様式 C-19

科学研究費補助金研究成果報告書

平成２２年　５月２８日現在

研究種目：若手研究(B)
研究期間：2008～2009
課題番号：２０７０００２３
研究課題名（和文）　組込みシステムのための実時間メモリ管理

研究課題名（英文）　Real-time Memory Management for Embedded Systems

研究代表者
鵜川 始陽（UGAWA TOMOHARU）
電気通信大学・電気通信学部・助教
研究者番号：50423017

研究成果の概要（和文）：組込みシステムでは，アプリケーションプログラムは長時間終了せずに動き続けるため，プログラムの実行とともに進行するメモリの断片化が問題となる．本研究では，リアルタイムアプリケーションの動作を妨げずに断片化を解消する方法として，「複製に基づくインクリメンタルコンパクション」を使ったガベージコレクタを提案した．本手法では，一部のデータだけ移動させて断片化を部分的に解消する．本研究では，効率よく断片化を解消できるような移動対象とするデータの選び方も提案した．

研究成果の概要（英文）：In embedded systems, since application programs run for a long time or forever, memory fragmentation, which becomes higher as the applications go along, is a serious problem. In this research, we proposed a garbage collector that reduces memory fragmentation without suspending the applications for a long time. The garbage collector uses replication-based incremental compaction. The compaction reduces memory fragmentation partially by moving a part of data. In this research, we also proposed an algorithm to choose data to be moved for efficient defragmentation.

交付決定額

（金額単位：円）

	直接経費	間接経費	合　計
２００８年度	700,000	210,000	910,000
２００９年度	600,000	180,000	780,000
年度			
年度			
年度			
総　計	1,300,000	390,000	1,690,000

研究分野：総合領域
科研費の分科・細目：情報学・ソフトウェア
キーワード：プログラム処理系，組込みシステム

１．研究開始当初の背景

組込みシステムにおいて，Java のような生産性の高い言語が使われる機会が増えてきた．高い生産性を支える技術のひとつに「ごみ集め」がある．しかし，通常のごみ集めは，ごみ集め処理の間アプリケーションの実行を止めるため，リアルタイムアプリケーションを実行するシステムでは使えない．そこで，

ごみ集め処理をアプリケーションと並行して少しずつ実行する，インクリメンタルごみ集めが研究されてきた．また，組込みシステムでは，アプリケーションプログラムは長時間終了せずに動き続けるため，プログラムの実行とともに進行するメモリの断片化が問題となる．そのため，データを移動させて連続した空き領域をつくるコンパクションが必須となる．

組込みシステム向けの代表的なごみ集めは次のいずれかに分類される．

（1） コンパクション行わないか，断片化が深刻になった時にはアプリケーションを停止してコンパクションを行う．メモリの割当て方法を工夫し，断片化しにくいようにすることもある．しかし，完全に断片化を防ぐことはできない．

（2） オブジェクトを固定長の大きさに分割して割当てる．Siebert による Jamaica VM はこの方法をとっている．これにより，メモリの断片化は問題とならないが，オブジェクトへのアクセス手順が複雑となり，実行時間のオーバヘッドと実装コストが大きくなる．

（3） ごみ集めの対象とならないメモリ領域を設け，実時間性が要求される処理は，ごみ集めの対象とならない領域だけを使う．Real-time Specification for Java により標準化され，Java の仕様に取り込まれている．ごみ集めの対象とならない処理は自分でメモリを管理する必要があり，Java の持つ生産性の高さを犠牲にしている．

（4） 実時間のごみ集めに，リードバリアを使ったコンパクションを組み合わせる． Bacon らによる Metronome はこの方法をとっている．リードバリアを使うことで，インクリメンタルにコンパクションをすることは可能だが，リードバリアは全てのオブジェクトからの読込み処理にオーバヘッドがかかる．一般に読込みは書込みより頻繁に行われ，実行時間に対するオーバヘッドが大きい．

（5） 複製に基づくごみコピー集めで，ごみ集めと同時に断片化も解消する．リードバリアは使わないが，コピーごみ集めを基にしているので，空間オーバヘッドが大きい．

２．研究の目的

組込みソフトウェアのためのメモリ管理として，次の特徴を備えたごみ集めアルゴリズムを開発することを目的とした．

（1） 高い実時間性を備える．多くの組込みアプリケーションでは，ネットワークやセンサからの入力やタイマなどの外部イベントにより計算が駆動され，一定時間以内に結果を出す必要がある実時間処理を含む．実時間処理中に一連のごみ集め処理を一括して行うと，その間はアプリケーションの実行が停止してしまい計算が終わるのが遅くなってしまう．そのため，ごみ集めはアプリケーションの実行の途中に少しずつ行うインクリメンタルごみ集めにする必要がある．

