Flash メモリを併用するスモールデバイス向けメモリ管理システム

井上 曜　千葉 滋

IOT で使われるようなスモールデバイスでは， メモリ使用量のオーバーヘッドの少ない言語が使われがちである．Flash メモリを拡張用に併用することで，高級言語をスモールデバイスで利用できるようになる．Flash メモリは書き込み耐性が低く，高レイテンシで消去操作の必要性や書き込みを一定単位でしか行えないという制限がある．そこで，Flash メモリを拡張用のメモリとして利用しても，書き込み限界に達しにくくするプログラミング言語用メモリ管理システムを提案する．Flash メモリの拡張は，仮想機械の read/write barrier を用いて，ページングとソフトウェア的なアドレス変換によって実現できる．そして，Flash メモリへの物理書き込みを減らすために，ページアウトの対象の選択をその対象の書き込み回数に応じて最適化したことと，分散した変更されそうなオブジェクトをガベージコレクション時にまとめることで，書き込み回数を減らすことを行った．

Software developers tend to use languages with low memory overhead when developing on small devices such as those used in the IOT. By using Flash memory as extended memory,softwared developers can use high-level languages on small devieces. Flash memory has many limitations such as low write dndurance, high latency, the need for erasing operations, and the ability to write only on a page-by-page basis. Therefore we propose memory management system for programming language that prevents the writing limit from being reached as much as possible even if the Flash memory is used as memory as extended memory. We have expanded the Flash memory by paging and software-like address translation using the read / write barrier of the virtual machine. Then, in order to reduce the physical writing to the Flash memory, we reduced the number of writes by optimizing the selection of page-out targets according to the number of writes of the target, and by grouping distributed and likely to be updated objects during garbage collection.

1　はじめに

IOT で使われるようなスモールデバイスでは， 搭載されている主記憶領域が小さく，メモリを多く消費するような生産性の高いデータ構造が使えないことがある．ソフトウェアを容易に開発するためには，メモリを多く使えるようにすることや， 自動で不要なオブジェクトを解放するガベージコレクションがあると良い．そして，IOT 用途でこのようなデバイスを使う場合は，多量にばらまくことがあるため，できるだけデバイスの価格コストを押さえる必要もある．

安価にメモリを増やすには，スモールデバイスの補助記憶装置である Flash メモリを仮想記憶を用いて拡張用のメモリとして使えば良い．ただし，スモールデバイスには，仮想的なアドレスを物理アドレスに変換するデバイスである MMU や，Flash が書き込みに対して脆弱であり，消去や書き込みに制約があるため，簡単に使うことはできない．さらに，仮想記憶とガベージコレクションのトレース操作は相性が悪いため，この欠点を相殺するような機能が必要である．

本論文では，Flash メモリの書き込みを低減するようなページングアルゴリズムと書き込み回数を考慮したコンパクションアルゴリズムを提案する．まず，Flash メモリのアドレスはソフトウェアでの変換を用いて，仮想記憶を使えるようにした．オブジェクトの書き込み回数や読み込み回数を記録し，ページアウト時への書き込み回数を押さえるような Conditional bit LRU というアルゴリズムを実現した．そして，

A memory management system using Flash memory for small devices

Akira Inoue Shigeru Chiba, 東京大学大学院情報理工学系研究科, Graduate School Information and Science Technology， The University of Tokyo .

Mark and Compact 時に書き込みのあったオブジェクトをまとめるようなアルゴリズムも実現した．

本提案手法を適用したメモリ管理システムとそれを組み込んだ Scheme 処理系を実装し，仮想的な環境で，Flash への書き込み回数とページフォルト数を計測した．結果は，Scheme 処理系と提案手法との相性が悪く，LRU と比較して書き込み回数がわずかにしか良くならなかった．そこで，スモールデバイス向け Ruby 処理系である，Mruby にも提案手法を組み込んで実験を行う予定である．

