不揮発性メモリの併用に特化したIOT向けプログラム言語ランタイム

井上 曜　千葉 滋

回路サイズ，価格コスト，消費電力などの制約で使えるリソースが少ない IOT 向けデバイスでは，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．補助記憶装置である Flash メモリを併用すれば，高級な言語をこのようなデバイスで動かせるが，Flash メモリの書き込み回数には制限がある．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バーチャルマシンを提案する．書き込み制限の問題に対して，ページングで管理された Flash メモリ領域への物理書き込みを低減するために，ページアウトの対象を書き込み回数に応じて最適化したことと，メモリ確保とメモリ解放機構を改良することで，書き込みの多いオブジェクトをまとめることを行った．

Programmer uses programming language with low memory overhead for Small size, low cost or low power consumption IOT devices that don't have have many resources. By using Flash memory, which is the auxiliary storage device of the IOT device, together with RAM, it is possible to use high-level languages for IOT devices, but it is not possible simply because the number of writes to Flash memory is limited. Therefore, we propose a virtual machine for programming language that uses Flash as memory to delays reaching the writing limit as much as possible. We optimize the pageout selection algorithm to reduce the number of writes to the Flash memory area managed by pagins and improves the memory allocation and garbage collection algorithm to separate objects that write a lot and those that don't.

1　はじめに

回路サイズ，価格コスト，消費電力などの制約で使えるリソースが少ない IOT 向けデバイスでは，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．メモリを仮想的にでも多く使うことができれば，メモリを多く消費するが，生産性の高い機能を使うことができる．そして，IOT 向けアプリケーションの開発者は容易にプログラミングを行うことができるようになる．

仮想的にメモリを増やすことは，IOT デバイスに搭載されている Flash 領域を仮想記憶として使えば良い．ただし，IOT デバイスではメモリ管理ユニットが搭載されていないという点や Flash には一定回数書き込むとその領域が使えなくなるという性質によって単純な方法では Flash 領域は仮想記憶として使うことができない．例えば，ATmega328P では書き込み回数は 10,000 回で頻繁に Flash 領域にあるオブジェクトを変更させるようなプログラムだとすぐに使えなくなるだろう．このような環境でも Flash を併用してメモリを仮想的に使うことのできるようなメモリ管理システムが必要である．そこで本論文では，Flash を併用してもメモリの書き込みをできるだけ抑えることができるメモリ管理機構を提案し，容易に IOT デバイスで開発できるようなプログラミング言語ランタイムを実現した．

2　背景

通常のメモリ領域以外の記憶領域である Flash をメモリと同じ用に使うには仮想記憶のような機構を作れ

Programming language runtime for IOT devices with non-volatile flash memory.

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. .

ば良い．Flash 上のアドレスはメモリアドレスと区別されているため，仮想的なアドレスに置き換えて通常のメモリアドレスのように使える必要がある．一般的な環境で仮想アドレスを実アドレスに変換する場合，メモリ管理ユニットを使うが，IOT デバイスではメモリ管理ユニットを搭載していないものが存在するため，仮想マシン上で read/write barrier を用いて実際のオブジェクトが置かれているアドレスへと変換するといった手法が必要である．このように read/write barrier が使えるプログラミング言語処理系であればメモリ管理ユニットが存在しないハードウェアでも Flash を追加のメモリ領域でも使うことができる．しかしそれだけでは Flash に一定回数書き込みを行うとその領域が使えなくなるため，現状では書き込み回数が多くなりすぎ実用的ではないことが課題である．

3 Conditional second chance FIFO と Mark and classified compact

本論文ではガベージコレクションつきの仮想マシンを使ってメモリ参照を抽象化することでページング方式の仮想記憶のようなメモリ管理システムを実現した．さらに，page out するページの選択方式として Second chance FIFO を改造したものを用いることと，ガベージコレクション時に compaction することで書き込み回数の低減を図ることを可能にした．

