プログラミング 8－1
（1996. 8. 28）

広い仮想記憶空間を利用した分散ガーベジコレクション

小出 洋
koide@koma.jaeri.go.jp
日本原子力研究所 計算科学技術推進センター
〒153 東京都目黒区中目黒 2-2-54

分散環境下でオブジェクトの動的領域確保を行ないたいという要請は高まっている．このとき自動領域管理機構としてのガーベジコレクタは必須である．本論で提案する新しい分散ガーベジコレクタ DC1 は，各プロセッサの局所領域として広い仮想アドレス空間が使用できるという前提で，分散環境のスケーラビリティを損なわず，プロセッサ間にわたるサイクルを含むすべてのデータ構造を回収でき，実行時のオーバーヘッドもない．基本的には，Hughes の方法を採用しているが，それと比較すると，タイムスタンプの伝搬を行なわない．このため，各オブジェクトに必要だったタイムスタンプを格納するための領域が不要であり，局所コレクションに効率的なコピー方式を採用できる．

A Distributed Garbage Collection based on Large Virtual Address Space

Hiroshi Koide
Center for Promotion of Computational Science and Engineering
Japan Atomic Energy Research Institute
2-2-54, Nakameguro, Meguro-ku, Tokyo, 153 Japan

In distributed environments, dynamic object allocation is strongly expected. At this time, garbage collection as an automatic memory management system is indispensable. Under the condition that every processor has a large virtual address space as its own local space, a new distributed garbage collector DC1 proposed in this paper does not spoil the scalability of distributed memory multiprocessors, has an ability to collect all types of data structures of garbage including cycle among several processors, and does not have any run-time overheads. Basically, the proposed collector employs Hughes's method, but does not propagate time-stamps over reachable objects. Therefore, the proposed collector does not require a space to restore a time-space in every object, and can employ an efficient copying collector as a local collector.

1 はじめに

オブジェクト指向型言語の使用，記号処理の必要性，オープン環境 (例えばインターネット環境) の利用などの要請のため，オブジェクトの動的領域確保の重要性が高まっている．

また，大規模アプリケーションの効率的な実行の要請のため，複数プロセッサを使用して計算機の性能の向上が行なわれている．特に，プロセッサ間で記憶領域を共有せず，プロセッサ間通信をネットワークを使用して行なう分散型マルチプロセッサに関する研究は，そのスケーラビリティの高さのため盛んである．

分散環境下で実行される大規模アプリケーションは，多くのモジュールに分割されて管理開発される．これらのモジュールが互いに協調して計算を進めることで，本質的な仕事が行なわれる．分散環境下でのアルゴリズムの記述は複雑なので，適当なツールが存在しない現在，逐次環境下より困難である．

そのため，分散環境下でのプログラム開発においては,「プログラマは本質的なアルゴリズムの記述に専念すべきで，オブジェクトの確保解放は自動的に行なわれた方が良い」という自動記憶領域管理機構の一般的考えはより広まっている．分散環境下で，オブジェクトの動的領域確保を行なうとき，自動領域管理機構としての ガーベジコレクタは必須である [1, 2, 3, 4, 5].

分散環境化におけるガーベジコレクタを考えるとき，分散ガーベジコレクション全体を管理する特別なマスタプロセッサを用意するアルゴリズムでは，分散型マルチプロセッサのスケーラビリティを損ない，望ましくない．過去に行なわれた分散環境化におけるマスタプロセッサが不要な分散ガーベジコレクタは，参照カウンタ方式を分散化したもの [3, 4, 5] とマーク·アンド·スィープ方式やコピー方式を分散化したもの [1, 2, 5] にわかれる．

参照カウンタ方式では，各オブジェクトに参照カウンタ (他のオブジェクトからの参照数) を保持する．他のオブジェクトからの参照数が変化する操作を行なうときは，いつも参照カウンタの増減を行なう．このとき，参照カウンタが 0 のオブジェクトは使用されていないオブジェクト (ガーベジ) と判断されすぐに回収することができる．

参照カウンタ方式を利用した多くの分散ガーベジコレクタが提案されている．この方式では，本来の計算時に参照カウンタを操作するための計算時間が必要であり，各オブジェクトに参照カウンタを記憶するための領域が必要である．また，原理的にサイクルを含む構造を構成するガーベジを回収できない．

参照カウンタ方式の欠点を解決するための別の方法として，マーク·アンド·スィープ方式やコピー方式を分散化したものが提案されている．この中で代表的なもののひとつに，Hughes の方法 [1] がある．

