Vol.2017-ARC-227 No.14
2017/7/26

ハードウェア支援型GCの消費エネルギー評価

早川 慎一郎[1]　河村 慎二[1]　津邑 公暁[1]

概要：モバイル機器の普及に伴い，ガベージコレクション（GC）の性能が及ぼす影響範囲が拡大している．GC は，古くからプロセス全停止に伴うレスポンス低下が問題視されており，これを解決するために，主にアルゴリズム面で改良がなされてきた．しかし，それらの改良ではスループットが犠牲になるなど，GC が抱える根本的な問題解決には至っていない．その解決策として，我々は GC における冗長なマーク処理を抑制するハードウェア支援手法を提案している．この手法が GC 性能に与える影響を調査した結果，GC によるシステムの停止時間を削減でき，レスポンス低下といった GC によるシステムの性能悪化を抑制することを確認している．しかし，追加ハードウェアなどが消費エネルギーに与える影響については評価されてこなかった．本稿ではこの手法の消費エネルギーを評価した上で，その結果を踏まえて，エネルギー消費を低減するため方法について議論する．

1. はじめに

スマートフォンなど，小容量のメモリしか搭載されていないモバイル機器の普及に伴い，**ガベージコレクション（GC: Garbage Collection）**の性能が与える影響範囲が拡大している．GC に関しては古くから，サーバサイド Java 環境などにおいて，全体性能に大きな影響を与えることが知られており，GC 実行時のプロセス全停止によるレスポンス低下が特に問題視されてきた．この問題を解決するため，これまで主にアルゴリズムの改良という観点から多くの研究がなされてきたが，それらはシステムの構成や実行するアプリケーションに合わせた煩雑なチューニングによって GC の発生頻度を抑えるものや，スループットを犠牲にしてシステムのレスポンスを改善するものがほとんどであり，GC が抱える問題の根本的な解決策とは成り得ていない．

これに対し我々はこれまで，多くの実行環境で用いられる代表的な GC アルゴリズムに共通して存在する構成処理要素に着目し，これをハードウェア支援によって高速化することを目指してきた [1][2]．その一つとして，GC における冗長なマーク処理を抑制するハードウェア支援手法 [3] を提案している．この手法は，GC アルゴリズムの構成処理要素の一つであるオブジェクト探索処理に含まれる，冗長なマーク処理を，ハードウェア支援によって省略するものである．この手法により GC によるシステムの停止時間を短縮し，レスポンス低下といった GC に起因するシステムの性能悪化を抑制できることを確認している．しかしながら，追加ハードウェアなどの要因により GC 実行時の消費エネルギーがどの程度影響を受けるかについては評価されてこなかった．消費エネルギーの増大は，駆動時間の減少に伴う可用性の低下につながるため，消費エネルギー量も GC 性能を評価するうえで重要な要素である．

そこで本稿では，この冗長なマーク処理抑制手法が GC 実行時の消費エネルギーに与える影響について調査し，その結果について考察する．また，調査結果に基づき，消費エネルギーに与える影響を抑制するための方法について議論する．

2. ガベージコレクション

GC とは，プログラムが動的に確保したメモリ領域のうち，不要になった領域を自動的に解放する機能である．ここで，プログラム実行時のメモリ領域の概念図を**図 1** に示す．なお，図中の矢印はオブジェクト間の参照関係を表している．

ヒープ領域内に配置されたオブジェクトを指すポインタは，グローバル変数やコールスタック，レジスタなどの，アプリケーションから直接参照可能な領域に格納される．このような領域の集合を**ルート集合**と呼び，これを起点としてポインタを辿ることで，ヒープ領域内のオブジェクトを参照することができる．また，ヒープ領域内に配置されたオブジェクトは，他のオブジェクトへのポインタを保持することもある．そのような他のオブジェクトから参照されているオブジェクトは，ルート集合から複数のオブジェクトを経由することで参照できる．このように，ルート集合

