A-014

Caterpillar GC: 旧世代領域の分割を行うインクリメンタルな世代別実時間ごみ集め

Caterpillar GC: Incremental Generational Real-time Garbage Collection with Partitioned Old Generation

尾沢 崇 † Takashi Ozawa　**矢崎 俊志** † Syunji Yazaki　**阿部 公輝** † Kôki Abe

1. まえがき

モバイル環境でのゲームなど実時間性を重視する場合，ガベージコレクション (GC)[1] の実行によるプログラムの停止時間が問題となる．

インクリメンタル GC は，GC 処理によるユーザプログラムの停止時間を短縮するために，GC の処理を分割する．しかし，分割により，リードバリアやライトバリア [2] のような，ユーザプログラムによるオブジェクトの変更を検知する機構が必要になり，オーバヘッドが発生する．

世代別 GC[3] は，オブジェクトの生存，すなわち，ユーザプログラム中で使われているかの判別 (走査) の対象を短寿命のオブジェクト (新世代) に限定することで停止時間を短くする．Treadmill GC[4] は，4 つの双方向リストを利用してオブジェクトを管理する．リストの繋ぎ替えにより，物理的にオブジェクトを移動させずにメモリを管理する．この手法では，ライトバリアに比べ処理に時間のかかるリードバリアが不要である．

世代別 GC に Treadmill GC の考えを取り入れた Opportunistic Treadmill GC[5] がある．この手法はメモリ領域を新世代領域と旧世代領域に分割し，走査の必要がないオブジェクトを走査対象から外すことで，走査すべきオブジェクトの数を減らす．また，リードバリアを必要としない．しかし，メモリ領域が大きく不足した時は，長寿命のオブジェクト (旧世代) の走査も必要となる．

本研究では，Opportunistic Treadmill GC を拡張し，メモリ領域が不足した時，旧世代を分割し，一定数のオブジェクトを新世代に加え，インクリメンタルに走査することで最大停止時間を減らす．本手法を Caterpillar GC と呼ぶ．本稿では Caterpillar GC を提案し，実装と動作実験の結果を述べ，考察する．

2. 関連研究

ここでは，提案手法の基となる，Treadmill GC と Opportunistic Treadmill GC について説明する

2.1.Treadmill GC

双方向環状リンクで全てのオブジェクトをつなぎ，リンクの付け替えでオブジェクトの移動を表現する．Copying GC[6] と似た動作をするが，オブジェクトの物理的な移動を行わない．

このアルゴリズムでは，図 1 に示すように，ヒープ領域を次の 4 つのセグメントに分割する．

- 解放済みまたは未使用のオブジェクトのリスト Free

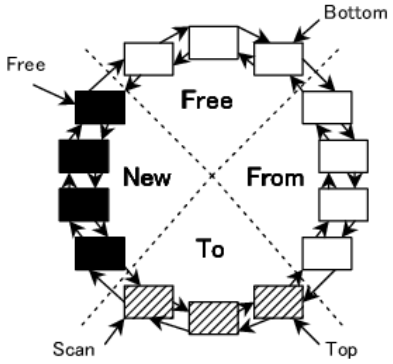

図 1: Treadmill GC

- 走査対象のオブジェクトのリスト From
- 生きているオブジェクトが移動する先のリスト To
- プログラムに新しく割り当てられたオブジェクトのリスト New

これらのセグメントの境界は，図 1 のように，4 つのポインタ，Bottom，Top，Scan，Free で表す．

オブジェクトの割り当ては Free リストから From リストに加えることでなされる．Free リストのオブジェクトが少なくなると GC が起動される．手順は次の通り．

1. From リストにおいて，生きているオブジェクトを探索し，To リストに加える．
2. To リストにおいて，Scan が指すオブジェクトをルートとして，From リストにある生きているオブジェクトを探索し，To リストに加える．
3. Scan が指す To リストのオブジェクトを New リストに加え，GC を中断する．
4. GC 中断中にオブジェクトの割り当て要求があると，Free リストからオブジェクトを New リストに加える．
5. To リストが空でなければ，1. に戻る
6. To リストが空になれば，From リストのオブジェクトはすべてごみなので (生きていないので)，From リストのオブジェクトをすべて Free リストに加え，To リストと New リストを合わせて新たに From リストとして，GC を終了する．