本研究は，昨年の発表の改良である．以前のシステムにたいして，オブジェクトが移動，Condtion bit LRU を新たに提案する．それから，Scheme 処理系での実験を追加した．

2 背景

Flash メモリは書き込み耐性が低く，書き込みのレイテンシが大きく，Flash メモリは書き込む前にブロック単位の領域削除が必要であるため， 単純に拡張用のメモリとして使うことはできない．また，スモールデバイスの多くは，仮想アドレスを実アドレスに変換するデバイスである MMU を備えていないため，仮想記憶を仮想機械の read barrier などのソフトウェア的なアドレス変換などを用いる必要がある．ソフトウェアを容易に開発するには，ガベージコレクションが必要だが，仮想記憶とガベージコレクションのトレース操作は相性が悪いため，この欠点を相殺するような機能が必要である．

Mruby や Micro Python，Picobit [2] など，スモールデバイス向けの言語処理系はいくつか存在するが，Flash メモリを併用して仮想的にメモリを増やし，Flash メモリの欠点を軽減するようなメモリ管理システムを搭載している言語処理系は知る限り存在しない．多くの処理系は，実装をリソースの消費しないようなロジックに置き換えるようにしたり，仕様を一部削ることでスモールデバイス上で動くようにしている．これらの処理系では，リソースを多く消費するような機能を使うにはメモリを増やさなくてはならず，システムのコストが高くなってしまう．

3 Conditional bit LRU と Mark and classified compact

Flash メモリは書き込み耐性が低いため，単純に拡張用のメモリとしてつかうことができない．本研究では，それを解決するために Conditional bit LRU と Mark and classified compact という 2 つの方式を搭載したメモリ管理システムを提案する．提案するメモリ管理システムは，ユーザに malloc のような動的確保とアドレス変換の関数を提供し，メモリ不足を検出すると自動的に不要なオブジェクトの回収を行う．この動的確保とアドレス変換関数を用いて，プログラミング言語のランタイムを実装すれば Flash メモリを追加のメモリとして扱えるプログラミング処理系を作ることができる．

3.1 メモリ管理システムの概要

本システムの領域の使い方の概要を図 1 に示す．ヒープ領域は，RAM 上にある volatile 領域と Flash メモリ上にある non-volatile 領域の 2 つに分けられる．volatile 領域上のオブジェクトは直接読み書きできるが，non-volatile 領域上のオブジェクトは，ページ単位で RAM 上のページ領域上にコピーして使う (ページイン)．ページ領域上にページがあふれるときにページを項 3.2 のアルゴリズムによって 1 つ選択し，non-volatile 領域に書き戻す (ページアウト)．それから，non-volatile 領域上のオブジェクトのオブジェクトヘッダは頻繁に変更されるため，RAM 上のヘッダ領域に配置した．

本システムでは，non-volatile 領域上のアドレスは，仮想的なアドレスとして表現している．ポインタの末尾ビットをフラグとして用いて各領域の区別や、固定長整数などのその他の値と区別するようなことを行う。参照バリアを用いて，non-volatile 領域上のオブジェクトが参照を検出し，対象のページをページインし，ページ領域にコピーされたオブジェクトのアドレスに変換して返す．

システムが動的にメモリ領域の確保が必要になった時，優先的に volatile 領域から確保する．volatile 領域が不足すると，volatile 領域内の不要なオブジェクトを