[1] 名古屋工業大学
Nagoya Institute of Technology

© 2017 Information Processing Society of Japan

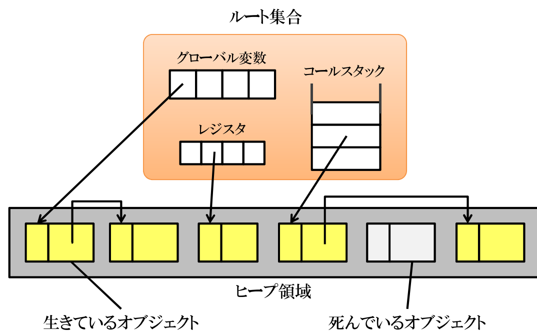

図 1　プログラム実行時のヒープ領域と参照関係の様子

から直接，または間接的に参照可能なオブジェクトを**生きているオブジェクト**，ルート集合から参照不能なオブジェクトを**死んでいるオブジェクト**と呼ぶ．GC では，この死んでいるオブジェクトを破棄し，これに割り当てられていたメモリ領域を再利用できるようにする．

GC の代表的なアルゴリズムの一つに **Mark & Sweep**[4] がある．このアルゴリズムは，ルート集合からポインタを辿り，生きているすべてのオブジェクトにマークを付けるマークフェーズと，マークの付けられなかった死んでいるオブジェクトを回収するスイープフェーズの二つのフェーズから構成されており，これらのフェーズを交互に繰り返しながら動作する．

また，他にも GC の代表的な GC アルゴリズムとして，**Copying**[5]，**Reference Count**[6] があり，現在研究されているすべての GC アルゴリズムは，これら三つのアルゴリズムの組み合わせ，またはその改良であることが知られている [7]．

3.　冗長なマーク処理を抑制する HW 支援手法

この GC を高速化するための方法として，我々は冗長なマーク処理を抑制するハードウェア支援手法を提案している．本章ではこの手法の概要と，必要となるハードウェアの構成，この手法が GC の性能に与える影響について，順に述べる．

3.1　GC の動作解析

我々は GC を高速化するためにまず，GC におけるボトルネックとなっている処理を調査した．なおこの調査は，モバイル端末の実行環境として代表的な **DalvikVM**[8] の Mark & Sweep GC を対象としている．調査の結果，DalvikVM の GC ではヒープ領域内の生きているオブジェクトを順に探索してゆく処理が GC の実行サイクル数の多くを占めていることを確認した．さらに，このオブジェクト探索処理を解析した結果，マーク処理が冗長になされていることを確認した．このマーク処理とは，オブジェクト探索処理で探しだされた生きているオブジェクトがマーク済みであるか否かを確認し，マークされていなければ当該オブジェクトにマークする処理である．DalvikVM の GC では，各オブジェクトがマーク済みであるか否かを管理するために専用のビットマップを用いている．ビットマップを使用したマーク処理では，マーク対象のオブジェクトに直接マークを付す代わりに，ビットマップ中の当該オブジェクトに対応するビットをセットする．そのため，オブジェクトがマーク対象となる度に，マークの有無を確認するため，当該オブジェクトに対応する，ビットマップ中のビット位置を計算する必要がある．しかし，同一オブジェクトに対するビット位置の計算は本来一回で十分であり，不要な計算が繰り返された場合，オブジェクト探索処理におけるオーバヘッドになり得る．

表 1　同一オブジェクトに対する冗長なマーク処理の割合

	最大	平均
AOBench	77.98 %	60.66 %
GCBench	71.58 %	71.58 %
crypto.aes	70.21 %	60.95 %
crypto.signverify	67.79 %	60.96 %
compress	67.29 %	60.90 %
serial	73.10 %	60.88 %

この不要なビット位置計算がどの程度発生しているのかを調べるために，マーク処理の全体回数に占める，同一オブジェクトに対する重複したマーク処理の回数の割合を計測した．DalvikVM 上で実行するベンチマークプログラムには，AOBench[9]，GCBench[10]，および SPECjvm2008[11] から 4 個の，計 6 個のベンチマークプログラムを使用した．**表 1** に示す結果から，全てのベンチマークプログラムにおいて，冗長なマーク処理の割合が 60%を超えており，不要な計算が多く発生していることを確認した．

