論文の内容の要旨

論文題目　世代方式ガーベジコレクタの効率化に関する研究

氏名　　吉川隆英

計算機のメモリ資源は有限である．そのため，計算機上で実行されるユーザプログラムは，動的に不要となったメモリ領域を解放し，再利用しなくては，メモリ不足によって途中で停止してしまうおそれがある．この再利用処理はプログラマが手動で行う場合と，言語処理系が自動的に行ってくれる場合とがある．C, C++などの言語処理系の多くは前者であり，Scheme, Java, KLICなどの処理系は後者である．前者では，プログラム中でメモリ領域を確保したら，使用後に必ず解放せねばならないため，プログラマの負担が大きい．一方後者は言語処理系が自動的に解放するのでその負担は小さい．

この不要になった領域を自動的に解放，再利用する機構はガーベジコレクタ(GC)と呼ばれている．GC はプログラマにとって非常に有用であるが，言語処理系には非常に重い処理でもある．そのため，これまでに多くの効率的な GC 手法が提案されてきた．世代方式 GC もその一つである．世代方式 GC では，メモリ領域を新世代領域・旧世代領域の２つに分割する．そして，新たに生成されたオブジェクトは，まず新世代領域に割り付けられ，何回かのＧＣを経た後，旧世代領域に移される（殿堂入り）．もし，適切な寿命予測に基づく GC 回収効率の予測が的確に行われれば，適切な世代分割を行うことによって GC の手間を削減でき，実行効率の向上が見込まれる．また，GC コストの見積もりが容易になり，プログラマは，より効率の良いプログラムを作成できるようになる．さらに，キャッシュ，メモリ階層を意識した世代の分割やオブジェクト配置も実現できれば，局所性を向上させることが出来る．これまで，寿命予測に基づくオブジェクト寿命分布モデル・GC 回収効率モデルとそれに基づく世代分割手法，及び局所性を向上させるオブジェクトの配置を実現する手法がいくつも提案されてきた．しかし，世代 GC をより効率よく行うためには双方を同時に考慮する必要があると考えられる．

そこで本研究では，まず以下の３つを行ない，効率の良い世代方式 GC の提案を行った．

1) データ寿命分布モデルと GC 効率モデルの提案．

まず，いくつかの KL1 プログラムで寿命分布を測定した結果，短寿命オブジェクトの寿命に対する確率密度分布は Gamma 分布，

$$g(x)=\frac{x^{k-1}e^{-x}}{\Gamma(k)},\quad \Gamma(k)=\int_0^\infty t^{k-1}e^t dt$$

に非常に近いことが分かった．そこで，短寿命オブジェクトと長寿命オブジェクトの比を $\alpha:(1-\alpha)$，長寿命オブジェクトの寿命を L（但し，L は十分大きな値とする．）として，全オブジェクトの寿命に対する確率密度分布を

$$f(x)=\alpha\frac{x^{k-1}e^{-x}}{\Gamma(k)}+(1-\alpha)\delta(L)$$

とした．（α は短寿命オブジェクト比率，k は短寿命オブジェクト寿命の広がりの指標．）
この確率密度分布と生存曲線のグラフを図 1,2 に示す．

次に，世代方式 GC の回収効率を示す指標である Mark/Cons 比（新世代 GC 時に回収されたオブジェクト量と，前回 GC 直後からのメモリ割付量の比）と新世代サイズとの関係のモデルを，上の寿命分布モデルから導き出した（図 3,4）．

その結果，新世代 GC の回収効率は新世代サイズが大きくなればなるほど改善されるが収束する値は $2(1-\alpha)$ である．また k の値が大きい（短寿命オブジェクトの平均寿命が長い）ほど M/C 比の収束は速いということが導かれた．

