Vol. 35 No. 11　　情　報　処　理　　Nov. 1994

解　説

ごみ集めの基礎と最近の動向

6. 分　散　ご　み　集　め†

市　吉　伸　行††

1. はじめに

分散ごみ集め (GC) は分散環境の記号処理系—特に，メッセージ通信型並列計算機上の記号処理系(分散記号処理系)—におけるごみ集め (GC) である．分散メモリ並列計算機が普及しつつあるが，記号処理言語は，ユーザが分散メモリを直接に意識する必要がなく，また，低レベル同期処理や分散メモリ管理に煩わされることがないため，応用プログラム開発において魅力的な選択肢である．一方，処理系設計者側からすると，プロセッサを渡る参照の処理やプロセッサを跨るデータ構造の回収など新たな問題に対処しなければならない．ここでは，この後者について，基本的な方法を解説する．

逐次処理系における GC と同様，分散処理系の GC にも**一括型** (mark-and-sweep GC) と**即時型** (incremental GC) がある．一括型では，割付け可能なメモリ領域が乏しくなった時点で通常処理を中断し，すべての生きているデータを同定し，それ以外の領域を回収してデータ割付け可能な自由領域とする．即時型では，通常処理中にデータがごみになったことを積極的に検出し，その場で回収する*.

また，GC 処理がプロセッサ**に閉じた局所的なものであるか否か，という観点からの別の分類もあり得る．したがって，分散処理系における GC は大きく次の 4 つに分類できる．

	局所的	非局所的
一括型	局所一括 GC	大域一括 GC
即時型	局所即時 GC	プロセッサ間即時 GC

実際の処理系では，総合性能とごみ回収率の双方の要求を満たすべく，上記のうちのいくつかを組み合わせて実現することが多いであろう．

本稿では，**2.** で分散処理系にとって基本的な外部参照の表現について述べ，**3.** で局所 GC，**4.** でプロセッサ間即時 GC，**5.** で大域 GC について述べる．

2. 外部参照の表現

プロセッサを渡るポインタ（外部参照）の表現は，分散処理系設計の上で最も基本的なことの一つである．メッセージ通信型並列計算機では，システム内のアドレスは，プロセッサとその局所アドレスの対によって特定される．$\langle p, a\rangle$（p：プロセッサ番号，a：局所アドレス）という表現が最も直接的だが，プロセッサ内で外部から指されているデータの移動を全系同期なしに行うためには，間接表を用意し，$\langle p, e\rangle$（p：プロセッサ番号，e：間接表内エントリ番号）という表現を用いるのが便利である（移動時には間接表からのポインタを更新する）．

内部参照を含めたすべてのポインタを上記のいずれかの形で表現しても良いが，内部参照に関する処理のオーバヘッドを大きくしてしまう．したがって，内部参照は逐次処理系と同じ表現を用い，外部参照は特別に扱うほうが良いであろう*.

† Distributed Garbage Collection by Nobuyuki ICHIYOSHI (Systems Science Department, Mitsubishi Research Institute).

†† (株)三菱総合研究所システム科学部

* なお，本稿では触れないが，このほかに，通常処理プロセス (mutator) と GC 処理プロセス (collector) が並行に動く型の GC がある (on-the-fly, incremental tracing などと呼ばれる)．

** 単一ないし複数のプロセッサと局所メモリとが密結合された単位をノードと呼ぶことが多いが，ここでは，それを単にプロセッサと呼ぶものとする．

* 内部アドレスと外部アドレスの区別のない分散共有メモリアーキテクチャはこの限りでない．ただし，共有，分散によらず，一般に，並列処理への対応によって逐次処理部分が複雑化し，数パーセント程度の性能低下を被ることが多い．そこで，逐次処理部分はできるだけ逐次処理系方式をそのまま保ち，並列処理に必要な機能を外づけ的に付加するという考え方である．