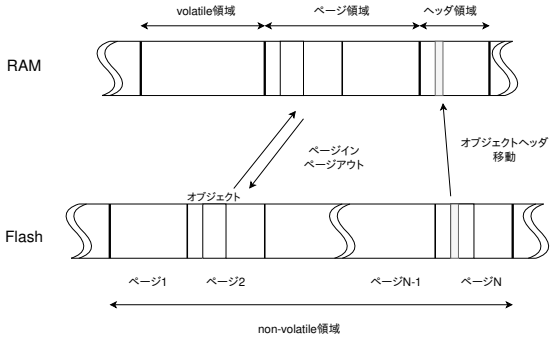

図 1　システムの概要

回収する volatile-GC を行う．この部分な GC のため，non-volatile 領域から volatile 領域への参照が生じた場合は，そのオブジェクトのアドレスの remembered set を作成しておく．volatile-GC 時に volatile 領域上の書き込みが行われていない，non-volatile 領域から参照されていないオブジェクトを non-volatile 領域に移動する．volatile-GC を行ってもオブジェクトを確保できなければ，non-volatile 領域から確保する．non-volatile 領域からオブジェクトを確保するには，non-volatile 領域からコピーしたページ領域上のページに一時的に確保され，そのページを non-volatile 領域に書き戻す時に non-volatile 領域に反映する．non-volatile 領域からも確保できなくなった場合は，volatile 領域と non-volatile 領域のすべての領域から不要なオブジェクトを解放する full-gc を行う．

本システムでは，提案手法のために書き込み，生存期間，マークビットをオブジェクトが持つヘッダに記録している．オブジェクトヘッダは，2bit を non-volatile 領域で書き込み回数を 3 回まで記録できる値，1bit を volatile-GC を生存したかのフラグ，残りをマークビットと型情報である．

3.2　Conditional bit LRU

Flash メモリは，書き込みに対して脆弱であるため，LRU のような通常のページアウトアルゴリズムでは短期間で Flash メモリの寿命に到達してしまう．そこで，書き込みのあったページを極力避け，ページミスもできるだけ低減するようなページアウトアルゴリズムである，Conditioal bit LRU を提案する．

図 2　Conditional second chance FIFO ページアウトの順序

最近参照されていない書き込みのないページ →
書き込みのない新規割り当てでないページ →
書き込みのない新規割り当てページ → 書き込みのあった新規割り当てでないページ →
新規割り当てページで空き領域が多いページ

Conditional bit LRU は，LRU の近似アルゴリズムである reference bit LRU を改良したものである．各ページには，reference bit と write bit と allocate bit の 3 種類のビットを用いる．ページ内のオブジェクトが参照されると，reference bit を立てる．また，一定時間おきに reference bit をオフにするページの書き込みは，オブジェクトの要素の更新と，初期化時に値をセットするものの 2 種類存在する．オブジェクトの更新は，write bit を立て，確保時の初期化を allocate bit を立てる．

conditional bit LRU では，最初に write bit と allocate bit がオフのもので，reference bit がオフのものを選択し，次に reference bit がオンのものをページアウトする．その次に，allcoate bit がオンのもので，使用率が一定以上のものをページアウトする．その次に，write bit がオンのものをページアウトする．最後に，それ以外をページアウトする．

3.3　Mark and classified compact

書き込みのあるオブジェクトが分散してページに置かれていると，最も都合良く書き込みのあるページを選択したとしても，書き込みが発生するようになる．ガベージコレクション時に書き込みのあるページをまとめることで，ページアウトの選択を都合の良い状態で行うことができるようになる．ガベージコレクションのトレース操作はページングと相性が悪いが，ページアウトによる書き込みの低減を行うことでこれを相殺できる．オブジェクトをまとめるには Mark and compact や Copy GC などの移動を伴うガベージコレクションを併用する必要がある．メモリの無駄を作

る Copy GC は，小さいメモリ空間しかない本システムに向かないため，Mark and compaction をベースとした．そして，Mark and compact に書き込みのあったオブジェクトをまとめる，Mark and classified compact を提案する．

Mark and classified compact は，Mark フェーズと Compact フェーズからなる．Mark フェーズでは，ビットマップマーキングを用いて，RAM 上のヘッダ領域にある生存しているオブジェクトに対応するビットを立てる．Compact フェーズは，アルゴリズム 3.3 にあるような手順で不要なオブジェクトの解放と再配置を行う．このアルゴリズムは，Lisp2 アルゴリズムというコンパクションアルゴリズムを改良したものである．まず，オブジェクトの書き込みフラグをもとに，オブジェクトを 2 つのキューに分ける．そして，ページ単位で書き込みができるタイミングで，Flash メモリに書き出す．