本システムはメモリ上にある容量の小さい volatile 領域と Flash 上にある volatile 領域より大きい non-volatile 領域がある．volatile 領域は容量が小さいがページングを必要とする non-volatile 領域と違い直接参照であるため高速であり変更に制限がないためこの領域に空きがあれば優先的に volatile 領域から確保する戦略をとる．volatile 領域は貴重であることと全領域の頻繁なガベージコレクションは計算コストが大きくと Flash への書き込みが生じるため volatile 領域だけの Mark and sweep のガベージコレクションが必要である．volatile 領域だけのガベージコレクションを使うには，non-volatile 領域に volatile 領域のオブジェクトを書き込みを行った時に仮想マシンの write barrier に remenberd set へ登録する機構の追加によって使用可能になる．non-volatile 領域は仮想マシンの read/write barrier によって必要に応じてページ単位でメモリにページをコピーする (ページイン)．そして，ページをコピーする領域が RAM になくなった場合は優先度の低いページをメモリから取り出し変更が必要であれば Flash へ反映させせる (ページアウト)．

提案手法としてページアウト時に書き込み頻度を下げるためにメモリ管理を工夫した．まず，書き込みのあるページをできるだけメモリ上に保つためのページアウト対象の選択アルゴリズムを改良した．さらに変更のあるオブジェクトだけをメモリ上にあるコピーされたページ内に保つためにガベージコレクション時の compaction で変更のあったオブジェクトをまとめるような機構も作成した．

3.1 volatile 領域と non-volatile 領域

オブジェクトが割り当てられる領域は，RAM 上に連続した区間で切り出された volatile 領域と volatile 領域に空きがないときに使われる，Flash 上の non-volatile 領域の 2 つが存在する．図 1 に各領域の使われ方を示す．non-volatile 領域は，仮想記憶のページングと同様に，ページ単位で RAM 上に確保されたページサイズの定数倍の大きさの領域のページ置き場にコピーして読み書きを行っている．本システムではポインタの末尾ビットに non-volatile 領域か判定するための情報が入っており，参照時に write-barrier によってページ置き場に置かれたページへのポインタを返す．それ以外は non-volatile 領域上のオブジェクトは volatile 領域上のオブジェクトと表現は変わらない．新しく non-volatile 領域にオブジェクトを割り当てたい場合は，ページ置き場に Flash 上の配置予定のページのアドレスを紐づけた空のページを置き，そこにオブジェクト用の領域を確保しておく．object は，volatile 領域に要求されたサイズが確保できるだけの余裕があった場合優先的に volatile 領域から確保される．volatile 領域のみガベージコレクションするために，non-volatile 領域から volatile 領域への参照を作った場合はそのオブジェクトのアドレスの remenberd set を作っておく．そして，volatile 領域だけのガベージコレクションしても領域不足で確

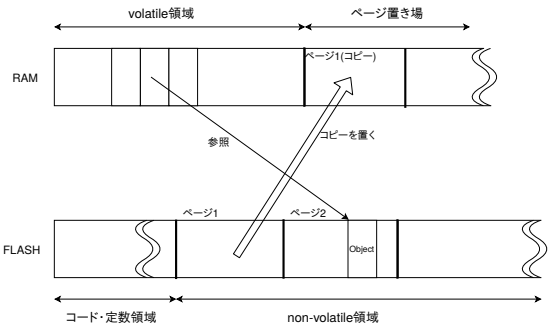

図 1　提案システムの空間の使われ方

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

保に失敗した場合，non-volatile 領域から確保する．non-volatile 領域上に存在するオブジェクトは，Flash 上のアドレスで管理されている．データを参照する時は，都度 RAM 上にコピーされた領域上のポインタに変換するコストがかかる．つまり，ページ置き場に存在しないページ上にあるオブジェクトの参照は，ページイン (ページアウト) とアドレスの変換で volatile 領域へのアクセスよりはるかに大きくなる．

