブジェクトが白だったら灰になった)と考える.つまり,灰になった時点で,オブジェクトが存在する場所では灰になったことを示す情報はなくてもかまわないものとする.

一方メッセージについていえば, collector はネットワーク内部をアクセスできないと仮定しなくてはならないので工夫が必要となる.さらにいえば,受動的メッセージを運んでいる mutator を停止させる(ネットワーク上のデータの移動を停止させる)のも難しい.そこでここでは,全 mutator を停止させるタイプの GC については考えずに,並行ゴミ検出について考える.このとき 3.3 節で場合分けしたように,(a) 消去バリアベースか,(b) コピーバリアベースかの二つが考えられる.(a) をベースにするとスナップショットに対して GC が行われることになる.このタイプの GC の終了判定の効率化を図ったものには文献 6) などがある.一方,提案する方式は (b) をベースとする.後に実装のところで提案するように,(b) をベースとすることで,通常計算時のオーバーヘッドが無く GC 開始時に長時間計算を止めないで済むアクティブバリアを利用することができる.

(飛行中)メッセージは生きているので,ゴミ検出を完了するには,ネットワーク中のメッセージすべてが黒になる(調査されている)必要がある.メッセージが黒とは受動的メッセージとその mutator がともに黒であることとする.そのためには,すでに飛行中のメッセージについては到着するのを待つこととするが,新しいメッセージについては最終的に黒として送出されるようにしていく必要がある.これを実現するには (b1) メッセージを送出するところで黒くする方法と,(b2) オブジェクトの mutator(のルート)を黒くしていく方法がある☆.(b1) はメッセージ送信時に新しいメッセージを黒として生成し,その初期化にコピーバリアを用いていると考えることができ,並列オブジェクト指向言語 POOL-T での GC[1] で用いられている.一方,(b2) は mutator が黒であるオブジェクトから送信されたメッセージはそのまま黒にできることを利用して,メッセージの初期化にコピーバリアを用いずに間接的にメッセージを黒にする.(b2) では黒にされた mutator が調査済みの状態を保つ必要があるが,これにはすでに説明したコピーバリアを用いればよい.提案する方式はこの (b2) の方式を用いた.(b1) 方式との比較は今後の課題である.

mutator と,受動的オブジェクトの関係を含めて整理すると,必要なコピーバリアは,(B1) 黒の mutator

☆ (b1) と (b2) を同時に行うのが本来であるが,どちらかを優先して行うことで必要なバリアを減らすことが可能になる.

がメッセージを取り込む際にメッセージが黒でなければメッセージから参照される先をマークする「メッセージ取り込みバリア」，(B2) 黒の mutator の並列オブジェクトがアクティブになった（=黒の mutator から受動的オブジェクトへの参照が回復した）ときに受動的オブジェクトが黒以外なら黒にする「アクティブバリア」，となる（さらに新しいオブジェクト生成に関してその初期化のところにもバリアが必要である．新しいオブジェクトの mutator は黒にしておく）．

本来のコピーバリアの考えだけからいうと，(B2) は，(B2') 並列オブジェクトがアクティブになったときに受動的オブジェクトが白なら灰にするコピーバリア，を用いれば十分なのであるが，その後，受動的オブジェクト内の参照を mutator のルートへ読み込む際のコピーバリア（リードバリア）が必要となり，リードバリアのオーバーヘッドの大きいため (B2) のようにした．(B2) により個々のオブジェクト内部の GC については並行 GC ではなく一括 GC を行っていることになる．一括処理することで，オブジェクト内部でのリード 1 回につき機械語数命令分のチェックが省け，オーバーヘッドをかなり小さくできる．

ゴミ検出は灰色のセルがなくなった時点で終了するため，メッセージの mutator，受動的メッセージ，オブジェクトの mutator が黒となっており，なおかつ，受動的オブジェクトが白または黒となっていることを検出する必要がある．最終的には灰色の受動的オブジェクトが存在しないことを検出する必要があり，そのためには，マークメッセージが存在しないことも検出する必要がある．

5. 実　　装

この章では，実際に EM-4 に実装した，より具体的なアルゴリズムを示す．オブジェクトは大域的アドレスで識別され，輸出入表などを介することなく高速にメッセージを交換可能な実装となっている．