（2） 実行時間オーバヘッドが小さい．組込みシステム用のハードウェアはデスクトップやサーバ用コンピュータに比べて計算性能が低く，また，計算には電力の消費を伴うので実行時間オーバヘッドは小さく抑える必要がある．インクリメンタルごみ集めで使われる手法の一つであるリードバリアは，オーバヘッドが大きいため使わない．

（3） 空間オーバヘッドが小さい．組込みシステムでは，アプリケーションプログラムは長時間動き続けるため，最も大きな空間オーバヘッドの原因はメモリの断片化である．したがって，メモリコンパクションが不可欠である．

（4） マルチコア対応．近年の CPU のマルチコア化の流れを見越して，マルチコアに対応させることを考える．本研究では CPU コア毎に用途を固定しない，SMP としてのマルチコアを対象に考える．

３．研究の方法

我々のこれまでの研究で， Java VM である K Virtual Machine にコンパクションを行わない実時間ごみ集めであるスナップショットごみ集めを実装している．ここに複製に基づくインクリメンタルコンパクションを実装する．このコンパクションは，メモリの一部だけをコンパクションの対象とする．そのため，どの部分をコンパクションの対象とするかで効率が大きく異なる．実装した処理系でいくつかのプログラムを実行し，その動きを観察して，効率のよいコンパクション対象領域の決定方法を検討する．

４．研究成果

組込み実時間アプリケーションのためのごみ集めとして，「複製に基づくインクメンタルコンパクション」を使ったガベージコレクタを提案した．このガベージコレクタは，既存の segregated フリーリスト方式メモリ管理とスナップショットごみ集めに，本研究で開発した複製に基づくインクリメンタルコンパクションを組み合わせたものである．

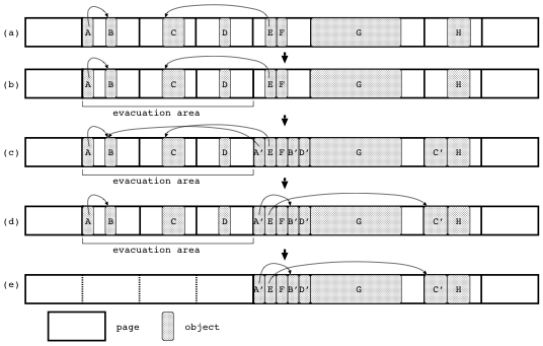

図 1 コンパクションの手順

空きメモリが少なくなると，まず，スナップショットごみ集めを行い，その後コンパクションを行う．

本研究で開発したガベージコレクタは，メモリを固定長の「ページ」という単位に分割して管理する．それぞれのページは均等な大きさの「ブロック」に分割する．ブロックの大きさには幾つかあり，ページ毎にブロックの大きさは異なってもよい．1 ページより小さなオブジェクトは，それを格納できる最も小さなブロックに割当てる．1 ページより大きなオブジェクトは連続するページ上に割当てる．

このようなメモリ管理は，断片化が起こりにくく，またメモリ割当てが高速に行え，オブジェクトの分割割当ても行わないため，組込みシステムに適している．しかし，アプリケーションを長時間実行していると，次の二種類の断片化が起る可能性がある．

（1） ある大きさの空きブロックが多くのページ上に分散し，空き領域の合計は十分あるのに，特定の大きさのオブジェクトしか割当てられなくなってしまう，「ブロック外部断片化」．

（2） 完全に空いたページがメモリ上に，ばらばらに分散し，大きなオブジェクトを割当てるための連続した空きページが確保できなくなってしまう，「ページ外部断片化」．

本研究では，これらの断片化を解消するために，複製に基づくコピーごみ集めの手法を用いた．しかし，コピーごみ集めのように，メモリ全体をコンパクションの対象とすると空間オーバヘッドが大きくなってしまうため，断片化が激しい部分だけに限定してコンパクションを行う．

図 1 にコンパクションの順を示す．（a）はコンパクション前の状態で，最初のページと 9 番目のページが空きページである．2 番目，4 番目，5 番目のページは，1 ページを 5 等分したブロックに分割されており，その一部だけにオブジェクトが割当てられている．3 番目と 8 番目のページはページの半分程度の大きさのブロックに分割され，やはり一部だけ使われている．また，6 番目と 7 番目を連続して使って，大きなオブジェクトが割当てられている．ここで，コンパクションを行うとする．

（b）のように，ある基準に従ってコンパクション対象の領域である「evacuation area」を設定する．最終的には evacuation area 内の全てのオブジェクトが evacuation area の外に移動し，evacuation area が連続した空きページとなる．