3.2　Conditional second chance FIFO

ページアウトでは，基本的にはページミスを低減するようにページを選択しなければならないが，書き込みのあ ったページのページアウトは Flash の寿命を縮めるため，それも避ける必要がある．書き込みが起こるページアウトは，set-car!や set-vector!などによって変更されたオブジェクトを含んだページと新規割り当て用のページアウト時に Flash に反映させる必要のあるページの 2 つで生じる．この 2 種類のページの追い出しの優先度を落としつつ，ページミスを低減するようなページアウトアルゴリズムが必要である．Second Chance を改良した，Conditional Second Chance FIFO を提案する．

まずページに対して，読み書きしたときに立つ参照フラグと書き込みを行った write フラグと新規当て用のページ用のフラグの 3 つを用意する．参照フラグは，FIFO の先頭に出た時点でオンならば，オフにして FIFO の最後尾に置かれ，それ以外ならばページアウトされる．この時点でページアウトされなければ，write フラグと新規割り当てページと割り当てられていない区間の割合を元にページアウトの対象を選択する．具体的には図 2 のような順序でページアウトされる．

3.3　Mark and classified compact

3.2 のページアウトの最適化は変更のあったページと変更のないページが別れていなければどれをページアウトしても Flash への書き込みが生じるため意味がなくなる．したがって，今後変更が生じるオブジェクトとそうでないものを分けてページに配置する必要がある．これは，過去に変更のあったものは今後も変更されやすいという仮定から，Mark and compact のようなメモリの再配置を行うガベージコレクションと連動して再配置時に過去の変更記録をもとに今後変更されやすいオブジェクトをまとめることで解決できると考えられる．そこで mark and compact の改良として，non-volatile 領域のための Mark and classified compact を提案する．

mark フェーズでは，Flash への書き込みをしないために Mark ビットを RAM 上に配置することでページは読み込みだけで済む．compaction フェーズは，アルゴリズム 1 で示される．non-volatile 領域の各ページの生存しているオブジェクトを過去に書き込みが発生したかに基づいて 2 種類に分ける．そして，書き込みが発生していないオブジェクトから flash の各ページに書き出し，その後書き込みが発生したものを flash の各ページに書き出す．実装では，すべてのオブジェクトを読むまで書き出しを待つ必要はない．1 ページ分の容量分のオブジェクトをどちらかの Queue に存在する場合は書き出すようなことをする