マスター collector と，各要素プロセッサ（PE）に配置した collector が，通常計算と並行して行う GC アルゴリズムは次のようになる．マスター collector は 0 番の PE に配置した（collector とも並行実行）．各 PE に配置した collector は，同じ PE 内のオブジェクトは自分でマーク・調査するが，異なる PE 上のオブジェクトはその PE の collector にマークメッセージを出してマーク・調査を依頼する．この時，ACK の返答先も送る☆．また，各 collector は，ACK の返答先を一つだけ保存し，PE 内に灰色のオブジェクトがなくなり，他の collector からの ACK もすべて戻ってきていれば，その一つだけ保存していた返答先に ACK を返す．既に ACK の返答先を保存しているときに ACK の返答先を受けとった場合はそのまま ACK を返す．マスター collector が，各 collector に最初に ACK 返答先を保存させておくことで，マスター collector にすべての ACK が帰ってきた時点で，系全体に灰色のオブジェクトが存在しないことが検出できる．

GC 処理の大まかな流れは以下のようになる．

(1) mutator はオブジェクトの生成ができない場合，すでに GC が開始されていなければ，マスター collector へ GC 開始の依頼．

(2) マスター collector は，最初の GC 開始依頼を受けて，各 PE の collector へ GC 開始の依頼（(6) 用の ACK 返答先付きで）．

(3) 各 PE の collector は，mutator と並行動作しながら，その PE の中の並列オブジェクトを一つ一つアクティブかチェックし，(1) アクティブであればその並列オブジェクトを調査する．(2) アクティブでなければ，そのオブジェクトのハンドラ（メッセージを受けとった際に起動されるコード）を置き換えることでアクティブバリアを有効にする．各 PE の collector は，すべてのチェックが終わればマスター collector へ通知．

(4) マスター collector はすべての collector からのアクティブチェック終了の通知を受けとったら，ブルドージング[6] を行う．ブルドージングとは，ネットワーク上のメッセージを押し出すことで，ブルドージング開始以前に送出されたすべてのメッセージがブルドージング終了までに到着していることを保証するものである．EM-4 では，FIFO 性を保証しているサーキュラ・オメガ網に沿ってブルドージングパケットを流す．ブルドージング終了後，ネットワーク上には（アクティブチェック終了以後送出された）黒のメッセージしかないことが保証される．

(5) マスター collector から各 PE の collector へ黒のメッセージしかないことを通知．この通知を受けとるまでは，各 PE の collector は，最後の ACK を返せないものとする．

(6) 各 collector で，マーク・調査処理が続けられ

☆ 実際にはマーク先と ACK の返答先は同時に EM-4 の一つのパケットに収まらないのでそれぞれ別のパケットで送る．この際，デッドロックを防ぐためには ACK の返答先を先に送る．

るが，マスター collector がすべての collector から ACK を受けとると終了．

(7) マスター collector から，各 collector へスイープの依頼．

(8) 各 collector は，白いオブジェクトを回収する．回収後，マスター collector へ通知．

(9) マスター collector はすべての collector からのスイープ終了の通知を受けとったら，各 collector へ GC 終了の通知．

アクティブバリアは，オブジェクトのハンドラをアクティブバリアを含むコードとアクティブバリアを含まないコードとに切り替えるだけで，着脱可能である．ノード collector は，そのノードでサスペンドしている mutator に（他の mutator と並行動作しながら，mutator を長時間ブロックすることなく）アクティブバリアを順に付けていくことができる．通常計算時は，アクティブバリアを含むコードは実行されないため，バリアのオーバーヘッドは全く無い．

また，mutator と collector は同じ PE 内で並行に動作するが，排他的にスケジュールされる．mutator は，自分がスケジュールされていない時には，collector がスケジュールされた場合に備えて，オブジェクト内の調査すべき変数の地図を保存しておく．

mutator が collector との並行動作に協力するためのバリアは，アクティブバリアだけでなく，黒のオブジェクトが黒でないメッセージを受けとった場合にメッセージに含まれる参照の先をマークするメッセージ取り込みバリアもある．このため，メッセージ送信時に送信元の mutator は，メッセージ中の参照の位置がわかるようにしておくとともに，メッセージに mutator の色を引き継がせる．送信元の mutator の色を引き継がないで，ブルドージング完了以前に届いたすべてのメッセージに関して，メッセージから参照される先をマークする方法も考えられる．この方法は，色の情報が必要ない分，メッセージ通信にかかるオーバーヘッドは小さくなるが，GC 時のマークの数は増える．