Algorithm 1 non-volatile 領域用の compaction algorithm

```
1: writtenQueue ← 空の Queue
2: noWrittenQueue ← 空の Queue
3: for page = flash の各ページ do
4:     for obj = page の各オブジェクト do
5:         if alive?(obj) and changed?(obj) then
6:             Push(writtenQueue, obj)
7:         else if alive?(obj) and not changed?(obj) then
8:             Push(noWrittenQueue, objectPtr)
9: addrTable ← 空の辞書
10: for page = flash の各ページ do
11:     if not empty?(writtenQueue) then
12:         while do
13:             obj ← pop(writtenQueue) または obj ← pop(noWrittenQueue)
14:             if obj が page にかけない then break
15:             write(page, obj)
16:             set(addrTable, obj の旧アドレス , obj の新アドレス ))
17: for obj = すべての object do
18:     for member = 0tosizeof(object) do
19:         if haveKey(addrTable, obj[member]) then
20:             obj[member] ← addrTable[obj[member]]
```

4 実験

Conditional bit LRU と Mark and classified compact が妥当であることを検証するために，Scheme 処理系を実装し，Flash 書き込み回数とページフォルト数を計測した．これらの計測は，表 1 の設定でコンピュータ上の仮想的な環境で行った．vector として表現した 16x16 のサイズの画像を 10 回繰り返す k-means 法で減色するプログラムを用いた．

表 1　実験の設定

volatile 領域のサイズ	1024byte
ページのサイズ	512byte
ページ数	512
ページ領域のサイズ	4

提案手法である Mark and classified compact と conditional bit LRU といくつかのコンパクション方法とページングアルゴリズムを変えた次の 4 つのもので比較した．

1. 提案手法: Mark and classified compact と Conditional bit LR
2. Conditional bit LRU: Mark and compact と Conditional bit LRU
3. Clean-first LRU: Mark and Classified compact と Clean-first LRU
4. Mark and classified compact: Mark and classified compact と LRU

4.1 Scheme 処理系

本研究のために，提案手法のメモリ管理システムを組み込んだ Scheme 処理系を実装した．M5Stack というデバイスや通常のコンピュータ上で動作する．この Scheme 処理系は，バイトコードインタプリタ方式

の仮想マシンとそのマシンに対するコンパイラからなる．継続の実装のために継続渡しスタイルへ変換する方式を採用している．継続の生成を定数時間で行うことができるが，代わりに実行中にクロージャを大量に生成するようになっている．

4.2 実験結果

Flash メモリへの書き込み回数を表 2 に示す．提案手法が最も書き込み回数が少ないが，言語処理系の性質でクロージャや一時的データの作成が多く，確保と回収が頻発しているため，大きな差が生まれなかった．ページフォルトの回数を，表 3 に示す．LRU が最もページフォルトが少なく，その次が提案手法であった．Flash メモリへの書き込みは，消去操作を伴うことや書き込みの時間が読み込みより大きいため，LRU より増えたミスペナルティよりも提案手法で削減した書き込みの時間コストの方が大きい結果であった．

表 2　実験結果 (書き込み)

1:提案手法	26813
2:Conditional bit LRU	27182
3:Clean first LRU:	27372
4:Mark and classified compact	27553

表 3　実験結果 (ページフォルト)

1:提案手法	45506
2:Conditional bit LRU	46693
3:Clean first LRU:	46402
4:Mark and classified compact:	39254

5 関連研究

5.1 KG-N KG-W

PCM など，Flash メモリのように書き込み耐性が低い不揮発性メモリがいくつか存在する．KG-N と KG-W [1] は，は PCM と DRAM を併用するメモリ管理手法である．PCM は，書き込みの耐性は Flash メモリほどではないが低いが，消去操作が不要でバイト単位での書き込みが可能である点が Flash メモリと異なっている．KG-N と KG-W も書き込み回数を削減することを目的にしている．