そして，このモデルは，実際の KL1 プログラムの振る舞いとほぼ一致した．

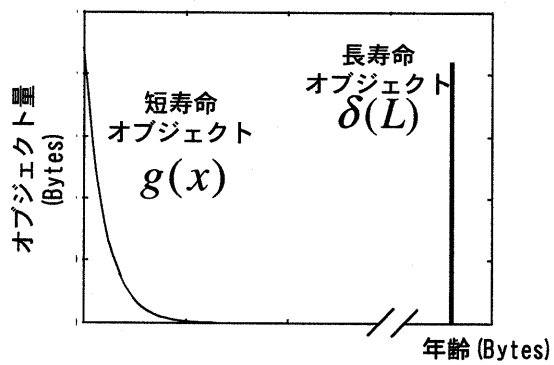

図 1. 寿命の確率密度分布

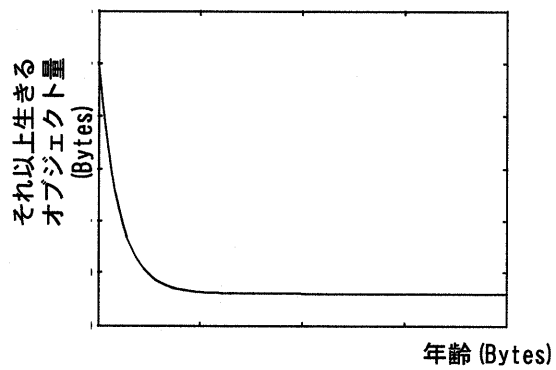

図 2. 生存曲線

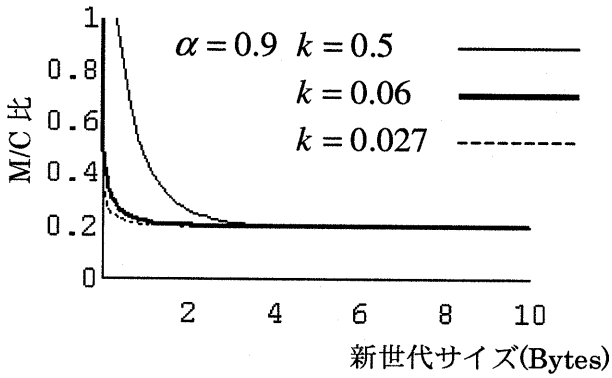

図 3. GC 回収効率のモデル（α固定）

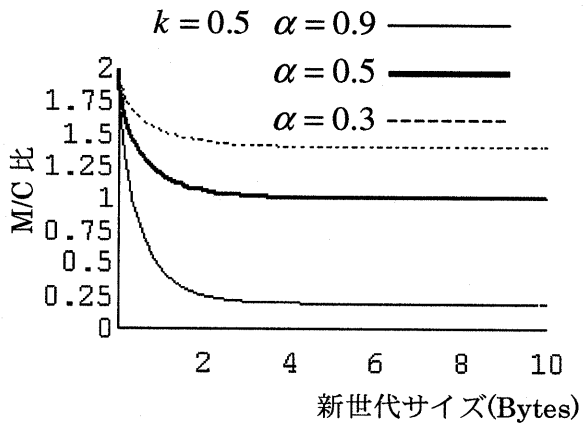

図 4. GC 回収効率モデル(k 固定)

2) 局所性を考慮した殿堂入り時期調節手法の提案.

次に，新世代サイズの違いによるキャッシュミス数と実行時間の関係を，いくつかの KL1 プログラムについて測定した．その結果，多くのプログラムでは，新世代サイズが大きくなるとキャッシュミスや TLB ミスなどの増加により，実行時間の低下が見られることが分かった.

一方，(1) の GC 回収効率のモデルから，新世代サイズを大きくすると，M/C 比の値が減少（＝GC 効率は改善）し，やがて $2(1-\alpha)$ に収束する．そのため，世代方式 GC では，新世代サイズを，M/C 比がほぼ $2(1-\alpha)$ で，局所性が悪くない値，すなわち，M/C 比と新世代サイズのグラフの傾きがある閾値よりも小さくなった値にするのが最も良いと考えられる.

3) 新世代サイズを自動調節する世代 GC 方式の提案.