3.2　手法概要

前節で述べた知見に基づき，この手法では，マーク済みのオブジェクトを記憶するための専用表を追加している．そして，オブジェクトがマーク対象となった際，専用表を参照することで，当該オブジェクトが既にマーク済みであるか否かを判別する．当該オブジェクトが専用表に登録されている場合，ビットマップ中の，当該オブジェクトに対応するビット位置の計算が省略できる．この計算の省略により，マーク処理によるオーバヘッドを抑制し，GC を高速化できる．

ここで，この専用表を用いたマーク処理の省略手順を**図 2** に示す．この図は，ヒープ領域上に存在する 4 個のオブジェクト A から D に対して，順にマークしていく例を示している．まず，ルート集合から参照を辿り，初めて A にマークする際 (i)，専用表にオブジェクト A が登録されていないため，表に A のアドレスを登録する．そしてこの動作を，マーク対象となるすべてのオブジェクトに対して繰り返すことで，やがて専用表には A から D のオブジェ

© 2017 Information Processing Society of Japan

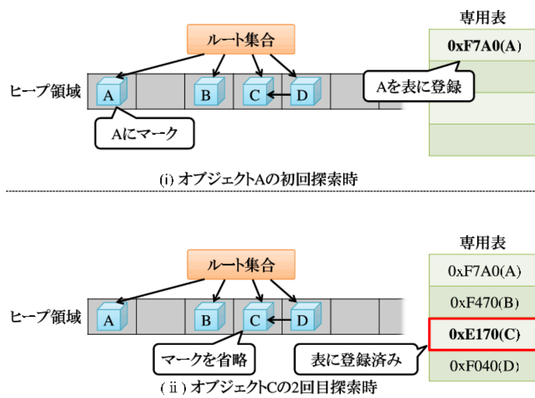

図 2　専用表を用いたオブジェクト探索処理の様子

クトのアドレスが登録される．その後，オブジェクト D の参照を辿って C をマークしようとする際 (ii)，専用表を参照すると C が既に登録されており，C はマーク済みのオブジェクトであることが分かる．そのため，C に対するマーク処理，およびそれに伴うビット位置計算が省略できる．

3.3　エントリの管理方法

この手法では，繰り返しマークされる回数が多いオブジェクトを優先的に管理するために，LRU ベースの追い出しアルゴリズムを用いている．しかし，単純な LRU では，新規オブジェクトが多数連続して登録された場合に，マーク回数の多いオブジェクトが追い出されてしまう可能性がある．

そこでこの表を，複数回マーク対象となったオブジェクトを管理する**一次検索表**と，初めてマーク対象となったオブジェクトおよび一次検索表から追い出されたオブジェクトを管理する**二次検索表**とに階層化している．この階層化により一次検索表では頻繁にマーク対象となるオブジェクトのみを管理でき，新規に登録されるオブジェクトによりこれらが追い出されることを抑制できる．

3.4　追加ハードウェアの構成

次に本節では，各専用表の具体的な構成について述べる．

3.4.1　一次検索表

一次検索表は，頻繁にマーク対象となるオブジェクトに対するマーク処理を省略するために用いられる．オブジェクトへのマークが省略可能か否かの判断に必要な，エントリの検索処理を小さなレイテンシで行うために，一次検索表は高速な連想検索が可能な汎用 CAM（Content Addressable Memory）を用いて実装する．また，この一次検索表では，頻繁にマーク処理の対象となるオブジェクトを優先的に管理するために，LRU ベースのアルゴリズムを用いている．

この一次検索表の各エントリは，オブジェクトのアドレスを保持する Address，および，LRU リストにおいて各オブジェクトの前後に存在するオブジェクトを示す prev,