紙面の都合で詳しく述べられないが，実際には，GC 中のオブジェクト生成に対応するための処理がある．注意しなくてはならないことの一つには，GC 中に生成されたオブジェクトはメッセージ取り込みバリアが有効になる前にもメッセージをキューに貯めていることがあり，これらのメッセージも調査しなくてはならないということがある．

6. 実験と評価

前章で提案した GC 方式を，並列オブジェクト指向言語処理系 ABCL/EM-4 に組み込む形で実装し，実際に GC 処理を含むプログラム実行を行い，動作を確認した．

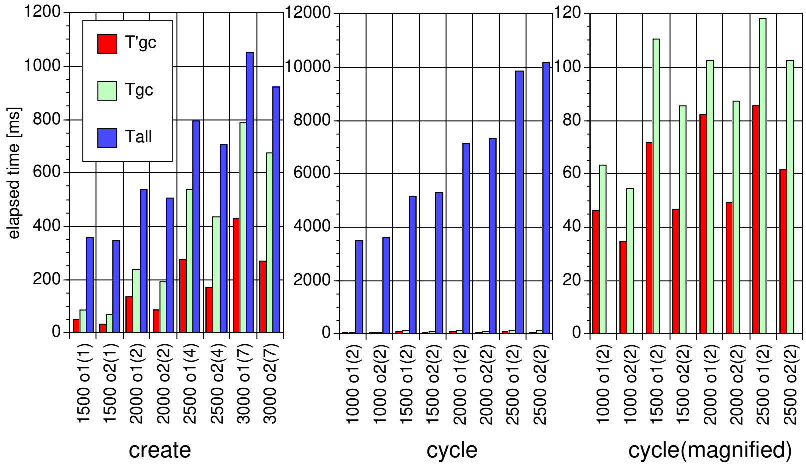
EM-4 の 80 個の要素プロセッサは，それぞれ 0.5MB のヒープを持つ．その約 2/3 を 0.5KB 単位のフリーリストとして並列オブジェクトに用いる．

アプリケーションプログラムには N 体問題の木構造を用いたアルゴリズム [2),4)] を用いた．本プログラムでは，並列オブジェクトをノードとする木構造 [4)] を生成するのに，木構造の足し合わせという手法を用いており，この際，足し合わせの中間結果にあたる並列オブジェクトがゴミとなる．また，木構造の親と子のノードは最終的に双方向の（循環した）参照を持つ．木構造の生成後は力を計算するフェーズに移る．木構造生成フェーズと力の計算フェーズを合わせて一サイクルとして一タイムステップ先の N 個の質点の位置と速度を計算する．各サイクルの木構造生成フェーズでは前回生成した木構造に沿って足し合わせを行う．このため，実際のプログラムでは，さらに初期木構造を生成するフェーズがあり，その後各サイクルを実行する．

今回はこの初期木構造生成フェーズ（**create**）と，通常のサイクル一回分（**cycle**）を評価の対象とし，質点数（N）は，500, 1000, 1500, 2000, 2500, 3000 と変化させた．**create** は，**cycle** より頻繁にオブジェクトを生成し，両フェーズの性質は全く異なるため，二つのアプリケーションで評価していると考えて良い．また，メッセージに色を付けずにブルドージング完了前のメッセージに関して常にマークを行う方式（**o1**）と，メッセージに色を付けて必要な時だけマークを行う方式（**o2**）について比較した．

図 **4** には，横軸を質点数と GC 起動回数（括弧内）とし，縦軸に実行時間を示す．T_{all} は GC 処理を含むプログラムの総実行時間である．T'_{gc} は，GC が起動されるようにした時の総実行時間の増加で，GC の処理に費やされているものと考えられる（明示的にゴミをフリーリストへの返却するコード（機械語 2 命令）を挿入したプログラムとの実行時間の差として計算）．一回の GC あたりの実行時間の増加は **o1** で 23〜69ms 程度，**o2** で 17〜43ms 程度であった．T_{gc} は実際に GC が起動していた時間を表す．この時間は T'_{gc} より長いが，これは GC が通常の計算と並行して実行されているためで，T'_{gc}/T_{gc} から，39〜80%程度の処理が GC に費やされていると考えられる．またブルドージングに要する時間は 0.07ms 程度であった．