3. 局所 GC

局所 GC (local GC) は，逐次処理系の GC と基本的に同じだが，以下のように，プロセッサを渡るポインタにともなう新たな処理が必要である．

（1） 外部（他プロセッサ）からのポインタにともなう処理

（i）外部から参照されているデータの同定

（ii）外部から参照されているデータの移動

（2） 外部（他プロセッサ）へのポインタにともなう処理

（i）外部へのポインタ回収時の処理

外部から参照されているデータは生きているので，ごみとして回収してはいけない．したがって，局所一括 GC の際のマーキングルートとして，そのようなデータを含める必要がある．そのために，外部からの参照を一括管理する**輸出表**を導入し，参照を外部に「輸出」するたびに登録するようにする．

局所一括 GC が複写方式や圧縮 GC である場合，生きているデータのアドレスが変わるが，通常処理中の他のプロセッサにアドレス更新を伝えることは非常に困難である．したがって，外部から指されているデータは移動しないようにするか(そのような GC 方式とするか，GC 時に移動をともなわない特別な領域を設ける)，または，**2.** で述べたような間接表を導入して，直接にアドレスを指せないようにする．前述の輸出表の導入はこの目的のためにも用いることができる．

一括および即時局所 GC によって，外部参照ポインタがごみとして回収されることが当然あり得るが，参照先にそれを伝えることにより参照先プロセッサでのごみの回収につながる．

なお，輸出表には，外部参照一つに一つのエントリを対応させるか，同じ参照先ごとに一つのエントリを対応させるか，などがあり得，局所 GC 方式や処理／メモリ・オーバヘッドとの関連で選択する．

3.1 局所 GC の特徴

局所 GC は，逐次処理系の GC 方式がほとんどそのまま使えること，各プロセッサが必要に応じた頻度で GC を行え，その間，他のプロセッサが通常処理を継続できること，などの長所がある．よって，分散処理系におけるベースの GC として採用するのに適当である．しかし，局所 GC のみでは外部参照されているデータを回収できないため，プロセッサ間即時 GC（局所 GC 時にごみとなった外部参照先を回収）や大域 GC と組み合わせることが必要である．別の問題として，作業分担型でなく協調型のプログラムでは，一つのプロセッサが局所一括 GC に入ったために，全体の処理が遅くなることがある．このため，プログラムによっては，性能を保つために，局所 GC を一斉に起動するなどの対策が有効である．

4. 即 時 GC

即時 GC (incremental GC) では，生きているデータへの参照がなくなったことを通常処理中に検出する必要がある．逐次処理系では，被参照データに参照カウント・フィールドを設ける参照カウント方式が一般的である．分散処理系の場合，参照カウントの増減をメッセージで行うことになる（インクリメントおよびデクリメント・メッセージ）が，メッセージの遅延にともなう問題がある．たとえば，外部参照を分割して他プロセッサに与える際に，参照先に送るインクリメント・メッセージの到着を確認せずに分割した参照を渡してしまうとしよう．このとき，外部参照を渡された側のプロセッサで参照がただちにごみになり，参照先に送ったデクリメント・メッセージがインクリメント・メッセージより先に到着し，その結果，まだ生きているデータが誤ってごみと認識されてしまう可能性がある．これは，実際の参照数と参照カウントが食い違っているクリティカルリージョンを，他プロセッサから正しく守らなかったために起きている．参照数を安全に更新するためには，インクリメント・メッセージへの返答を受信した後に，参照のコピー（複写）を他のプロセッサに渡す必要があるが，オーバヘッドが大きい．

このように，単純な参照カウント方式は分散環境では問題がある．以下に，並列処理系向きの二つの即時 GC 方式を紹介する．

4.1 1ビット参照カウント方式

1ビット参照カウント方式[6] では，参照側がそれが単一参照であるか否かを示す多重参照ビットをもつ．処理系は，ポインタの多重参照ビットが

立っていないときは，それが単一参照であること，すなわち，参照先データへのポインタが当該ポインタのみであることを保証する．そのようなポインタを捨てると参照先はごみとなる．多重参照ビットに関する処理がすべてポインタ側で局所的に行えるのが特徴である．回収時を除いて，参照先にメッセージを送る必要はない．