並行並列論理型言語処理系 KLIC-3.003 版を，プログラム実行中に，

- 新世代サイズと M/C 比の関係を測定可能,
- 新世代サイズを変更可能,

に出来るようにし，実行時に動的に (2) の方針に従って新世代サイズを「M/C 比の変化が閾値以下になる値」とすることによって，非常に効率の良い実行が行えるようにした.

実行時のサイズ変更は，以下のように行うようにした.

まず，新世代サイズの初期値は一次データキャッシュサイズの半分とし，二回目の GC 時に新世代サイズを二倍にする．そして，新世代サイズを変更する毎に M/C 比と新世代サイズを記録し，前回新世代サイズ変更時の M/C 比，新世代サイズと比較した結果：

(A) $\dfrac{(previous\_M/C\_ratio)-(current\_M/C\_ratio)}{(previous\_M/C\_ratio)} > (Threshold)$

今回の新世代サイズ変更では，M/C 比が閾値以上改善しているので，次の GC 時には，同じ方向同じ変化幅で新世代サイズを変更する（図 5).

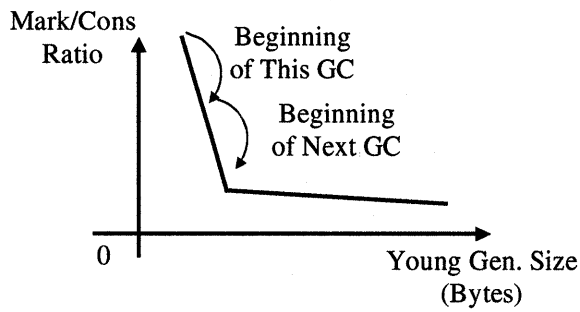

図 5 サイズ変更ルール(Case A)

(B) $\dfrac{(previous\_M/C\_ratio)-(current\_M/C\_ratio)}{(previous\_M/C\_ratio)} > -(Threshold)$

このとき，新世代サイズ変更では，M/C 比が閾値以上悪化しているので，次の GC 時には，反対方向，半分の変化幅で新世代サイズを変更する（図 6）.
なお，変化幅を半分にするのは，新世代サイズが過度の振動をしないためと，最適値に収束しやすくするためである.

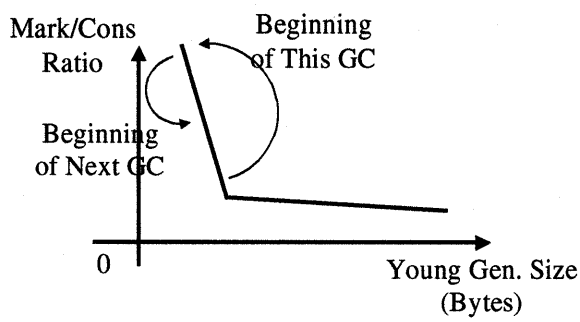

図 6 サイズ変更ルール(Case B)

(C) M/C 比の変化が閾値以下の場合

このとき，新世代サイズが十分大きく，M/C 比の値はほぼ$2(1-\alpha)$に収束してきていると考えられる．そのため，次の GC 時に新世代サイズをより大きく変更しても，M/C 比はあまり変化しないと考えられるので，新世代サイズを元に戻して新世代サイズ変更を一旦止める（図 7）.

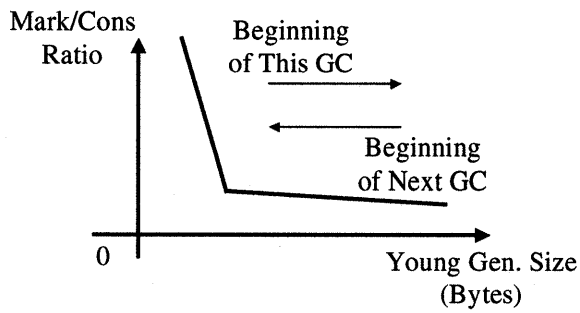

図 7 サイズ変更ルール(Case C)

