プログラミング－言語・基礎・実践－ 13－16
（１９９３．８．２０）

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

鎌田十三郎 松岡 聡 米澤 明憲
東京大学理学部情報科学科

非同期ネットワークで結合された、分散メモリを持つ高並列計算機上のガーベジコレクション(GC)では、ノード境界を越えた参照や未到着メッセージのため、オブジェクトの参照関係の決定は難しい。本研究では、(1)参照の局所性を考慮した、(2)ノード境界を越えたマークを効率的に行なう、(3)応用プログラムの実行を長時間中断しない、大域的GCのアルゴリズムを提案している。その際、GCのためのメッセージを減らすため、終了判定に重みを用い、また未到着メッセージの到着確認法として、メッセージ数をカウントする方式とブルドージングを行なう方式を提案している。並列計算機AP1000上で実装を行ない、アルゴリズムの評価・2方式の比較を行なった。

Efficient Global Garbage Collection on a Massively Parallel Computer

Tomio KAMADA Satoshi MATSUOKA Akinori YONEZAWA
Dept. of Information Science, The University of Tokyo

On distributed-memory high-performance MPP's where processors are interconnected by an asynchronous network, efficient Garbage Collection (GC) becomes difficult due to inter-node references and references within pending, unprocessed messages. Our parallel global GC algorithm (1) takes advantage of reference locality, (2) efficiently traverses references over nodes, and (3) admits minimum pause time of ongoing computations. The algorithm employs global weight counting scheme to substantially reduce message traffic. The two methods for confirming the arrival of pending messages are available: one counts numbers of messages and the other uses network 'bulldozing'. Performance evaluation in actual implementations on a multicomputer with 64-512 nodes, AP1000, exhibits good performance.

Email : {kamada,matsu,yozezawa}@is.s.u-tokyo.ac.jp

1 はじめに

分散環境でのGC (ガーベジコレクション) では、ノード外から参照されたことのあるゴミを少ない通信量で回収するのは困難である。分散GCには、主に次の2種類のアルゴリズムがある。

- リファレンスカウントを用いたもの：
  基本的にノード毎に独立に実行可能なローカルGCを行ない、ノード境界を越えた参照に関してリファレンスカウントを用い管理するもの。但し、ノード間にまたがった循環した参照を持つゴミがとれないもの、とるのが困難なものが多い。[2, 4, 6]
- ノード境界を越えて参照をたどるもの：
  全ノード共同で、ノード境界を越えた参照もたどることで、マークアンドスイープタイプのグローバルなGCを行なうもの。参照の循環したゴミも回収できるが、応用プログラムを完全にブロックするものや、通信量が高いものが多い。通信量に関しては、単にノード境界を越えて参照をたどるためだけでなく、終了判定などにさらにメッセージを必要としている。各ノードで独立実行可能なローカルGCを備えたものもある。[1, 3, 5, 7]

本研究では、非同期メッセージパッシングに基づいた計算モデル上の分散GCのアルゴリズムを提案する。想定する環境は、非同期で信頼できるネットワークで結ばれた、分散メモリをもつ高並列計算機である。

今回の分散GCは、以下のような望ましい性質を持つ。

1. グローバルGCとしてノード境界を越えて参照をたどる種類に属し、グローバルGC時に、すべてのゴミを回収できるものとする。
2. 従来のノード境界を越えて参照をたどる種類のGCでは通信量が問題となっていたが、今回は、グローバルGC時の通信量を抑えるものとする。
3. 応用プログラムの実行 (以下通常計算とする) を長時間中断しないこととする。これはグローバルGCは全ノード共同で行なわれるため、その間全通常計算がブロックされるのは好ましくないためである。このため、グローバルGC時に通常計算とマーキングは並行に行なわれる。
4. ノード毎に独立に行なえるローカルGCを提供し、ノード外から参照されたことのないゴミを効率的に回収できるものとする。

グローバルGCを行なう場合、メッセージ通信によって動的にオブジェクトの参照関係が変化する環境では、非同期ネットワーク上のメッセージ中の参照も考慮し、GC開始以前に送信されたすべてのメッセージの到着を確認する必要がある。その到着確認を低い通信量で行なう方法として、メッセージ数をカウントするメッセージカウント方式と、ブルドージングを用いてネットワーククリアを行なうブルドージング方式[7]を実装し、比較している。