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]]
```

ことで，書き出し待ちの 2 つの Queue によるヒープの使用量を低減することができる．

3.4 本方式の汎用性

本方式はバーチャルマシンや事前にネイティブコンパイルする言語において read/write-barrier と動的確保部分を置き換えることで実現できる．バーチャルマシンでの実装は本研究で実現できた．ポインタを直接操作しないような言語用のコンパイラで提案手法を実現するにはまず提案手法のメモリ管理を組み込む必要がある．そして，メモリを動的確保する関数を提案手法のものに置き換えることと，オブジェクトを参照する時に read/write barrier を挿入するようなコードを出力して non-volatile 領域かを判定すればよい．

4 実装

本システムは Scheme 用のバーチャルマシン (レジスタマシン) として実装されている．将来的には R7RS (large) 準拠させるつもりだが，現在は R7RS subset 程度の機能である．処理系の中間コードは CPS 変換したもので，そのまま逆 CPS 変換せずにターゲットのバーチャルマシンに変換した．Scheme からバーチャルマシンへのコンパイラも実用のために用意した．Aruduino Uno 程度のスペック (表 1) で動作することが目的ではあるが，現状は開発のしやすさから M5Stack Basic (表 2) で動作するものになっている．

表 1 Arduino UNO のスペック

CPU	16MHZ
RAM	2KB
FLASH	32KB
EEPROM	1K

表 2 M5Stack Basic のスペック

CPU	240MHZ
RAM	520KB
FLASH	16MB

5 関連研究

5.1 CFLRU

フラッシュメモリ用のページ置換アルゴリズムは他にもあり，基本的なものでは LRU の改良である，CleanFirst LRU (CFLRU) [1] がある．Flash の書き込み時間は読み込み時間より大きく，書き込みのあるページと読んだだけのページを区別せずにページアウトすると時間的な効率が悪くなる．そこで，書き込みをしていないページ (Clean page) を優先的に RAM から追い出すことをしている (Clean First)．すべてにおいて Clean page を優先的にページアウトし続け

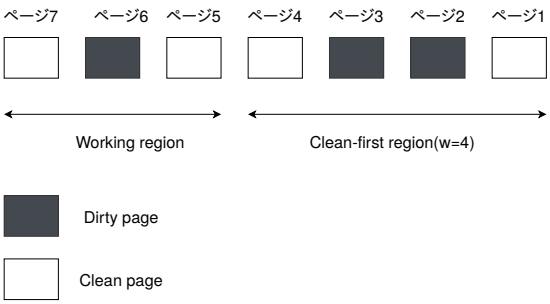

図 3　CFLRU のアルゴリズムのサンプル

るとキャッシュヒット率が著しく低下するため，LRU の先頭 w ページ分を Clean First とし，それ以外は書き込みをしているかどうか区別せずページアウトするようになっている．例えば，CFLRU が図 3 のような状態の時，ページアウトされる優先順位は，(ページ 1，ページ 4，ページ 2，ページ 3，ページ 5，ページ 6，ページ 7) となる．

CFLRU は書き込みのあったページをできるだけページアウトしない方針である．今回提案するシステムでは，確保されたが Flash に一度も書き込まれていないオブジェクトを含むページもページアウト時に Flash への書き込みが発生するページとして扱われる．このような確保用のページは書き込みは 1 度は発生するがオブジェクトに変更を加えたわけではないため，今後も Flash への書き込みが起こりやすいページとは限らないということを考慮が必要である．Conditional second chance FIFO は書き込みを抑えるという点では CFLRU と目的は同じだがプログラム言語ランタイム用のページングとして確保用のページの性質もページアウトの優先順位の要素として加えた点が異なっている．

5.2　Second chance FIFO

Second chance FIFO は FIFO を改良したアルゴリズムで参照された時に立てられる参照ビットが各ページに存在しており，FIFO の先頭のページが参照ビットが立っていた場合，参照ビットをクリアして最後尾に繋げるアルゴリズムである．提案手法である Conditional Second chance FIFO はこれに書き込みフラグと初期化用のページのフラグを付けて，参照ビット以外にもそのフラグを立っているものも優先度を下げるために再度 FIFO に挿入されるようにしてある．メモリにコピーされたすべてのページが page out 時に Flash への書き込みが発生しないページの場合 Second chance FIFO と同じになる．

6　まとめ

本論文では，メモリの足りない IOT デバイスでも，不揮発性メモリを仮想的に使えるようなメモリ管理方法を示した．書き込みの多いページをできるだけメモリに保つために．Second chance FIFO を改良して ページアウトの優先順位を変更するアルゴリズムを提案した．それから，Mark and compact の compaction を改良し，書き込みのあったオブジェクトをまとめることで，クリーンなページだけをページアウトすることも実現した．

今回提案した手法の問題がいくつかある．まず，すぐに不要になりうるオブジェクトが non-volatile 領域に置かれるようになっている．また，変更の多いオブジェクトもガベージコレクション時にまとめられるようになっているが，non-volatile 領域に置かれたままになっているのも問題になっている．この両方で必要な要素は，volatile 領域と non-volatile 領域でのオブジェクト間の移動をする機構である．使用頻度や書き込みの有無，オブジェクトの寿命などの要素から適切な領域に配置するような機構を今後導入を検討している．それから，今回のアルゴリズムの評価のための実験を行うことも今後の課題である．ページアウトのアルゴリズムを CFLRU [1] と比較したり，実用的なアプリケーションでの書き込み頻度と寿命を計測したりすることを検討している．

参 考 文 献

[1] Park, S., Jung, D., Kang, J., Kim, J., Lee, J: Cflru:A replacement algorithm for flash memory, *Proceedings ofthe 2006 International Conference on Compilers, Architectureand Synthesis for Embedded Systems, ser. CASES '06*, NewYork, NY, USA, 2006, pp. 234–241.