次に（c）のように，evacuation area 内のオブジェクトを evacuation area の外にコピーする．この処理には時間がかかるため，コピーしている間にもアプリケーションは動作している．アプリケーションがオブジェクトの内容を読み込む際，そのオブジェクトがコピーされていると，どちらから読み込むか分からない．本研究の方式では，どちらから読み込んでも最新の内容が読み出せるように，コピー元とコピー先の両方を最新の状態に保っておく．そのために，アプリケーションがオブジェクトに書き込む際に，そのオブジェクトがコピーを持っていたら，コピーにも同じ値を書き込む．

全てのブジェクトをコピーし終ったら，（d）のように，メモリを一通りスキャンし，evacuation area 内のオブジェクトを指すポインタを，そのコピーを指すように書き換える．この処理も時間がかかるため，スキャンしている間もアプリケーションは動いている．そのためオブジェクトへの書き込みはコピー元とコピー先の両方に行う．さらに，スキャンが終わった領域に，evacuation area 内へのポインタが書き込まれるのを防ぐために，ポインタを書き込む際に，evacuation area 内のオブジェクトへのポインタを書き込もうとしているかどうかを調べ，もしそうであれば，代わりにコピーへのポインタを書き込む．

メモリのスキャンが終わると，スタックや大域変数もスキャンし，evacuation area を指すポインタを置き換える．これが終わると evacuation area 内のオブジェクトを指すポインタはなくなるので，（e）のように，evacuation area 内の全てのページを空きページとする．Evacuation area は連続したページを選ぶため，広い連続した空きページが確保され，ページ外部断片化が解消される．

ブロック外部断片化がどの程度解消されるかは，連続した evacuation area を，メモリのどの部分に設定するかに依存する．本研究では，各ページの断片化の度合いを定義し，その度合いをページの順に並べた数列上の最大部分列和問題を解いて evacuation area を決定する方法を提案した．つまり，数列上の区間のうち，その区間に含まれる数の合計が最大となるような区間を evacuation area

とする．ただし，evacuation area を大きくとりすぎるとコピーごみ集めに近くなり，空間オーバヘッドが大きくなってしまうため，コンパクション開始時にあらかじめ evacuation area の上限を決めておき，それ以下の長さの部分列を探す．この問題は動的計画法により短時間で解けることが知られている．

これに加え，evacuation area 内のオブジェクトがどこにコピーされたかを記録する方法を工夫することで，我々の用いたベンチマークテストにおいて，

（1） 1 回のごみ集めによる停止を 200 マイクロ秒以下に抑える，十分な実時間性，

（2） 実時間でないマークコンパクトごみ集めに比べ 1%ー14%の実行時間オーバヘッド

（3） 51%ー186%と固定の 8KB の空間オーバヘッド

を達成した．実装の方式が大きく異なるため，実行時間について他の処理系と直接比較することが難しいが，空間オーバヘッドについては，リードバリアを使っている既存処理系と比べても同程度で収まっており，本研究の方式は有効であると言える．

本研究では，これ以外に，一部のオブジェクトを移動させずにコンパクションを行う方法の研究とマルチコア対応についての研究も行った．前者は移動できないオブジェクトが多い処理系である Ruby 言語の処理系に実装して実験した．Ruby では一定の成果があったものの，組込みシステムのような小規模では難しいことが判った．後者は本研究と同時期に海外の研究者によっても研究されており，その結果を調査すると，Java の意味論を実現するには，リードバリアを併用しなければ困難と分かった．

５．主な発表論文等
（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計 1 件）

① 鵜川始陽、Ruby における Mostly-Copying GC の実装、情報処理学会論文誌：プログラミング、査読有、2 巻、2009、1-12

〔学会発表〕（計 5 件）

① Tomoharu Ugawa, Hideya Iwasaki, Taiichi Yuasa, Improved Replication-Based Incremental Garbage Collection for Embedded Systems, ISMM 2010, 査読有, Toronto, (2010 年 6 月 6 日発表予定)

② 鵜川 始陽、湯淺 太一、複製に基づくインクリメンタルコンパクションを用いたガベージコレクタ、日本ソフトウェア科学会第 26 回大会、査読無、島根大学、(2009 年 9 月 17 日)

③ Tomoharu Ugawa, Masahiro Yasugi, Taiichi Yuasa, Replication-based Incremental Compaction, ISORC 2008, 査読有, Orlando, (2008 年 5 月 7 日)

〔その他〕
ホームページ等
日本ソフトウェア科学会高橋奨励賞受賞(第 26 回大会)：鵜川 始陽、湯淺太一、複製に基づくインクリメンタルコンパクションを用いたガベージコレクタ

６．研究組織
(1)研究代表者
鵜川 始陽（UGAWA TOMOHARU）
電気通信大学・電気通信学部・助教
研究者番号：20700023

(2)研究分担者
なし

(3)連携研究者
なし