また、グローバルGCの終了判定を低い通信量で行なうのも困難である。今回は重みを用いることで通信量を抑えたアルゴリズムを提案する。

我々の提案したアルゴリズムを、高並列計算機AP1000上で実装および性能評価を行ない、グローバルGCにかかるマークメッセージ以外の通信量の減少を確認した。以下、2節でGCの機構、3節で何をマークするか、4節で終了判定法、5節で性能評価、6節で結論を述べる。

2 機構

2.1 グローバルGCの構成

グローバルGC (以下GGCとも書く) のシステムは、各ノード上のローカルコレクタとグローバルGCホスト (以下単にホストと呼ぶ) から構成される (図1)。各ローカルコレクタはノード内の参照をたどり、マークを行なう。ノード外への参照 (以下リモートな参照とよぶ) があった場合は、参照先のノードにマークメッセージを送ることでリモートな参照をたどる。マークメッセージを受けたノードでは、マークの依頼をされたオブジェクトに対し、それを起点とするマークを行なう。

ホストはGGCの開始の決定・マークの終了判定を行なうために存在し、ノードとの間で終了判定などのための情報を送受信する。あるノードがGGCを行なおうとした場合、GGCの起動をホストに要請、それをもとにホストはGGCの起動を決め、全ノードにGGCの開始を伝える。各ノードはGGCの起動の告知、あるいはマークメッセージの受信があった時点でマークを開始する。一方、マークの終了判定は、ノードからの情報をもとにホストが判定し、その後、各ノードにメモリの解放を指示する。

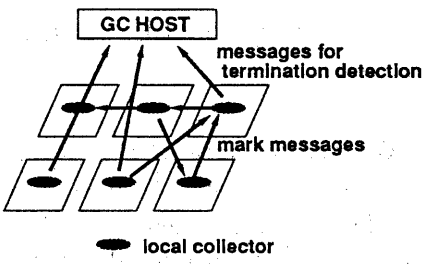

図1: グローバルGCの構成

2.2 ローカルGCへの対応

各ノードはローカルGCを実行するため、外部から参照されうるオブジェクトの集合 (輸出集合とする) を各々持ち、オブジェクトへの参照が最初にノード外に渡される

時、そのアドレスを輸出集合に登録する。ローカル GC を高速に行なうため、輸出集合に属するオブジェクトはグローバル GC 時にのみ GC の対象となる。これは、GGC 開始時に輸出表をクリアし、ノード外から参照されていたもの、つまりマークメッセージによってマークされたもののみを再登録し直すことで実現される。

ローカル GC の際、各ノードは活性オブジェクト、未処理メッセージ、輸出集合をルートとし、これらからノード内への参照によってたどれないものをゴミとする GC を行なう。この際外部から参照されるオブジェクトのアドレスの変更を行なわないよう、ローカル GC ではアドレスを変えない GC をおこなうか、もしくは、輸出集合を輸出表に拡張し、オブジェクトのノード外からの参照はすべて輸出表を介した間接参照にする。今回の実装ではアドレスを固定している。

3 何をマークするのか

分散 GC を実装する上で困難な点としては、(1) 何をマークすれば良いのか、(2) 分散環境でいかに終了判定するか、といったことがある。この節では (1) について議論し、(2) については 4節で議論をする。

3.1 ゴミの定義

まずゴミおよびライブオブジェクトの定義をする。

1. メソッド実行中、もしくは (潜在的に) 未処理のメッセージを持つオブジェクト (以下これらを活性オブジェクトと呼ぶ)、ルートオブジェクト (外部から直接参照されているオブジェクト、大域変数など) はライブオブジェクトである。
2. ライブオブジェクト、およびその未処理メッセージから参照されているオブジェクトは、ライブオブジェクトである。
3. 以上がライブオブジェクトであり、それ以外がゴミである。

今回の GGC は、GGC スタート時点の論理的なスナップショットに対して行なわれ、GGC 開始時点ですでにゴミとなっていたものを回収し、GGC 中にゴミになったものは回収されないものとする。また GGC 中に新たに割り当てられたものは GC の対象としない。