†電気通信大学， The University of Electro-Communications

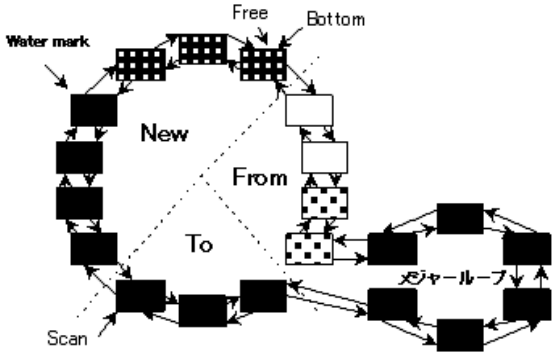

図 2: Opportunistic Treadmill GC

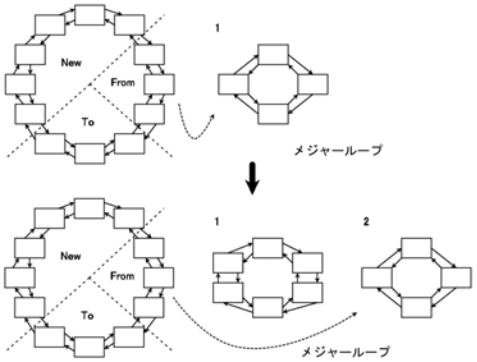

図 3: 提案手法におけるメジャーループの作成

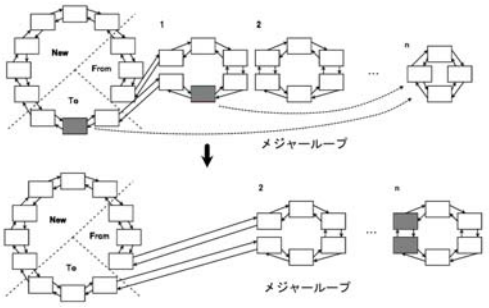

図 4: 提案手法のメジャー GC

2.2.Opportunistic Treadmill GC

世代別 GC は，新しく確保されたオブジェクトは，長く生存しているオブジェクトに比べてごみになりやすいという世代別仮説に基づく [3]．短寿命のオブジェクトは，新世代領域，長寿命のオブジェクトは，旧世代領域とし，GC は新世代領域のみで行う．

Opportunistic Treadmill GC は，世代別 GC の各世代を，Treadmill で構成する [5]．新世代領域の Treadmill をマイナーループ，旧世代領域の Treadmill をメジャーループと呼ぶ．メモリの割り当てやごみの回収は通常はマイナーループのみで行う．これをマイナー GC と呼ぶ．マイナーループで一定の回数生き残ったオブジェクトは，メジャーループへ繋ぎ替えられる．これを殿堂入りという．マイナー GC で十分な空き領域が確保できなかった場合，図 2 のように，メジャーループのオブジェクトを全てマイナーループに繋ぎ替えることで，全てのオブジェクトに対して GC を行う．これをメジャー GC と呼ぶ．

マイナー GC では走査すべきオブジェクトの数が少なくなるため，停止時間が短くなる．しかし，メジャー GC では全てのオブジェクトに対して走査を行うため，GC に Treadmill GC と同程度の時間がかかり，停止時間が増大する．

3. 提案手法

Opportunistic Treadmill GC では，マイナーループのみを走査する場合，停止時間が短時間になることが保証されている．しかし，メジャーループの挿入を行った場合，その停止時間は世代を持たない通常の Treadmill GC と同じとなり，短時間で終わることが保証されない．この問題を解決するために，提案手法ではメジャーループのオブジェクトの個数に上限を定め，上限を超えると，新しい別のメジャーループを作成する．この様子を図 3 に示す．図では，作成された 2 つのメジャーループ 1 および 2 が示されている．このように，提案手法では，ヒープ領域は，1 つのマイナーループと，一定数以下のオブジェクト数で作られる複数のメジャーループからなる．