Hughes の方法では，すべてのプロセッサから参照できる大域クロック (global clock) を用意する．各オブジェクトは生成された時刻で初期化されたタイムスタンプを持つ．現在時刻のタイムスタンプを持つ各プロセッサの *root* から，すべての使用中オブジェクトにタイムスタンプの伝搬が行なわれる．使用中オブジェクトのタイムスタンプは単調増加するのに対し，ガーベジのタイムスタンプはある値に留まる．使用中オブジェクトの最も古いタイムスタンプ値を分散アルゴリズムを使用して求め，その値より古いオブジェクトをガーベジとみなす．

Hughes の方法は，参照カウンタ方式と異なり，実行時の参照カウンタを増減するための計算と各オブジェクトに参照カウンタを記憶するための領域を必要としないし，プロセッサ間にわたるサイクルを含むガーベジを回収できる．しかし，大域クロックとタイムスタンプを記憶するための領域が必要である．また，タイムスタンプの伝搬はマーク·アンド·スィープ方式を利用して行なうため，分散ガーベジコレクション 1 サイクルの計算時間は記憶領域に比例する．

本論で提案する方式 (DC1 と呼ぶ) は，Hughes の方法における大域クロックとタイムスタンプの代わりに，各プロセッサが持つ

広いページ方式の仮想アドレス空間 [6] を利用する．参照カウンタ方式と異なり，実行時の参照カウンタを増減するための計算も各オブジェクトに参照カウンタを記憶するための領域も不要であり，プロセッサ間にわたるサイクルを含むガーベジを回収できる．大域クロックが不要であるため，タイムスタンプを記憶するための領域も不要である．また，タイムスタンプの伝搬も行なわないため，コピー方式 [7, 8, 9] を局所コレクタに使用した場合，分散ガーベジコレクション 1 サイクルは，使用中オブジェクトの総容量に比例する計算時間だけで済む．

本論では，各プロセッサが広いページ方式の仮想アドレス空間が利用できるという条件で，実装可能な分散ガーベジコレクタを提案した．以降，2 章では Hughes の方法を概観し，3 章で DC1 のアルゴリズムを述べる．4 章で DC1 と Hughes の方法を比較し，4 章でまとめを行なう．

2 Hughes の方法

本章では，Hughes の方法 [1] について簡単に述べる．

Hughes の方法では，つぎの想定のもとで，各プロセッサは，局所領域に対して局所ガーベジコレクションを行なう．また，すべてのプロセッサが協調して，分散ガーベジコレクションも行なう．

(1) 各プロセッサは，タイムスタンプを記憶するための領域を持つ．

(2) すべてのプロセッサはタイムスタンプに記憶するための大域クロックを参照できる．

(3) プロセッサ間通信の遅延は有限時間で済む．

各プロセッサの記憶領域の構成を図 1 で示す．Entry 表と Exit 表の各エントリは普通のオブジェクトと同様に生成時刻で初期化され

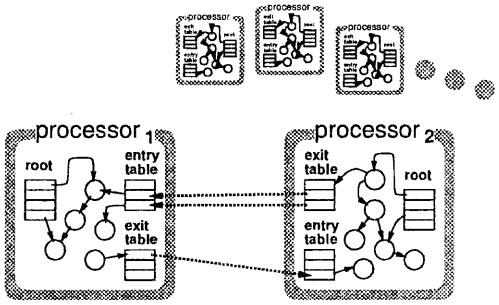

図 1: Hughes の方法における記憶領域の構成.
Figure 1:Memory Organization of Hughes's Method.

たタイムスタンプを持つ．各プロセッサの *root* は，いつも現在時刻のタイムスタンプを持つ．

各プロセッサの局所コレクタは，*root* と Entry 表から，使用中オブジェクトのリスト構造を探索して，印付けと同時にタイムスタンプの伝搬 (リスト構造全体にわたり，親 オブジェクトのタイムスタンプより子オブジェクトのタイムスタンプが古かったら親と同じ値にする作業) を行なう．リスト探索が Exit 表に行きついたら，それ以上の探索は行なわない．

root と Entry 表からのタイムスタンプの伝搬が終了したとき，*root* から到達可能な Exit 表のエントリのタイムスタンプは，局所コレクションの開始時刻に設定され，Entry 表から到達可能な Exit 表のエントリのタイムスタンプは，Entry 表のタイムスタンプの値に設定される．

局所コレクションの最後の段階で，Exit 表の各エントリについて，そのエントリが指す Entry 表のエントリのタイムスタンプを更新するために，メッセージを対応するプロセッサに送る．

分散ガーベジコレクションでは，すべてのプロセッサが協調して，すでに伝搬が終ったすべてのタイムスタンプのうちで最も古い値 (Global Threshold) を求める．Global Threshold より古い値の タイムスタンプを持つオブジェクトはガーベジである．局所コレクションが起動されてから後に生成されたオブジェクトは，生成時刻のタイムスタンプを持つの