今回の GGC は、ネットワーク中のメッセージも考慮した上で、GGC 開始時点での大域変数、論理的な未処理メッセージ、論理的な活性オブジェクトから参照によってたどることのできるものをマークする GC となる。

3.2 問題点

では実際に何をマークするかであるが、非同期ネットワークにおいてはネットワーク中にメッセージが存在しているため、オブジェクトの参照関係の決定が困難になる。

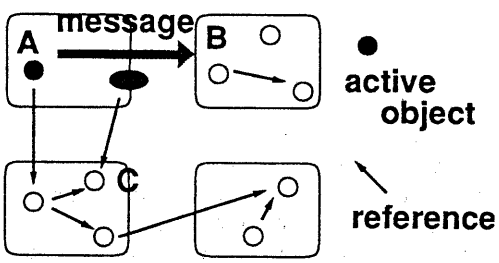

図 2: GC 以前に送信されたメッセージ

図2の例の場合、オブジェクト A からオブジェクト B に対して、オブジェクト C への参照を含んだメッセージが投げられている。この時点で GGC が起きた場合、B、C、およびそこから参照されるオブジェクトは回収されてはならない。なぜなら、この GC 以前に送信されたメッセージ (以後オールドメッセージと呼ぶ) が将来到着した際、B は活性オブジェクトになり、C は B から参照されうるためである。

オールドメッセージは、各オブジェクトの未処理メッセージの一種であり、受信オブジェクトは活性オブジェクトであるとみなせる。つまり、オールドメッセージから参照されるオブジェクトはゴミではなく、そのメッセージから参照されるオブジェクトに対するマークを行なう必要がある。

3.3 対応

これらの議論を踏まえた上で、各ノードはいったい何をマークすれば良いのか考える。

まず、GGC 開始時点での、各ノード内の活性オブジェクト・大域変数・未処理メッセージから参照によってたどることのできるものすべてをマークする。

また、3.2節で見たように、GGC 開始時点で未到着であったオールドメッセージから参照されるオブジェクト (受信先オブジェクトも含む) はライブオブジェクトであるため、これに対するマークを行なう必要がある。

オールドメッセージに対するマークに対する責任を、送信側で持つようにすると、通常計算時に負荷がかかるため、オールドメッセージを受信した時点で、受信側がマークを行なうこととする。つまり、各ノードは GGC 中受信メッセージを監視しておき、オールドメッセージ (もしくはオールドメッセージである可能性を持つメッセージ) を受信した場合、これに対するマークを行なう。何をもってオールドメッセージと判断するかについては、終了判定法にも関わるため、4.1節に述べる。

また、オールドメッセージがマーク終了後に到着するようなことがあると、オールドメッセージから参照される

オブジェクトは誤って回収されかねないため、すべてのオールドメッセージの到着確認が行なわれた後に、マークの終了判定がなされなければならない。これに関しては4節で議論する。

最後に、通常計算とマーキングは並行に行なわれるため、通常計算によって、マークが済んでいない箇所への書き込みが行なわれ、GGC 開始時の参照関係が壊れる可能性がある。今回、書き込みチェックを用い、書き込みによって失われる参照情報に対して、これを起点とするマークを行なう。これにより、GGC 開始時点でのすべてのライブオブジェクトに対して、マークを保証することができる。

まとめると、GGC 中、ローカルコレクタは以下のマークを行なう。

1. GGC 開始時点での、各ノード内の活性オブジェクト・大域変数・未処理メッセージに対するマーク
2. マークメッセージに対するマーク
3. GGC 開始後に到着した、オールドメッセージに対するマーク
4. マークされていない領域に対する、通常計算による書き込みの際、失われる参照情報からのマーク
5. 1~4 でマークされたものから、参照によってたどることのできるものへのマーク

また、ノード外への参照に対するマークを行なう場合、参照先オブジェクトの所属ノードのローカルコレクタに、マークメッセージを送信する。

4 終了判定法

分散環境で、マークの終了判定をいかに低い通信量で行なうかという事も、分散 GC で重要な問題の一つである。

3節での議論を踏まえた上で、次の条件が成立した際にマークが終了したと考えられる。(以後、マーク終了の3条件と呼ぶ)

1. オールドメッセージの到着確認が行なわれ、オールドメッセージに対するマークが終了している
2. すべてのローカルコレクタが非活性状態(ノード内にマークするものがない状態)である
3. マークメッセージがネットワーク中にない