マイナー GC で十分な空き領域が確保できなかった場合，隣接するメジャーループをマイナーループに挿入することでメジャー GC を行う．この方法で一度に挿入される走査対象のオブジェクトの量を減らし，停止時間が短時間で終了することをはかる．

この手法では，メジャーループとなる Treadmill にそれぞれ挿入する順番となる番号を割り振る．マイナー GC では従来手法通りマイナーループに対してのみ GC を行う．メジャー GC が必要になった場合，マイナーループに隣接するメジャーループ i をマイナーループに挿入し，GC の対象とする．図 4 の上部では，マイナーループにメジャーループ 1 を挿入してメジャー GC を行い，その結果生き残ったオブジェクトを終端のメジャーループ n へ移動する様子を示している．この後，GC を終了する．さらにこの後，新たにメジャー GC を開始する場合，図 4 の下部のように，マイナーループに隣接するメジャーループ 2 を挿入して GC を行う．

複数のメジャーループを作成することで，1 回のメジャー GC で回収できるオブジェクトの個数が小さくなるため，メジャー GC を開始する時期が早くなることが考えられる．また，1 回のメジャー GC によって十分な量のオブジェクトが回収できず，メモリの飢餓状態に陥る危険性がある．この問題はメジャーループの大きさの上限値を適切に設定することで，回避できると考えられる．

4. 実装と動作実験

Catterpillar GC を実装し，Opportunistic Treadmill GC の動作と比較する実験を行った．

はじめに Treadmill の構造を持った通常の GC を実装し，インクリメンタル化した．次に世代別 GC の各世代

```
void test_02(int num) {
    Object iv_0 = new_ivalue(memory,  0);
    Object pair = push_pair(memory,  iv_0,  iv_0);
    Object *cdr = &PAIR_CDR(pair);
    int i;
    for (i=1;i<num;++i) {
        Object iv_i = new_ivalue(memory,  i);
        (*cdr) = new_pair(memory,  iv_i,  iv_i);
        cdr = &PAIR_CDR(*cdr);
    }
    printf("%p:",  pair);
    print_object(pair);
    printf("\n");
    fixed_memory_pop(memory);
}
```

図 5: テストプログラム

を Treadmill で構成し，殿堂入りとメジャー GC を実装することで，Opportunistic Treadmill GC を作成した．さらに，Opportunistic Treadmill GC のメジャーループを複数化することで，Caterpillar GC を実装した．

GC の方式は Exact GC を用いた．Exact GC はオブジェクトに型の情報を持たせることでポインタの区別を行う方法である．また，ライトバリアはオブジェクトにライトバリア用のポインタを持たせ，ライトバリアに含まれるオブジェクトが線形リストとなるよう実装した．

実装は，C 言語により Microsoft 社の Visual Studio 2008 Version 9.0.30729.1 SP 及び.NET Framework Version 3.5 SP1 の環境で行った．

4.1. 動作実験

実験では，ヒープ領域のオブジェクトの最大数を 32 とし，Caterpillar GC では，1 つのメジャーループに含まれるオブジェクトの上限数は 4 とした．

テストプログラムは図 5 に示すように，car 部と cdr 部をポインタとして持つオブジェクトで線形リストを指定した数 num 個だけ生成する．car 部は数値のオブジェクトを指し，cdr 部は次のポインタのオブジェクトを指す．メモリを確保した時，ヒープ領域の空のオブジェクトが，最大数の半分以下であった場合に GC が開始される．