1ビット参照カウント方式では，データ生成時には多重参照ビットは落ちており，いったん立つと多重参照ビットは決して落ちず，そのため，通常の参照カウント方式と比べると，回収されないごみの割合が大きい．したがって，一括型の GC と組み合わせることが必要である．1ビット参照カウント方式の意義は，ごみの溜る速度を減らすことによって一括 GC の頻度を下げることにあると言える．

1ビット参照カウント方式を並行論理型言語に適用した MRB 方式では，単一書込み・単一読出しにおける通信データの即時回収や単一参照データの破壊的更新などの最適化を可能にしている[5]．

4.2 重み付き参照カウント方式

通常の参照カウント方式では，参照ポインタを複写するときに参照の増加を参照先に報告していた．ところが，参照ポインタの複写は参照の増加としてではなく，参照の分割と捉えることもできる．すなわち，参照ポインタには参照の重みが付随していて，参照の複写時に重みを分割するのである．たとえば，初めに参照ポインタの重みを100とすると，参照の分割のときに，50ずつに重みを分ける．一方，参照を捨てるときには参照減メッセージによって参照の重みを返却する．これを重み付き参照カウント方式 (Weighted Reference Counting, WRC) と呼ぶ[3),12)]．

通常の参照カウント方式では，

(A) (データ X の参照カウント)＝
(X への参照ポインタの数)

が成り立ったが，WRC 方式では，メッセージ中の参照を含むすべての参照に正の参照重みを与え，

(B) (データ X の参照重み)＝
(X への参照ポインタの参照重みの和)

を保つようにする (**図-1**)．これにより，被参照データ X に関して，X への参照ポインタが存在しないことと，X の参照重みがゼロであること，は同値になる．通常の参照カウント方式は，WRC 方式においてポインタの参照重みを1に制限したものとみなせる．WRC 方式が通常の参照カウントより優れている点は，1ビット参照カウント方式と同様，参照ポインタを分割する際に被参照側のカウントを更新する必要がないことである．したがって，インクリメント・メッセージは不要である．通常の参照カウント方式では，上記(B)の条件は通信遅延によって壊されないため，参照カウント方式であったような誤った回収の恐れはない．一方，1ビット参照カウント方式と違い，ごみの回収率は損なっていない．

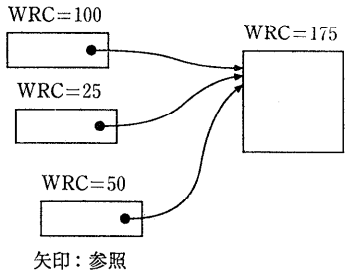


図-1 重み付き参照カウント

WRC 方式の実装に当たっては，

(1) 重みの表現
(2) 初期の重み
(3) 重みの分割の方法
(4) 重み1の参照の分割の方法

を決めなくてはならない．参照重みの分割は2等分が自然であろう．また，重みを必ず2等分することにすると，参照側では底を2とする対数によってコンパクトに重みを表現できる☆．

重みが1となった参照の分割は，重みがそれ以上分割できないので，なんらかの特別な処理が必要となる．元のポインタへの間接ポインタを新たな参照とする方式と参照先から新たに参照重みをもらう方式とがある．重み1の参照は通常の参照カウント方式での参照と似ており，参照重みの補給を要求するメッセージとその返答はインクリメント・メッセージとその返答に対応する．後者では，参照分割の中断処理が必要になること，およ

☆同一外部参照の複数回の「輸入」をまとめるための**輸入表**を各プロセッサに置くようにすると，参照側でも重みの加算が必要となるので，対数表現はできなくなる．しかしながら輸入表方式では，外部参照当たり数ワードを費やすため，数ビットの節約はあまり意味がないであろう．

