fork システムコールを用いた並列ガーベジコレクション

佐藤 憲一郎† 松井 祥悟††
† 神奈川大学大学院理学研究科 †† 神奈川大学理学部

1 はじめに

ガーベジコレクション (GC) は，プログラムが消費し使用済みとなったメモリを回収して再利用可能にするメモリ管理機能である．通常 GC は，割り当て可能なメモリが枯渇した時点で，メモリを消費して計算を行うプログラム (mutator) を停止し，メモリ回収プログラム (collector) を実行して使用済みのメモリを回収する停止-回収方式で行う．GC による停止時間や処理時間を短縮するために，世代 GC や並列 GC が開発されている．本論文では，fork システムコールを用いて，停止-回収型マークスイープ GC を容易に並列 GC へと変更する手法について提案する．

2 並列 GC のアルゴリズム

並列 GC は停止-回収型の GC を並列化したものである．並列 GC の基本的なアルゴリズムと並列化において問題となる点をあげる．

2.1 基本的なアルゴリズム

一般的な並列 GC は，*incremental update*(IU) 型と *snapshot-at-beginning*(SB) 型の 2 種類に分類できる [1]．IU 型は停止-回収型 GC を単純に並列化したものである．SB 型は GC 開始時にセルの状態をスナップショット写真を撮るように記録しておき，記録上で使用済みとなったセルだけを回収する．IU 型は SB 型よりも効率がいいが，セルへの追加の印付けのために mutator に対して予測不可能な停止が起こる．

2.2 バリア

停止-回収型 GC を単純に並列化した GC を考えると，GC の印付けやコピー中に mutator がセルのポインタを書き換えた時，使用中のセルに印が付かず，誤って回収される場合がある．それを防ぐために，mutator によるセルの書き込みに対してバリアを設け，GC へ通知する必要がある (図 1)．

3 純スナップショット型 GC

スナップショット型 GC は，通常スナップショット時にルートセットだけがコピーされるので，mutator によるセルの書き換えに対してバリアを設定する必要がある．しかし，セル空間全体を別の空間にコピーして，GC がこのコピーされた別空間のセルに対して印付けを行えば，バリアは必要なくなる．これを純スナップショット型 GC という [2]．

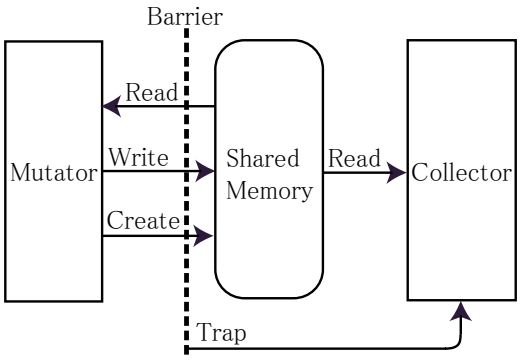

図 1: バリア

3.1 fork システムコールを用いた並列 GC

fork システムコールを用いた純スナップショット型 GC(Fork GC) を図 2 に示す．fork によりセル全体が子プロセス側へコピーされる．fork 実行後，親プロセスは mutator として通常の計算処理を続行する．子プロセスは GC プロセスとして停止-回収型マークスイープと同じ GC 処理を行う．GC プロセスはスイープ時に再利用可能なゴミセル情報を親プロセスへ通知し，exit() で終了する．

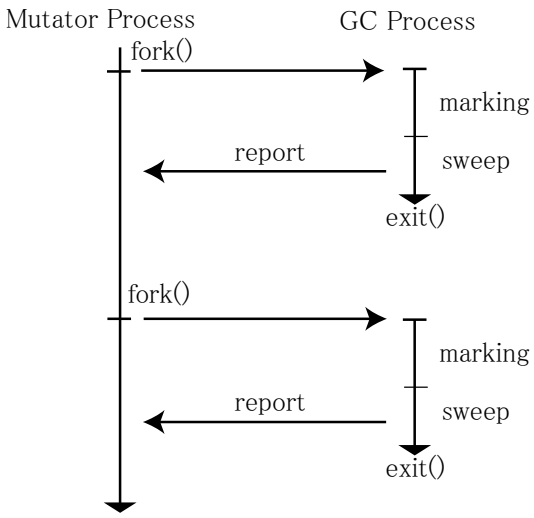

図 2: Fork GC の動作

Parallel Garbage Collection using fork system call
† Kenichiro SATO (ken16@ml.info.kanagawa-u.ac.jp)
†† Shogo MATSUI (sho@ml.info.kanagawa-u.ac.jp)
Graduate School of Science, Kanagawa University (†)
Department of Information Science, Kanagawa University (††)