条件2・3を満たした状態をマーキングの静寂状態と呼ぶ。

まず、条件1より、すべてのオールドメッセージに対するマークの完了がいえる。また、条件2・3より、すべてのマーク活動が行なわれていないことがいえる。この時点で、すべてのライブオブジェクトはマークされたこととなり、これから後マークされていない領域に対する書き込みも起こらず、新たなマークも行なわれない。

まず最初に、4.1節で、条件1の成立、つまりすべてのオールドメッセージの到着を確認する方法について述べる。その上で、4.2節で、条件2・3の成立判定法、つまり静寂状態の検出法について述べる。

4.1 メッセージの到着確認

非同期ネットワークでは、GGC 開始時点で未到着であった、オールドメッセージから参照されるオブジェクト(受信先オブジェクトも含む)に対するマークを行なわなくてはならず、ネットワーク中のメッセージの到着確認が必要である。

今回は到着確認を行なう方法として、ACK(Acknowledge 応答)を用いた方式を新たに改良したメッセージカウント方式と、ネットワーククリアの一種であるブルドージング方式を、それぞれ実装し評価している。

4.1.1 メッセージカウント方式

従来から用いられている、ACK を用いた方式とは、通常計算メッセージの到着を知らせる ACK メッセージを送り返す方法である。但し、この方法を単純に実装すると、通常計算時のメッセージ送信量が2倍になり、またメッセージ内に送信側のノード ID を含む必要がある。

我々の考案したメッセージカウント方式は、ACK 方式を改良したもので、各ノードで、あらかじめオールドメッセージの送信数と着信数をカウントしておき、システム全体での送信数と着信数が等しいかみることで、到着確認を行なうものである。

そのために、各ノードは、GGC 毎に代わる共通した色を持ち、通常計算メッセージの送信の際に、その時点での色をつけて送る。メッセージの送受信に際しては、メッセージの色毎にカウントを行なう。

どのようにしてシステム全体での総送受信数を回収、到着確認を行なうかであるが、各ノードは GGC 開始時にオールドメッセージの送信数をホストに送る。すべてのノードから送信数がホストに集められていれば、総送信数がホストに集められたことになる。また、各ノードはノード内にマークするものがなくなった時点で、オールドメッセージの受信数をホストに送信する。その後再びオールドメッセージが到着した場合、オールドメッセージに対するマークが行なわれ、再びノード内にマークするものがなくなった時点で、受信数をホストに送信する。ホストは総送信数が受理されており、それが受信数の和に一致していれば、すべてのオールドメッセージの到着確認が行なわれることとなる。

また、マークが完了した、つまりマーク終了の3条件が成立した時点で、ホストにすべてのノードからの総着信数が送られることになり、有限時間内に確かに到着確認がなされるといえる。

色については、同時に1回の GGC しか行なわれず、

受信の際に混ざり合う色は高々2色しかないので、実際には1ビット分の情報があれば十分である。

また、メッセージカウント方式では、GGC 開始以前の色をつけたメッセージを、オールドメッセージであるとみなす。GGC 中のオールドメッセージに対するマークでは、各ノードは到着メッセージをチェックし、GGC 開始以前の色をつけたメッセージを、含まれる参照を起点とするマークを行なえばよい。

到着確認に ACK を用いた場合と比較をすると、ACK を用いた方式では、通常計算時のメッセージに色の代わりに、送信ノード ID を付ける必要があり、ACK メッセージの送受信を、GGC 中にまとめて行なったとしても、すべてのノード対間で ACK 情報が送受信されることとなる。一方、メッセージカウント方式では、到着確認のためのメッセージを GGC 中にまとめて行ない、各ノード間での到着確認をネットワーク全体でまとめて行なっているので、通信量が抑えられている。

4.1.2 ブルドージング方式

ブルドージング方式は、FIFO 性のあるネットワークからメッセージを吐き出す、ブルドージングというネットワーククリアの1手法を用いている。

ネットワークの FIFO 性を用い、新たなメッセージの送信を禁止することなく、すべてのパスにブルドージングメッセージを流し、その到着をもってネットワーク上のオールドメッセージの到着確認を行なう [7](図3)。この場合、ブルドージングメッセージ到着以前に、マーク開始後に送信されたメッセージが到着することもある。ブルドージングは単にオールドメッセージの吐き出しのみを行なっている (図3)。