び，被参照側の重みに上限を設けられなくなるという問題点があり，普通は単純な前者を採用するほうが良いであろう．

4.3 即時 GC の特徴

即時 GC は，非常に大きなデータ構造を一度に回収しようとしない限り，それによって通常処理が長く中断することがない．また多くの場合，通常処理で扱っているデータを操作するため局所性に優れるという長所がある．

しかしながら，循環構造は回収できず，さらに，圧縮せずにデータの割付け，回収，再割付けを繰り返すと，メモリ領域のフラグメンテーションが生じて，通常処理の局所性が徐々に減っていくという問題などがある．また，処理系開発の上でも，回収したメモリ領域の再割付けのためのフリーリスト管理が必要であることや，ごみの同定・回収処理が加わるために通常処理が複雑化・低速化するという問題がある．

分散処理系においては，循環構造がごみになることが稀な状況では，局所 GC を一括型で行い，プロセッサ間 GC を即時式で行うという組合せは，大いにあり得る選択であろう．たとえば，第五世代コンピュータ・プロジェクトで開発した Multi-PSI/PIM 上の KL 1 処理系[13]では，一括型（複写方式）と即時型（MRB 方式）の局所 GC および即時型（WRC 方式）の局所 GC[7]を組み合わせ，大域一括 GC は実装していない．

5. 大域一括 GC

プロセッサを跨る循環構造を回収するためには，大域一括 GC（ないし，単に大域 GC (global GC)）の実現が必要となる．

5.1 大域 GC の流れ

大域 GC の処理の流れはおよそ次のとおりである．

1. 大域 GC の起動
2. マーキング処理
3. メモリ領域再初期化
4. 通常処理への復帰

以下，各段階について順を追って説明する．

5.2 起　動

大域 GC を始めるタイミングは，処理系に都合の良いように決めれば良い．たとえば，あるプロセッサで（局所 GC 後に）割り付けられるデータ領域がなくなったら，大域 GC を起動する．このような大域 GC 起動要求は非同期に発生するが，大域 GC の流れを指揮するマスタプロセッサを一つ固定しておいて，そこで処理を逐次化すれば良い．

マスタプロセッサは，大域 GC 要求を受信すると，全プロセッサに通常処理停止の指令をブロードキャストする．ブロードキャストメッセージを受信したプロセッサは，マスタプロセッサに ACK メッセージを返すとともに，通常処理を中断して待ち状態に入る．待ち状態の間に到着する通常処理用メッセージ（プロセッサ間データ読出し／書込み，プロセス移動などのためのメッセージ）は，処理せずにバッファ領域に複写しておく．マスタプロセッサは，全プロセッサからブロードキャストメッセージへの ACK メッセージを受信した時点で，通常処理が中断したことが分かるので，マーキングフェーズ開始指令をブロードキャストする．

ここで問題となるのは，ネットワークに残っているかもしれない通常処理メッセージである．GC 起動前に送信されたが，まだ宛先プロセッサに到着していないメッセージがあるかもしれないのである．ネットワーク内にメッセージが存在しないことをハードウェアの機能として検出できない場合は，ソフト的に確認する必要がある．メッセージのネットワーク内滞在時間にあらかじめ知られた上限があればその時間だけ待てばよいが，ネットワークは輻輳する可能性があり，一定時間内のメッセージ到着は一般に保証されない．メッセージ消滅をソフト的に保証する一つのやり方は，なんらかの分散終了検出アルゴリズムを用いることである．しかしながら，分散終了検出アルゴリズムは，メッセージカウントの管理などのオーバヘッドを通常処理に負わせてしまう．通常処理に変更を加えずにメッセージ消滅が保証できることが望ましい．そのために，以下のような方法が考えられる．メッセージが通過するチャネルの数にはメッセージによらない上限 d（ネットワークの直径）がある．もし，すべてのチャネルに一斉に掃き出し用の特別なメッセージを流すことができれば，それを d 回繰り返した後には，すべてのメッセージをネットワークから掃き出すことができる．