コピーオンライト (copy-on-write, COW) 型の fork システムコールの場合，プロセスを生成するときには，メモリ空間をコピーしたように見せかけておく．ページへの書き込みが発生すると，はじめて実際にコピー処理を行いメモリ空間を分離する．このように fork システムコールを用いることで，容易にセル全体のスナップショットを撮ることができる．

3.1.1 mutator と GC プロセス間の通信

fork システムコールによって mutator と GC プロセスのメモリ空間は分離されるため，GC プロセスはスイープ時に mutator プロセスへゴミセル情報を通知する必要がある．代表的なプロセス間通信方法として，パイプと共有メモリがある．

パイプを使用する方法では，プロセス間でパイプを作成し，GC プロセスから mutator プロセスへパイプを通してゴミセル情報を送信する．共有メモリを使用する方法では，mutator プロセスと GC プロセスで情報を共有するためのメモリ領域を用意する．GC プロセスはスイープ時にゴミセル情報を共有メモリ上に書き込む．mutator はセルが必要となったときに共有メモリから取得する．つまり，共有メモリをフリーリストとして利用する．

パイプによる通信の場合，GC プロセスからゴミセル情報の送信があると mutator プロセスは処理を中断してフリーリストを作成しなければならない．この停止時間は GC プロセスが回収したセルの数 (最悪の場合セル全体) に依存するので予測不可能である．共有メモリによる通信の場合には，このような mutator プロセスの停止はない．

4 評価

Lisp 1.5 処理系で停止-回収型マークスイープ GC の euzak lisp に，パイプ通信型と共有メモリ型の 2 種類の Fork GC を実装した．

表 1 に停止-回収型マークスイープ GC から Fork GC へ変更したときのソースコード (C 言語) の変更量を示す．

表 1: ソースコードの変更量

Fork GC の種類	ソースコードの総変更行数
パイプ通信型	170
共有メモリ型	163

ソースコード全体は約 7700 行なので，変更は全体の約 2%である．主な変更箇所は，fork による GC プロセスの生成部分 (約 100 行) である．

次に，停止-回収型のマークスイープ GC と並列型 Fork GC の処理時間を比較した結果を表 2 及び表 3 に示す．処理時間の計測には boyer ベンチマークプログラム [3] を用いた．処理時間は GC を含めたベンチマーク全体の時間である．

表 2: 処理時間の比較 (multi CPUs)

GC の種類	user(s)	sytem(s)	elapsed(s)
停止-回収型	26.73	0.28	27.03
パイプ通信型	33.22	19.43	38.63
共有メモリ型	34.25	15.24	32.89

表 3: 処理時間の比較 (single CPU)

GC の種類	user(s)	sytem(s)	elapsed(s)
停止-回収型	22.87	0.21	23.15
パイプ通信型	32.72	18.21	50.71
共有メモリ型	29.32	14.50	45.19

fork システムコールによるプロセス生成コストや，セルの書き換えによる COW 処理のオーバーヘッドにより，停止-回収型よりも並列型 Fork GC の方が処理時間が長くなった．また，パイプ通信型はフリーセルリストの作成やパイプによるプロセス間の非同期通信に時間がかかるため，共有メモリ型の方が処理時間が短くなった．

5 結論

fork システムコールを用いることで，一般的な停止-回収型マークスイープ GC を容易に並列型 GC へと変更することができる．しかし，停止-回収型よりも並列型 Fork GC の方が遅くなった．今後は，COW 処理のオーバーヘッドやプロセス間通信のコストを改善する必要がある．

参考文献

[1] Paul R. Wilson. Uniprocessor garbage collection techniques. In *Proc. Int. Workshop on Memory Management*, No. 637 in Lecture Notes in Computer Science, pp. 17–32. Springer-Verlag, 1992.

[2] 松井祥悟, 田中良夫, 前田敦司, 中西正和. 並列ガーベジコレクションの実用化技術. 第 40 回プログラミング・シンポジウム報告集, pp. 159–170, 1999.

[3] Richard P. Gabriel. *Performance and Evaluation of Lisp Systems*. MIT Press, 1985.