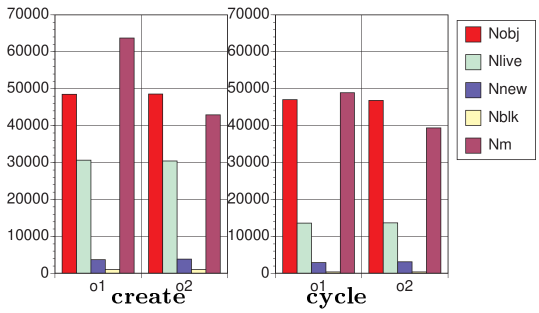
メッセージに色を付けない方式（**o1**）とメッセー



時間の単位はミリ秒; グラフ中のバーは左から順に，T'_{gc}: GC による実行時間増加分，T_{gc}: GC 時間，T_{all}: GC を含む実行時間

図 4 GC 実行時間の比較

Fig. 4 Evaluation of elapsed time



グラフ中のバーは左から順に，
N_{obj}: GC の対象とされたオブジェクトの数，
N_{live}: GC を生き残ったオブジェクトの数，
N_{new}: GC 中に生成されたオブジェクトの数，
N_{blk}: GC 中に実行をブロックされたオブジェクトの数，
N_m: GC 中にオブジェクトに対して行なわれたマーク数

図 5 GC のプロファイル（一回あたりの平均）

Fig. 5 GC average profile (for one GC invocation)

ジに色を付ける方式（**o2**）の実行時間を比較すると，**create** については **o2** のほうが総実行時間（T_{all}）が短い．一方，**cycle** についてはもともとのオーバーヘッドの小さい **o1** のほうが総実行時間が短い．**o1** の方が GC の処理が増える理由は，メッセージに色を付けずおいて，メッセージを受け取ったオブジェクトが必要以上にマークするためと考えられる．そこでマークの数などのプロファイルをとった結果を図 **5** にまとめた．図 5 は，N を 500 から 3000 と変化させたときの **create** と **cycle** のそれぞれ 14 回，10 回の GC でのオブジェクトの数やマーク数などの合計から一回の GC 当たりの平均値を求めたものである．これによると **o1** のほうがマーク数に関してだけ多いことがわかる．また，図 5 から今回評価に用いた N 体問題のプログラムでは，GC を生き残るオブジェクトは **create** で平均 63%程度，**cycle** で平均 29%程度であること，GC 中にも GC を生き残ったオブジェクトの 12～23%程度の新たなオブジェクトの生成が行われていること，GC 完了までオブジェクトを生成できずに実行をブロックされたオブジェクトは GC を生き残ったオブジェクトの 3%程度と少ないこと，などがわかる．なお，GC 開始時の空きメモリ量のばらつきについては，25%以上の空きがあるケースは 1%未満であり，約半数のケースで空きは 12%以下であった．

7. おわりに

本論文では，データ駆動並列計算機上の並列オブジェクト指向プログラミング言語処理系実装において開発した，並列オブジェクトのガーベジコレクション（GC）方式について述べた．並列オブジェクトモデルを分析し，mutator とセルを用いた単純なモデルに帰着させることで，ガーベジコレクタの設計を容易に正しく行うことができた．また，提案した GC 方式は，(1) 循環した参照を持つゴミも回収する，(2) 通常のプログラムの実行を長時間ブロックしない，(3) 通常のプログラム実行に課するオーバーヘッドが小さい，という特徴を持つ．

ハードウェアでスケジューリングを行うデータ駆動計算機では並行 GC を開発する必要があったが，提案した GC 方式では，オーバーヘッドの大きいリードバリア・ライトバリアを並列オブジェクトの内部に対して用いずに，個々の並列オブジェクトを単位として一

表 **1**　提案する GC 方式の特徴
Table 1　Features of the proposed GC scheme

	提案方式の選択	他の選択候補
基本アルゴリズム	検出・回収方式	参照カウント方式
ルートの処理	並行	一括
バリア方式	コピーバリア	消去バリア
バリア対象の単位	メッセージと並列オブジェクト	ノード，またはスロット
メッセージの処理	取り込み時	送信時

括処理を行うことで，並行 GC と低オーバーヘッドの両立を可能とした．また，通常計算時に完全に取り外せるアクティブバリアを提案している．その他，提案方式の特徴を表 **1** にまとめる．