このルールで新世代サイズを目標値に調節することが出来ると考えられる．しかし，最適な新世代サイズはプログラムの実行フェーズによって変わる事がある．そこで，新世代 GC10 回毎に新世代サイズを２倍，５０回毎に新世代サイズを１／２倍することにした．

サイズ変更の閾値は，小さすぎると新世代サイズが過度に変動してしまい，実行速度が低下してしまう．逆に大きくし過ぎると適切なサイズに変更される前にサイズ変更を止めてしまうため，GC 回収効率や局所性が悪化し，実行速度が低下してしまう．そこで，いくつかの KL1 プログラムにおいて，様々な閾値を設定して実行時間を比較し，その結果が最も良かった 0.45 を閾値とすることにした．

そして，並列論理型言語処理系 KLIC で，以下の３つの GC 手法での性能比較を行った．

- ２面の Stop-and-Copy GC (SC).
- 従来の世代 GC 方式 (GGC).
- 新世代サイズを調節する GC 方式(AGC).

実行環境は Intel Celeron 333MHz(1st D-Cache 16KB, 2nd cache 128KB), Linux 2.2.5. GGC と AGC の初期新世代サイズは 8KB （1 次データキャッシュサイズの半分), 旧世代サイズは 88KB とし，SC の初期ヒープサイズは片面 96KB とした．実行時間のグラフを図 8 に示す．

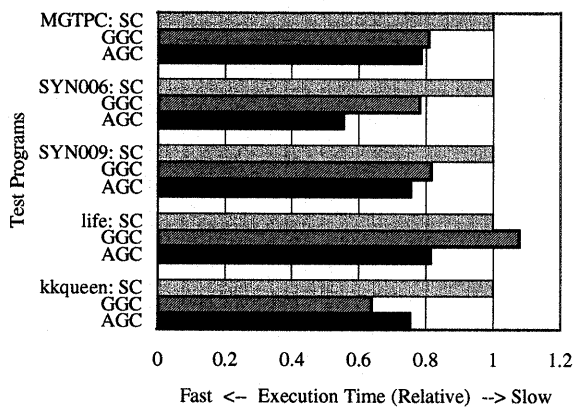

図 8 実行時間の比較

その結果，多くの KL1 プログラムにおいて，本手法が有効であることが示された．ただ，kkqueen でのみ AGC が GGC よりも遅くなってしまっている．このプログラムには，２つの

実行フェーズがあり，実行初期の短いフェーズでは大きなサイズが，そしてそれに続く長いフェーズでは小さなサイズが新世代サイズ調節の目標値となっている．AGC は，実行初期の段階で新世代サイズを拡大し，その後のフェーズで縮小を始めたのだが，GC 回数が全体で 298 回しか無かったため，最適な新世代サイズ(約 30KB)までの縮小が間に合わなかった．一方，GGC では，最初からあまり新世代サイズを拡大しなかったため，後半のフェーズで AGC よりも効率のよい GC が出来，実行速度も AGC を上回った．

まとめると，本研究では，

(1) データ寿命，GC コスト予測のためのモデルを作成．

単純なデータ寿命分布モデルから M/C 比(GC 回収効率)と新世代サイズの関係を導いた．これにより，実行時にデータ寿命分布を予測し，新世代 GC 回収効率を改善可能にした．

(2) 局所性を考慮した殿堂入り時期調節法の提案．

キャッシュ新世代サイズを大きくし過ぎるとキャッシュミスや TLB ミスなどにより実行速度が低下する．そこで，新世代サイズを M/C 比の変化が，ちょうど閾値以下になる値に設定することとした．

(3) 新世代サイズを調節する世代 GC 方式の提案．

M/C 比と新世代サイズを観測しながら，(2)で定めた値に新世代サイズを自動調節する世代 GC 方式を提案した．

そして，この GC 方式を並列論理型言語処理系 KLIC に実装し，いくつかのテストプログラムにおいて，他の GC 手法との比較を行った．その結果，本手法では多くのプログラムにおいて，新世代 GC の回収効率と局所性のバランスが取れ，世代 GC の効率化が実現された．