4.2. 実験結果

テストプログラムを実行すると，ヒープ領域のオブジェクトが 16 個まで確保されたところでマイナー GC が実行される．マイナー GC が実行されても，空のオブジェクトが増えないため，メジャー GC が開始される．メジャー GC 開始直前のヒープ領域は，Opportunistic Treadmill GC では図 6，Caterpillar GC では図 7 のようになる．Caterpillar GC では，メジャーループのオブジェクトの上限数が 4 に設定されているため，大きさ 4 のメジャーループが 4 個，大きさ 1 のメジャーループが 1 個できている．図にはマイナーループに近接するものから順番に番号が割り振られている．

テストプログラムでは，マイナー GC 後もマイナーループの空のオブジェクト数が増えないため，メジャー GC が開始され，メジャーループが挿入される．このとき，Opportunistic Treadmill GC では図 8，Caterpillar GC では図 9 のようになる．

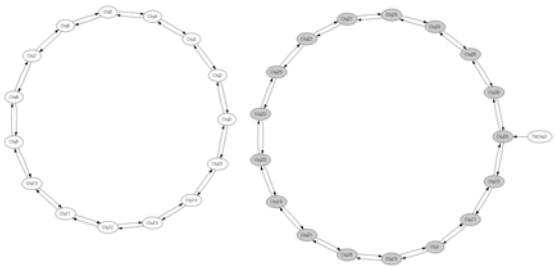

図 6: Opportunistic Treadmill GC におけるメジャー GC 開始直前のヒープ領域．白ノードはマイナーループのオブジェクト，灰ノードはメジャーループのオブジェクトを表す．

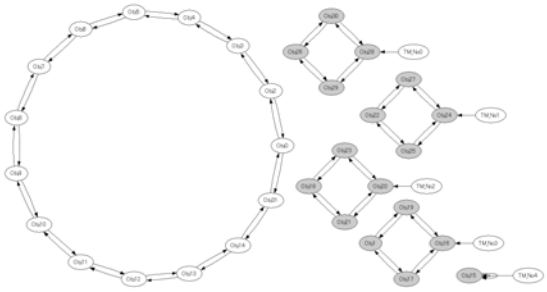

図 7: Caterpillar GC におけるメジャー GC 開始直前のヒープ領域．白ノードはマイナーループのオブジェクト，灰ノードはメジャーループのオブジェクトを表す．

これらの図において，黒ノードのオブジェクトがメジャー GC を行うために新たに挿入されたオブジェクトであり，次の GC で走査対象となる．図 8 の Opportunistic Treadmill GC では，大きさ 16 のメジャーループが挿入される．図 9 の Caterpillar GC では，マイナーループに近接するもっとも番号の小さいメジャーループが 1 つ挿入され，マイナーループのオブジェクトの増加数は 4 となっている．このように，提案手法ではメジャー GC において走査すべきオブジェクト数が減少することを確認した，この結果，メジャー GC による停止時間が短縮されると期待される．

5. 考察

提案手法の実時間性を考察するために，メジャー GC の停止時間を見積もる．

ヒープ領域の全てのオブジェクト数を N とする．マイナーループに含まれるオブジェクト数を N' とする．メジャーループの数を n，マイナーループとメジャーループのオブジェクトの生存率をそれぞれ k_{young}，k_{old} とした場合，メジャー GC の停止時間 t は次のように見積もられる．

$$t \propto N' \times k_{young} + \frac{(N - N') \times k_{old}}{n} \quad (1)$$

$$= (k_{young} - \frac{k_{old}}{n})N' + \frac{k_{old}}{n}N \quad (2)$$

図 8: Opportunistic Treadmill GC のメジャー GC．白ノードはマイナーループのオブジェクト，黒ノードはメジャー GC によってマイナーループに追加されたオブジェクトを表す．