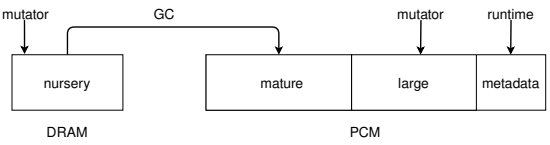

図 3　KG-N

KG-N は，世代別 GC をベースにしている．図 3 のように，若いオブジェクトに書き込みが多く発生するため DRAM に割り当て，古くなったオブジェクトを mature に割り当てる．大きいオブジェクトはコピーにコストがかかるので，最初から PCM の large 領域に割り当てる．ランタイムのメタデータは，PCM の metadata 領域に置く．

KG-W も，世代別 GC をベースにしているが，書き込みの監視を行ない，長く生存したオブジェクトを書き込み回数に応じて Flash メモリか DRAM に振り分ける．図 4 のように，一定期間生存したオブジェクトは DRAM の observer 領域に移動する．observer 領域では，オブジェクトの書き込みを監視し，一定期間生存したものの内，書き込みが多いものは PCM の mature に移動し，それ以外を DRAM の mature に移動する．大きいオブジェクトは，KG-N 同様 PCM の large 領域に割り当てるが，書き込み回数を動的に監視し，書き込みが多いと DRAM の large 領域に移動し，DRAM の large 領域で多く書き込まれた場合は，PCM の large 領域に移動する．KG-W は，KG-N よりも PCM への書き込み回数の削減はできるが，書き込みの監視の分計算時間のオーバーヘッドは大きい．

KG-N/W のアルゴリズムを本研究の対象である flash メモリにそのまま当てはめることはできない．なぜなら，この手法は書き込みと読み込みがバイト単位であることを前提としており，書き込み前に消去が必要な記憶デバイスを考慮していないからである．

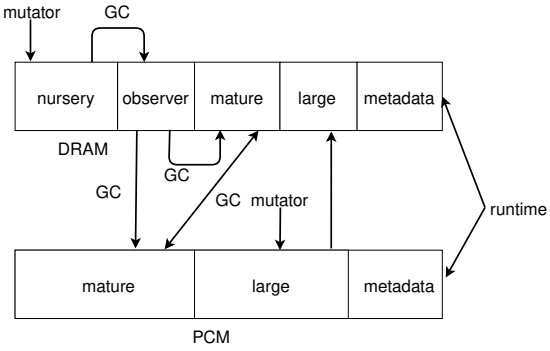

図 4　KG-W

6　まとめと今後の課題

本論文では，スモールデバイスの Flash メモリを拡張用に併用することで，高級な機能を利用できるようにする方法を示した．拡張方法は，ソフトウェア的なページング用いた仮想記憶を用いた．Flash メモリは書き込みに対して脆弱である問題があるため，書き込みをできるだけ少なくする方式を採用したメモリ管理手法を提案した．書き込みのあったページを追い出すことを遅延させる Reference bit LRU と Mark and compact を改良した Condtional bit LRU と compact 時に書き込みのあったオブジェクトをまとめることで Reference bit LRU の効果を最大化することを実現した．

実験では，処理系と提案手法の相性が悪く十分な効果を示すことができなかった．現在，組み込み向け Ruby 処理系である，Mruby をベースにしたシミュレーターを実装中で，今後実験で再度評価を行う予定である．

参 考 文 献

[1] Shoaib Akram, Jennifer B. Sartor, K. S. M. and Eeckhout., L.: Write-rationing Garbage Collection for Hybrid Memories., *ACM SIGPLAN Conference on Programming Language Design andImplementation*, (2018).

[2] Vincent St-Amour, M. F.: PICOBIT: A Compact Scheme System for Microcontrollers, *IFL'09: Proceedings of the 21st international conference on Implementation and application of functional languages*, (2009).