ところが、ブルドージングを用いた実際的な実装はなかった。[7] で提案された方法では、ブルドージングメッセージが、$O(N^{1/2})$ 回 (N: ノード数)、ノードを経由して送られ、遅延が問題となる。また、すべてのノード対間でメッセージを流した場合、通信量が $O(N^2)$ 問題となる。

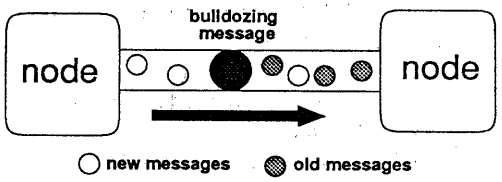

図 3: ブルドージング方式

今回の我々の AP1000 上の実装では、ブルドージングメッセージ数とブルドージング時間の抑えるように、すべてのノード対間でメッセージを流すのではなく、物理的なネットワークトポロジを考えながらブルドージング開始以前のメッセージの吐き出しを行なう。ブルドージングに用いられたメッセージ数はノード数の8倍程度で、また、メッセージは最大6回、ノードを経由しながら送り、メッセージの送受信に関する遅延を抑えるようにした。今回のブルドージングは単にオールドメッセージの吐き出しのみを行なっている。

また、ブロドージングメッセージをマーク中に並行して流すことで、ブルドージングはマークと並行に行なわれる。

ブルドージング方式では、ブルドージングの終了通知以前に到着した通常計算のメッセージのみが、オールドメッセージである可能性がある。今回、メッセージの色づけは行なわず、GGC 中のオールドメッセージに対するマークでは、GGC 開始からブルドージング終了通知を受けるまでの、すべてのメッセージに対して、含まれる参照を起点とするマークを行なう。この場合、余分なマークを行なうこととなるが、色情報が不必要になり、通常計算時の通信量が減り、色情報を扱うためのオーバーヘッドはなくなる。

4.2 静寂状態の検出

分散環境でマークを行なう際のもう一つの課題は、マーク活動の終了を低い通信量で判断することである。

良く知られた終了判定に、自分を起点とするマークの終了を親に知らせていく方法がある。すべてのノードのルートセットの要素が、マークの終了メッセージを受けとった時点で、全体のマークが終了したと判断できる。この場合、マークメッセージと同数の終了メッセージを送信しなくてはならない。また、[1, 5] では GC の色に3色を持ちいた終了判定を行なっているが、この場合もマークメッセージの到着確認に ACK メッセージを送信している。

今回、マークの終了条件の2・3、つまりマーキングの静寂状態を検出する方法として、重みを用いた方法を提案する。マークメッセージやマーク中のノードに対して、ホストが正の重みを与え、すべての重みがホストに返却された時点で全体のマークの終了とする方法である。

ホストは、GGC 開始時点で各ノードに対して重みを与える。各ノードはマークメッセージを送信する際、正の重みを付加し、その分ノードの持つ重みを減ずる。受信側ではマークメッセージの重みを、自分の重みに加える (図4)。つまり、各ノードとネットワーク中のマークメッセージの重みの総和がホストの与えた重みに等しく保たれる。この際、マーク中のノード内の重みが0にならないようにし、必要ならホストに重みを要請しする。その間、そのノードからのマークメッセージの送信はブロックされる。

各ローカルコレクタは一定時間待ってもマークするものがないと、重みをホストに返却する (図5)。

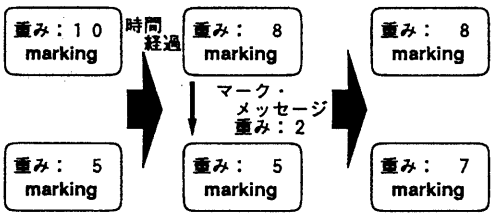

図 4: 重みの移動 I

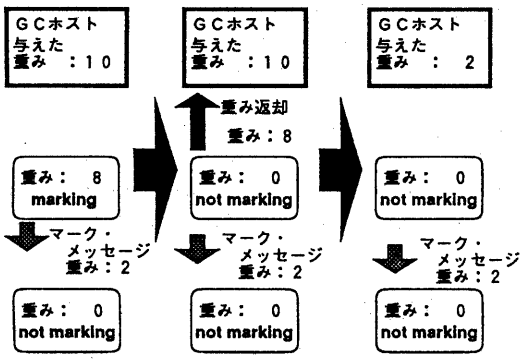

図 5: 重みの移動 II

ホストでは、すべてのオールドメッセージの到着確認が終了し (終了条件 1 成立)、重みもすべて返却された (終了条件 2・3 成立) 時点で、マークの終了を決定する。