で使用中と判断される．

各プロセッサは，Entry 表から完全に伝搬の終っていないタイムスタンプをすべて記録しておき，*root* と Entry 表から完全に伝搬の終っていない最も古いタイムスタンプを表す変数 *redo* をいつも計算しておく．各プロセッサにおいて，この変数 *redo* はつぎの性質を持つ．

- *redo* より古い値を持つすべてのタイムスタンプの伝搬はすでに終っている．
- *redo* は局所コレクション終了直後には，現在時刻に再設定される．この値は，そのプロセッサの局所領域におけるすべてのタイムスタンプと等しいか新しい．
- すべての使用中オブジェクトは *root* か *redo* のタイムスタンプのうちで古いものより等しいか新しいタイムスタンプを持つ．

以上の性質から，すべての *redo* のうちで最古のもの (*minredo* と呼ぶ) より古いタイムスタンプを持つオブジェクトはガーベジである．逆に使用中オブジェクトのタイムスタンプは，*minredo* と等しいか新しいタイムスタンプを持つ．すべてのプロセッサは，次節に示すアルゴリズムを使用して，*minredo* の値を計算する．

2.1 *minredo* を求めるアルゴリズム

このアルゴリズムは Rana の分散停止問題 [10] の解と同様に行なう．

[10] と同様に，すべてのプロセッサは仮想的に (または実際に) ハミルトン-サイクルで結ばれている．すなわち，すべてのプロセッサは successor を持ち，各メッセージは一方方向のサイクルを通ってすべてのプロセッサに伝えられる．また，各プロセッサは大域クロックを参照でき，通信の遅延は有限時間で済むことを想定する．

(1) あるプロセッサにおいて，*redo* を *minredo* より大きな値に更新する場合，そのプロセッサは 3 つ組 (現在時刻 t, プロセッサ番号, *redo* の新しい値) を successor に送る．

(2) 3 つ組を受けとったプロセッサは，時刻 t から現在時刻までの間でそのプロセッサでの *redo* の最小値を求める (*redo* の値は単調増加しない)．

つぎに，新しく求めた *redo* の値が 3 つ組中の *redo* の値より古かったら，新しく求めた *redo* の値とそのプロセッサ ID を採用し，そうでない場合は 3 つ組の内容を変更せずに，次の successor に 3 つ組を転送する．

なお，新しく求めた *redo* の値が，すでに計算されている *minredo* の値より古かったら，次の successor に 3 つ組を転送する必要はない．

(3) 3 つ組が発信元に戻ってきたとき，その 3 つ組中の *redo* の値が *minredo* の新しい値である．

(4) すべてのプロセッサに新しい *minredo* の値を知らせる．

3 DC1 のアルゴリズム

本章では，DC1 の詳しいアルゴリズムについて述べる．DC1 では，Hughes の方法と異なり，各オブジェクトのタイムスタンプのための領域と大域クロックを必要としない．しかし，Hughes の方法と同様に，プロセッサ間の通信の遅延は，有限時間で済むことを仮定している．また，DC1 では，各プロセッサの局所領域が独立した広い仮想アドレス空間 [6] (例えば，そのアドレス長が 64 ビット) として，利用できることを想定している．

3.1 分散コレクタ

分散コレクタは，Global Threshold(M と呼ぶ) アドレスのより大きな領域に，すべての使用中オブジェクトをコピーする (図 2)．

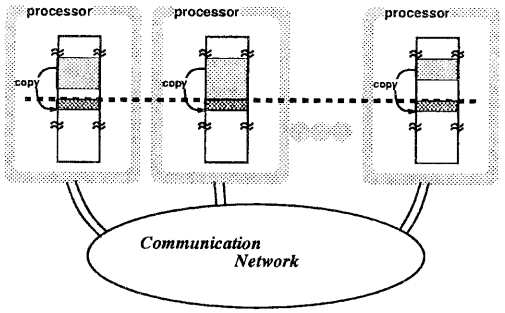

図 2: 分散環境における DC1 の実行.
Figure 2: Execution of DC1 on a Distributed Environment.

(1) あるプロセッサで局所コレクションを行なっても，充分な大きさの新しい記憶領域を確保できないとき，そのプロセッサは，2 つ組 (アロケーションポインタ u, プロセッサ ID) を successor に送る.

(2) 2 つ組を受けとったプロセッサは，実質的な計算を一時停止して，現在実行中の局所コレクションが終ったときのアロケーションポインタの値 u' を計算する.

そのプロセッサで局所コレクションを行なっていなければ，そのときのアロケーションポインタの値が u' である.

また，つぎのように u と u' の比較も行なう.

- $u \geq u'$ の場合，プロセッサはつぎの successor にメッセージを変更しないで転送する.
- $u < u'$ の場合，プロセッサは u' とプロセッサ ID の 2 つ組を successor に送る.

(3) 2 つ組がすべてのプロセッサから転送されて発信元プロセッサに戻ってきたとき，その 2 つ組中の u が新しい M である.