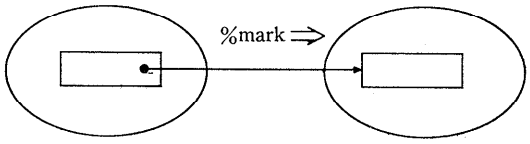


図-2 マーキングメッセージ

5.3 マーキング

すべての生きているデータを再帰的に同定するフェーズである．マーキングは全プロセッサが協力して，並列処理として行う．プロセッサ内マーキングは，基本的に逐次 GC のマーキングの方法を用いれば良い．プロセッサを渡る参照はマークメッセージによって追っていく（図-2）．マーキングルートは，各プロセッサ内のスケジューリング・キューにつながれたプロセス（オブジェクト，関数インスタンス，ゴールなど）および未処理メッセージ中のポインタである．また，他のプロセッサから送られてくるマークメッセージの参照先もプロセッサ内マーキングの出発点となる．

各プロセッサは，マーキング・フェーズ中に，マークメッセージを取り込まなくてはいけないが，ポーリング型受信 (receive プリミティブでの受信) の場合，メッセージ受信バッファが溢れないように，定期的に受信バッファをポーリングする必要がある．割り込み型受信（能動メッセージなど）の場合は，到着したメッセージがマーキングルートに自分の参照先を追加するようにする．その場合でも，外部からのマーキングルート領域が溢れない注意が必要であろう．(分散 GC のメッセージバッファ領域の管理については，参考文献 2) を参照)．

大域的なマーキングの終了は，局所的には判定できず，なんらかの分散終了検出アルゴリズムを用いる．分散終了検出は分散アルゴリズムのエッセンスとも言え，エレガントな方式の考案，問題や仮定の定式化，正しさの証明など研究者を引き付ける要素に富んでおり，多くの方式が提案されている．研究し尽くされた感もあるが，読者が独自に考えてみるのも楽しいであろう．(前述の WRC を応用した方式に参考文献 9), 8) がある．11) は GC 方式との対応の観点から，各種方式をサーベイとしている．終了検出を含む分散アルゴリズム全般については，4)が標準的である)．

なお，生きているデータの同定のために，各プロセッサで複写方式の GC を用いても良い．ただし，複写によって生きているデータが新空間に移動するので，輸出表方式でない場合，アドレス更新メッセージを参照側に送って移動先アドレスを知らせる必要がある．

5.4 メモリ領域の再初期化と通常処理の再開

マーキングフェーズが終了すると，マスタプロセッサはメモリ再初期化指令をブロードキャストする．それぞれのプロセッサで独立にメモリ管理情報を再初期化する（ヒープトップポインタの再設定，フリーリストの再初期化など)．

全プロセッサのメモリ再初期化終了を確認したマスタプロセッサは，通常処理再開指令をブロードキャストし，各プロセッサは指令を受信すると通常処理を再開する．(通常処理を再開したプロセッサの送信するメッセージが再開指令の到着前に着く可能性もある．受信したプロセッサは通常処理を再開し，後から到着する再開指令を無視するか，あるいは，再開指令到着前の通常処理メッセージをバッファしてもよい)．

5.5 大域 GC の特徴

大域 GC では，外部参照数に比例するメッセージが飛び交うため，通常処理が長く中断する．また，GC 処理の負荷のバランスによって，その性能が左右されるであろう．(極端な場合，仮に複数のプロセッサを何周もするような生きたデータ構造があると，それがクリティカルパスとなる)．このように，大域 GC のみだと，システム全体の GC オーバヘッド率が大きくなると予想されるため，いわば，最後の手段として行うのが好ましいであろう．(たとえば，大域 GC のオーバヘッドがシステム処理全体の数パーセント以内になるように GC を組み合わせ，また，パラメータをチューンすることができれば良いと思われる．)