すべてのオールドメッセージの到着確認がなされた後は、マークを行なっているノードは、基本的に正の重みを持ち、マークメッセージも正の重みを持つので、すべてのマークが終了するまで重みがすべて返却されることはない (図 5)。例外的に、マークされていない領域に対する書き込みのため、あるノードが重み 0 でマークを開始することがある。しかし、この場合、そのノードはいずれ外部からマークされるオブジェクトを持っているので、システム全体での重みは正であり、誤った終了判定がなされることはない。

また、マークが終了した、つまりマークの終了条件 1~3 が満たされた時、有限時間以内にすべての重みはホストに返されるため、確実に静寂状態の検出が行なわれる。

マークの終了メッセージを返す方法と、今回の重みを用いた方法の、通信量を比較する。前者では、マークメッセージ以外に同数のマーク終了メッセージが必要となるが、後者の場合は、マークメッセージ以外にかかるのはノード単位で行なう重みの返却・要請だけであり、通信量は少なく、また終了メッセージの返却先を覚えておくための記憶領域もいらない。逆に後者は重みの返却・要請のメッセージがホストへ集中しがちであるが、これらのメッセージはノード単位で扱うため数も比較的少ないと期待される。

4.3 アルゴリズム

実際に各ノードのローカルコレクタが何を行なうか、オールドメッセージの到着確認法に、各 2 方式を用いた場合をそれぞれ説明する。

1) メッセージカウント方式を用いた場合

通常計算フェイズ：

- メッセージの送受信数を色別にカウントする。GGC 開始の通知またはマークメッセージの到着で GGC フェイズに移る

GGC フェイズ：

- 初期化：
  色を新たにし、ノード内の活性オブジェクト・大域変数・未処理メッセージをマークのルートセットとする。オールドメッセージの受信数がホストに送る。
- 通常メッセージ：
  メッセージの送受信数を色別にカウント。
  GGC 開始以前の色のメッセージに対しマークを行なう
- マークメッセージ：
  参照先オブジェクトからのマークを行なう。
- マークされていない領域からのマーク：
  書き込みチェックを行ない、失われる参照情報からのマークを行なう
- 一定時間たってもマークするものがない場合、メッセージの受信数と、重みをホストに返却する
- マーク終了の通知でメモリ領域を解放、通常計算フェイズに戻る

2) ブルドージング方式を用いた場合

通常計算中：

- GGC 開始の通知またはマークメッセージの到着で GGC フェイズに移る

GGC フェイズ：

- 初期化：
  ノード内の活性オブジェクト・大域変数・未処理メッセージをマークのルートセットとする。
- 通常メッセージ：
  ブルドージング終了通知以前は到着メッセージに対するマークを行なう。
  ブルドージング終了通知以後は何もしない。
- マークメッセージ：同上
- マークされていない領域からのマーク：同上
- ブルドージング終了通知：
  もし、ノードがマーク中でないなら、重みをホストに返却。
- 一定時間たってもマークするものがない場合、ブルドージング終了通知以後なら重みをホストに返却。

- マーク終了の通知でメモリ領域を解放、通常計算フェイズに戻る。

両方式とも、GGC 開始時にノード間の同期は行なわれず、通常計算がブロックされるのは GGC 開始時・オールドメッセージの到着時・マークされていない領域への書き込みの際など、いずれも短時間である。また、GGC 中も通常計算はマークと並行に行なわれる。

今回のアルゴリズムでは、GGC の際にマークメッセージ以外に、終了判定のために、ホストと各ローカルコレクタの間でメッセージの送受信がなされる。メッセージカウント方式では、到着確認のためのメッセージ受信数情報と、重みの返却は同時に行なわれている。