電総研で開発されたデータ駆動並列計算機 EM-4 上での実装を行い，GC の動作を確認した．メッセージに色を付ける場合とメッセージに色を付けない場合とを比較すると，メッセージに色を付ける場合は，GC そのものの処理量は小さくなるが，通常計算へのオーバーヘッドがあり，GC が起こりにくいフェーズが長く続く場合は，メッセージに色を付けないほうがプログラム全体の実行時間は短縮されることが確認された．

一方，提案した GC 方式を，データ駆動計算機以外の（通信性能が劣り，リモート参照が少なく，スケジューリングがソフトウェアでなされる）メッセージパッシング型並列計算機向けに改良する場合は，次の点が有効であると考えられる：(1) ノードを跨った参照関係とノード内部の参照関係を区別して扱い，ノード外から参照されるセルをルートとしたノード内 GC を併用する．(2) スケジューリングキューの内部をアクセスを可能としたり，ノード内部の参照関係をできるだけ辿ることで（通常のプログラムの実行を長時間ブロックしないという性質は失われるものの）GC の処理の粒度を大きくし，またノード内のバリアを削減する．(3) マークや ACK の複数のメッセージを一つにまとめることで通信回数を削減する．

今後は，EM-4 の運用が終了したため，次世代機である EM-X に処理系を移植するとともに，今回評価しなかったメッセージ送信時のバリアを用いた方式との性能比較を行っていきたい．

謝辞　電総研の山口喜教，坂井修一（現筑波大学），佐藤三久（現 RWCP），児玉祐悦の各氏には，EM-4 の使用に際して技術的な点を含む貴重な御助言をいただいた．また，神戸大の鎌田十三郎助手には初期段階の原稿にコメントをいただいた．ここに深謝する．

参 考 文 献

1) Augusteijn, L.: Garbage Collection in a Distributed Environment, *Proc. of PARLE*, LNCS, Vol. 259, Springer-Verlag, pp. 75–93 (1987).
2) Barnes, J. and Hut, P.: A Hierarchical $O(N \log N)$ Force-Calculation Algorithm, *Nature*, Vol. 324, pp. 446–449 (1986).
3) Sakai, S., Yamaguchi, Y., Hiraki, K., Kodama, Y. and Yuba, T.: An Architecture of a Dataflow Single Chip Processor, *Proc. of International Symposium on Computer Architecture*, pp. 46–53 (1989).
4) Yasugi, M.: A Concurrent Object-Oriented Programming Language System for Highly Parallel Data-Driven Computers and its Applications, Technical Report 94-7e, Dept. of Information Science, University of Tokyo (1994).
5) Yonezawa, A.(ed.): *ABCL: An Object-Oriented Concurrent System*, The MIT Press (1990).
6) 鎌田十三郎, 松岡聡, 米澤明憲: 超並列計算機上の高効率な大域的ガーベジコレクション, 並列処理シンポジウム JSPP’94, pp. 33–40 (1994).
7) 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).
8) 八杉昌宏, 松岡聡, 米澤明憲: ABCL/EM-4：データ駆動並列計算機上の並列オブジェクト指向言語処理系の実装と評価, 情報処理学会論文誌, Vol.38, No. 9, pp. 1790–1799 (1997).
9) 松井祥悟, 田中良夫, 前田敦司, 中西正和: 相補型ガーベジコレクタ, 情報処理学会論文誌, Vol. 36, No. 8, pp. 1874–1884 (1995).
10) 湯浅太一: 実時間ごみ集め, 情報処理, Vol. 35, No. 11, pp. 1006–1013 (1994).

(平成 9 年 10 月 30 日受付)
(平成 10 年 3 月 31 日採録)

八杉　昌宏（正会員）

1967 年生．1989 年東京大学工学部電子工学科卒．1991 年同大学大学院電気工学専攻修士課程修了．1994 年同大学大学院理学系研究科情報科学専攻博士課程修了．1993 ~ 1995 年日本学術振興会特別研究員（東京大学，マンチェスター大学）．1995 年神戸大学工学部助手．1998 年より京都大学大学院情報学研究科通信情報システム専攻講師．博士（理学）．並列処理，言語処理系などに興味を持つ．日本ソフトウェア科学会，ACM 会員．