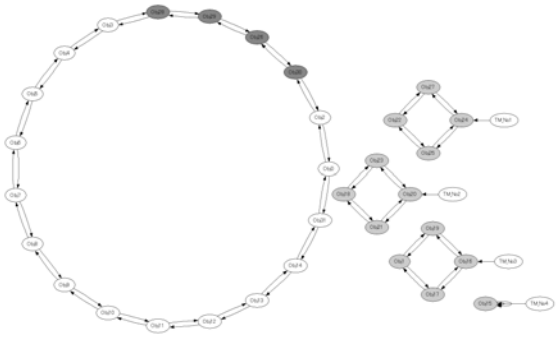

図 9: Caterpillar GC のメジャー GC．白ノードはマイナーループのオブジェクト，灰ノードはメジャーループのオブジェクト，黒ノードはメジャー GC によってマイナーループに追加されたオブジェクトを表す．

全てのオブジェクトとマイナーループに含まれるオブジェクトの比 $N/N' = 1, 2, 4, 8, 16$ において，メジャーループの数 n が増加した場合の式 2 の値 t の変化を調べた．新しいオブジェクトがごみになりやすいという世代別仮説に従って，それぞれのオブジェクトの生存率を $k_{young} = 0.1$, $k_{old} = 0.9$ としてグラフを作成し，図 10 に示す．

図 10 から，メジャーループの数が増えるほど停止時間が減少することがわかる．また，N/N' の値が大きい場合，メジャーループの数が増えるほど停止時間が減少することがわかる．例えば，$N = 32$ とする．$n = 4$, $N' = 16$ の場合，図 10 から提案手法では $t \simeq 1.4$ である．従来手法は $n = 1$ に相当するので，$t \simeq 2.8$ となり，停止時間は約 1/2 となる．マイナーループのオブジェクト数が少ない状態，例えば $N' = 2$ であれば，$n = 4$ のとき，停止時間は約 1/3 となる．$n = 16$ であれば，停止時間は約 1/7 となる．

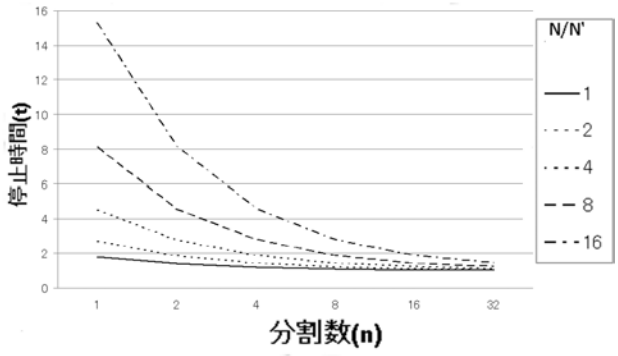

図 10: メジャーループの数による停止時間の変化の見積もり

6. おわりに

本稿では，メジャーループのオブジェクト数に上限を設け，複数のメジャーループを作成する Caterpillar GC を提案した．動作実験では，メジャー GC において Opportunistic Treadmill GC に比べ走査すべきオブジェクト数が少なくなることを示した．また，考察では Caterpillar GC の最大停止時間に関する見積もりを行った．Opportunistic Treadmill GC との比較実験については今後の課題である．

参考文献

[1] R. Jones, R. Lins: *Garbage Collection, Algorithms for Automatic Dynamic Memory Management*, Wiley & Sones, 1996.

[2] A. L. Hosking, J. En B. Moss, D. Stefanovic: A Comparative Performance Evaluation of Write Barrier Implementations. *ACM SIGPLAN Notices*, Vol. 27, Issue 10, 1992.

[3] M. Hirzel: Data layouts for object-oriented programs. *SIGMETRICS Perform. Eval. Rev.*, Vol. 35, No.1, pp. 265-276, 2007.

[4] H. G. Baker: The Treadmill: Real-Time Garbage Collection Without Motion Seckness, *ACM SIGPLAN Notices*, Vol.27, No.3, pp. 66-70, 1992.

[5] 小池龍信，岩井輝男，中西正和: オブジェクトの世代を考慮に入れたインクリメンタルなごみ集め処理，情報処理学会論文誌，Vol.40, No.SIG7, PRO 4, pp. 1-8, 1999.

[6] C.J. Cheney: A nonrecursive list compacting algorithm, *Comm. ACM*, Vol.13, No.11, pp.677-678, 1970.