(4) 発信元プロセッサは，すべてのプロセッサに新しい M を知らせる.

この段階で，各プロセッサは最後の局所コレクションか分散コレクションがつくった送信バッファの内容を対応するプロセッサに送る.

(5) 各プロセッサはすべての使用中オブジェクトを M よりも大きなアドレスにコピーする．このとき，不要になった領域の物理ページの再利用を行なう.

この段階で，各プロセッサの受信バッファにはそのプロセッサの Entry 表のエントリへのポインタの集合が入っている．この集合にない Entry 表のエントリはガーベジである.

コレクタがリモートポインタをコピーするときは，次回の分散コレクションを行なうために，そのリモートポインタの内容を送信バッファに入れておく.

(6) 各プロセッサは，実質的な計算を再開する.

4 DC1 と Hughes の方法の比較

Hughes の方法では，すべてのオブジェクトにタイムスタンプを格納するのに充分なビット長の領域が必要である．応用プログラムに必要なオブジェクトの個数は，実行前にわからないのが普通であるから，このビット長をあらかじめ決めておくことは困難である．したがって，Hughes の方法を使用する場合，この領域を充分な余裕をもって確保しておくことになる．DC1 では，このような領域は不要である.

Hughes の方法では，局所コレクションの伝搬段階終了後，フリーリストを作るために記憶領域全体を走査するため，記憶領域の大きさに比例する計算時間が必要である．DC1 の局所コレクタは，使用中オブジェクトをよりアドレスの大きな領域にコピーするだけで済むため，使用中オブジェクトの大きさに比例する計算時間で済む.

DC1 では，新しいオブジェクトの割り当ても単にアロケーションポインタの値を増加させるだけで済むため，フリーリストの管理が必要な Hughes の方法よりも領域割り当てにようするコストは小さい．

5 まとめ

広い仮想アドレス空間を持つ共有メモリ型並列計算機で効率的な領域確保法がすでに提案されている．DC1 とこの領域確保法を組み合わせて使用することにより，共有メモリ型並列計算機クラスタで計算を効率良く行なう環境を構築することができると思われる．

DC1 では，局所コレクタにコピー方式 [7, 8, 9] を採用しているが，このコピー方式が実時間型コピー方式 [11] なら，実質的な計算の停止時間の上限を短くすることができる．実時間型コピー方式コレクタの DC1 への適用は，今後の課題である．

DC1 は，Hughes の方法よりも単純であり，その計算時間も短いため，Hughes の方法と比較して実装も容易と考えられる．DC1 の実際の分散環境での評価についても，今後の課題である．

参考文献

[1] Hughes, J.: A Distributed Garbage Collection Algorithm, *Proc. FPLCA '85*, LNCS 201, pp. 256-272 (1985).

[2] Ali, K. A. M. and Haridi, S.: Global Garbage Collection for Distributed Heap Storage Systems, *International Journal of Parallel Programming*, Vol. 15, No. 5, pp. 339-387 (1986).

[3] Bevan, D. I.: Distributed Garbage Collection Using Reference Counting, *Proc. PARLE'87*, LNCS 259, pp. 117-187 (1987).

[4] Watson, P. and Watson, I.: An Efficient Garbage Collection Scheme for Parallel Computer Architecture, *Proc. PARL'87*, LNCS 259, pp. 432-443 (1987).

[5] Plainfossé, D. and Shapiro, M.: A Survey of Distributed Garbage Collection Techniques, *Proc. IWMM'95*, LNCS 986, pp. 211-249 (1995).

[6] Koide, H., Suzuki, M. and Nakayama, Y. : A new memory allocation scheme for the shared memory parallel computer with a vast virtual memory space, *Technical Report CSIM-95-5*, The University of Electro-Communications, Japan (1995).

[7] Fenichel, R. and Yochelson. J.: A LISP Garbage-Collector for Virtual-Memory Computer Systems, *Comm. ACM*, Vol. 12, No. 11, pp. 611-612 (1969).

[8] Baker, H. G.: List Processing in Real Time on a Serial Computer, *Comm. ACM*, Vol. 21, No. 4, pp. 280-294 (1978).

[9] Appel, A. W., Ellis, J. R. and Li, K.: Real-time Concurrent Garbage Collection on Stock Multiprocessors, *Proc. PLDI'88*, pp. 11-20 (1988).

[10] Rana, S. P.: A Distributed Solution of the Distributed Termination Problem, *Inf. Process. Lett.*, Vol. 17, pp. 43-46 (1983).

[11] Tanaka, Y., Matsui, S., Maeda, A. and Nakanishi, M.: 'Partial Marking GC', in *Proc. CONPAR '94/VAPP VI, Lecture Notes in Computer Science*, Vol. 854, pp. 337-348, Springer-Verlag (1994).