大域 GC が大域的にメモリをスイープするという短所を逆手にとると，大域情報を利用した処理を「ついでに」行うようなことも考えられる．たとえば，ごみになったプロセスは永久に中断していることが分かるので，デバッグ情報に反映できる．また，大域 GC 時にオブジェクトのマイグレーションを行うという方式が参考文献 10) で提案されている．

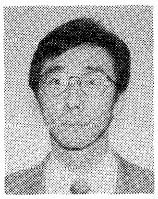
6. おわりに

分散 GC の基本的な手法を紹介した．On-the-fly 型の GC や分散システムにおけるフォールトトレラントな方式なども含む詳しいサーベイとして参考文献 1) があり，参考となろう．本稿に対して貴重なコメントを下さった，元同僚 ICOT 六沢一昭氏に感謝したい．

参 考 文 献

1) Abdullahi, S. E., Miranda, E. E. and Ringwood, G. A.: Collection Schemes for Distributed Garbage, In *Proceedings of the International Workshop on Memory Management*, pp. 43-81, Springer LNCS 637 (1992).
2) Ali, K. A. M. and Haridi, S.: Global Garbage Collection for Distributed Storage Systems, *International Journal of Parallel Programming*, 15 (5): 339-387 (1986).
3) Bevan, D. I.: Distributed Garbage Collection Using Reference Counting, In *Proceedings of Parallel Architectures and Languages Europe*, pp. 176-187 (June 1987). Also in *Parallel Computing*, 9(2) pp. 179-192 (1989).
4) Chandy, K. M. and Misra, J.: *Parallel Program Design: A Foundation*, Addison-Wesley (1988).
5) Chikayama, T. and Kimura, Y.: Multiple Reference Management in Flat GHC, In *Proceedings of the Fourth International Conference on Logic Programming*, pp. 276-293 (1987).
6) Friedman, D. P. and Wise, D. S.: The One-Bit Reference Count, *BIT*, (17): 351-359 (1977).
7) Ichiyoshi, N., Rokusawa, K., Nakajima, K. and Inamura, Y.: A New External Reference Management and Distributed Unification for KL 1, *New Generation Computing*, 7 (2 & 3): 159-177 (1990).
8) Mattern, F.: Global Quiescence Detection Based on Credit Distribution and Recovery, *Information Processing Letters*, 30 (4): 195-200 (Feb. 1989).
9) Rokusawa, K., Ichiyoshi, N., Chikayama, T. and Nakashima, H.: An Efficient Termination Detection and Abortion Algorithm for Distributed Processing Systems, In *Proceedings of the 1988 International Conference on Parallel Processing, Vol. I Architecture*, pp. 18-22 (1988).
10) Taura, K., Matsuoka, S. and Yonezawa, A.: Incorporating Locality Management into Garbage Collection in Massively Parallel Object-Oriented Languages, In *JSPP '93*, pp. 277-282 (1993).
11) Tel, G. and Mattern, F.: The Derivation of Distributed Termination Detection Algorithms from Garbage Collection Schemes, *ACM Transactions on Programming Languages and Systems*, 15(1) (1993).
12) Watson, P. and Watson, I.: An Efficient Garbage Collection Scheme for Parallel Computer Architectures, In *Proceedings of Parallel Architectures and Languages Europe*, pp. 432-443 (June 1987).
13) 瀧 和男(編): bit 別冊：第五世代コンピュータの並列処理，共立出版 (1993).

(平成 6 年 4 月 25 日受付)



市吉 伸行（正会員）

1955 年生．1979 年東京大学理学部情報科学科卒業．1981 年同大学院理学系研究科修士課程修了．1982 年(株)三菱総合研究所入社．1984 年より 1 年間米国バテル研究所にて人工知能を研修．1987 年より 1992 年まで(財)新世代コンピュータ技術開発機構(ICOT)に出向．論理型言語並列処理系，並列プログラムの研究開発に従事．その後，RWCP 超並列 MRI 研究室にて並列プログラミング環境の研究，汎用機向け LK1 処理系 (KLIC) 共有ヒープGC の研究等に従事．