また、ホストへ終了判定のための情報の送信は、一定時間たってもマークするものがない場合に行なわれている。この間、別の通常計算が行なわれることになる。これは、一度マークが中断し、その後マークの再開が行なわれた場合、ローカルコレクタからホストへの終了判定のためのメッセージが送信されるが、その頻度を抑え、ホストへのメッセージの集中を抑えるためである。この時間を長くすると、ホストのメッセージの集中は抑えられるが、マークの終了判定までに要する時間が長くなると思われる。さらに、GGC に要するおおよその時間がわかっていれば、マーク開始から一定期間、重みの返却をブロックすることで、ホストへのメッセージ数は減少できると考えられる。今回、これらの 2 方法を用いて、ホストへのメッセージの集中の減少を計っている。

5 性能評価

これらの分散 GC を評価するため、64(8×8) ノードの AP1000(各ノードは 25MHz SPARC プロセッサと 16MB メモリからなる) 上で N-Queens のプログラムを実行し、GC の様子を調べた結果、表 1、図 6、図 7 を得た。

	message count	bulldozing
# of gc messages except for mark messages : A	410	640
# of mark messages :B	13700	15700
A / B	3.0 %	4.1 %
marking time(msec)	191	191
# of gc messages to host	410	210

表 1: グローバル GC の性能評価 (12-Queens)
重みの返却のブロックには、計時を行なう代わりに、何回メソッドが実行されたかで判断している

GGC を実現するためのマーク以外のメッセージ数を考える。一回の GGC で行なわれる重みの返却・要請、プ

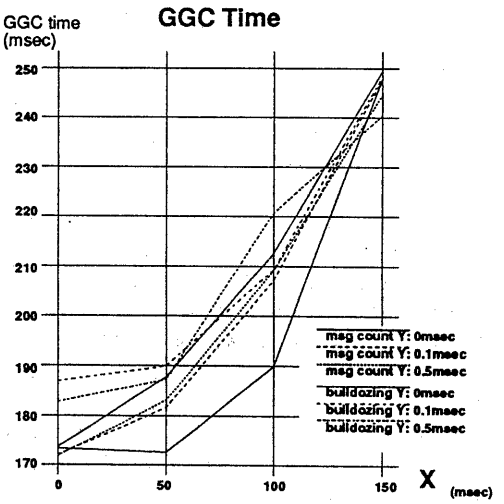

図 6: 重み返却のブロックとマーク時間
X: マーク開始からどれだけの期間、重み返却をブロックしたか
Y: マーク活動中断からどれだけの期間、重み返却をブロックしたか
図 6,7の評価では実際に計時を行なっているため、表 1の場合に比べ GGC は全体に遅くなっている

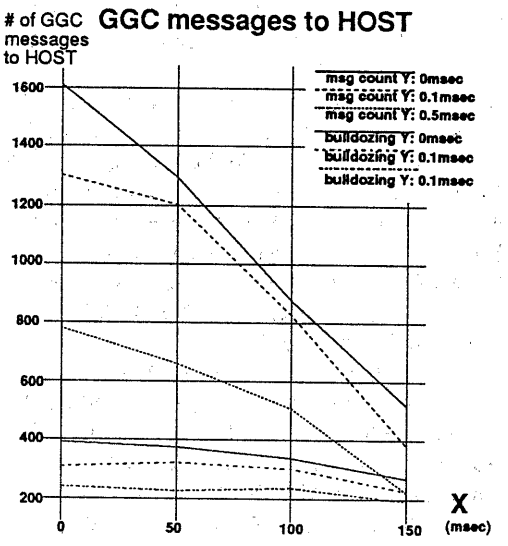

図 7: 重み返却のブロックと GGC メッセージのホストへの集中

ルドージングなどマーク以外に用いられるメッセージ数(A)は、その際に行なわれるマークメッセージ数(B)に対し3~10％程度であった。つまり、GGCに要したメッセージ数は、リモートな参照数の1.1倍以内で収まっている。GGCの終了判定にマークメッセージ数と同数の終了メッセージを用いた場合、GGCに要するメッセージ数は、リモートな参照数の2倍を越えるので、我々の方法では通信量は抑えられていることがわかる。

両方式とも数100msecでGGCのマークが終了していた。但し通常計算が並行に行なわれており、単にGGCのみが行なわれていたわけでない。この間、メモリの解放も行なわず、また、ローカルGCがブロックされるが、その時間としては許容される時間であると思われる。

メッセージカウント方式とブルドージング方式を比較すると通信量はブルドージングの方が多い。これは、ブルドージング方式では、オールドメッセージの到着確認のため、ブルドージングメッセージを送受信する必要があるのに対し、メッセージカウント方式ではメッセージの到着確認が重みの返却と同時に行なえるため、通信量が抑えられたためである。また高速なブルドージングを考えるとハードウエアに依存したものになりそうである。逆にブルドージングは通常計算で色に伴うオーバーヘッドがないなど高速化への利点も持つ。

ホストへのメッセージの集中は少なく、今回は問題とならなかった。今回の場合、ノード数の3~20倍程度のメッセージがホストに集中しているが、重みの返却のブロックを行なうった場合には、ノード数の3~5倍程度で収まっている。但し、ノード数が増えるとメッセージの集中は避けられないと考えられるため階層化を用いるなどの対処が必要である。

重みの返却を一定時間ブロックした場合、どの程度ホストへのメッセージの集中を抑えることができたかを詳しく見ていく。まず、マークが開始してからしばらくの間重みの返却をブロックした場合、メッセージカウント方式では効果が得られているが、ブルドージング方式では効果がほとんどない。これは、もともとブルドージング終了まで重みの返却が行なわれていないためである。また、ブロックしている時間がある程度を越えると、GGCにかかる時間にそのまま影響している。一方、マーク活動が一旦中断した時、しばらく重みの返却をブロックし、マーク活動が再開されない場合に重みを返却する場合、比較的短い時間のブロックであっても、メッセージ数は減少した。また、終了判定がなされるまでの時間も、それほど影響なく、効果的であることがわかった。

今回の実装では、GGC関連のメッセージは通常メッセージと同様に処理されている。このため、GGCの作業が優先的に行なえず、メッセージが複数のノードを経由して送られた場合、遅延が大きくなっている。もしGGC関連のメッセージを優先的に扱うことができれば、GGCにかかる時間は少なくなり、またアルゴリズムの選択の幅が増えると思われる。

6 結論

今回のアルゴリズムを用いることで、グローバルGC時のメッセージの数はともに少なくできた。終了判定などのマークメッセージ以外のGCのためのメッセージの数は、マークメッセージ数の3~10％程度に抑えることができた。これは、従来の方式に比べ、グローバルGCに必要なメッセージ数が約半分になったことを意味する。

メッセージカウント方式とブルドージング方式を比較すると、後者は色を用いなくて良いなど利点を持つが、ブルドージングの手順がハードウエア依存になりがちで、またブルドージングメッセージの中継の際の遅延が問題であった。

今回は、グローバルGC用メッセージも通常メッセージと同様に処理されたのであるが、優先度つきの低コストなメッセージ読み込み機構があると高速化がはかれそうである。

今後の課題としては、実際にコンパイラに組み込めるよう実用的なものにすること、ノード数が増えた場合にホストへのメッセージの集中を抑えるための階層的な枠組、負荷分散・オブジェクトの移動などへの対応を考えている。

謝辞

アルゴリズムの考案、AP1000上での実装など、多くの面で助言をして頂いた米澤研究室の八杉、田浦両氏に深く感謝致します。

参考文献

[1] Lex Augusteijn. Garbage collection in a distributed environment. Vol. 259 of *LNCS*. PARLE, 1987.

[2] D I Bevan. Distributed garbage collection using reference counting. Vol. 259 of *LNCS*. PARLE, 1987.

[3] Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-grained mobility in the emerald system. ACM Transactions on Computer Systems, 2 1988.

[4] Bernard Lang, Christian Queinnec, and Jose Piquer. Garbage collecting the world. POPL, 1992.

[5] Niels Christian Luul and Eric Lul. Comprehensive and robust garbage collection in a distributed system. In *Memory Management*, Vol. 637 of *LNCS*, 1992.

[6] Marcel Schelvis. Incremental distribution of timestamp packets: A new approach to distributed garbage collection. OOPSLA, October 1989.

[7] Nalini Venkatasubramanian, Gul Agha, and Carolyn Talcott. Scalable distributed garbage collection for system of active objects. In *Memory Management*, Vol. 637 of *LNCS*, 1992.