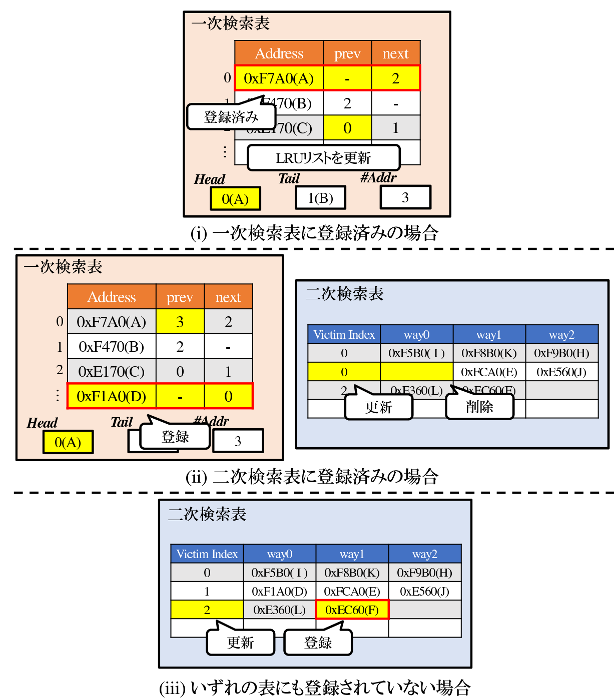

図 3　専用表に対する操作

next の，計 3 つのフィールドから構成される．なお，prev, next の各フィールドは，それぞれ該当するオブジェクトの，一次検索表内におけるインデクス番号を保持している．また，LRU リストの先頭と末尾のオブジェクトのインデクス番号を記録するレジスタ（Head，Tail）と，表に記録されているオブジェクト数を記録するレジスタ（#Addr）とを用意する．

3.4.2　二次検索表

二次検索表は，頻繁にマーク対象となるオブジェクトを判別して一次検索表に登録するため，およびそのオブジェクトが専用表の管理から完全に外れてしまうことを抑制するために用いられており，二次検索表に登録されているオブジェクトはマーク処理の省略対象とならない．そのため，二次検索表に対する操作はマーク処理と並行して行え，そのレイテンシは隠蔽できる．そこで，高速な連想検索が可能ではあるが回路面積や消費電力の面で劣る CAM ではなく，RAM を用いて二次検索表を実装する．なお，ヒット率と検索オーバヘッドのバランスを考慮し，二次検索表ではハッシュを用いたセットアソシアティブ方式を採用する．この二次検索表の各セットは，オブジェクトのアドレスをウェイ数分確保するためのフィールドと，次に上書きするウェイ番号を記憶するためのフィールド（Victim Index）とで構成する．この Victiom Index にはウェイ番号の最大値と等しい値までをカウントできるリングカウンタを用いる．

3.5　専用表に対する操作

これまでに述べた専用表に対して行うマーク処理時の操

© 2017 Information Processing Society of Japan

表 2　シミュレータ構成

Platform	ARM-RealView PBX
Processor	ARMv7
Frequency	2.0 GHz
L1I Cache	32 KB
ways	4 ways
L1D Cache	32 KB
ways	4 ways
L2 Cache	1 MB
ways	8 ways
Memory	256 MB
OS	Linux 2.6.38.8-gem5

作を，**図 3** を用いて示す．まず最初に，マーク対象のオブジェクトを一次検索表から検索する．一次検索表に当該オブジェクトが登録されている場合 (i)，当該オブジェクトに対するマーク処理，およびそれに伴うビット位置計算を省略する．そして，LRU リストにおいて前後関係が変化するエントリの prev，next フィールド，および Head，Tail レジスタをそれぞれ更新することで，LRU リストの先頭に当該オブジェクトを挿入する．

一方，一次検索表に当該オブジェクトが登録されていない場合，当該オブジェクトのアドレスからハッシュ値を計算し，これを用いて二次検索表を検索する．その結果，二次検索表に登録されていた場合 (ii)，まず当該オブジェクトのアドレスを二次検索表から削除し，Victim Index の値を当該オブジェクトのアドレスが記憶されていたウェイ番号に更新する．その後，当該オブジェクトのアドレスを一次検索表に登録する．その際，#Addr を確認して表のエントリに空きがあるか否かを確認する．その結果，エントリに空きがある場合，Addr の値が 0，つまりオブジェクトが登録されていないエントリを検索し，そのエントリに登録する．その後，#Addr の値をインクリメントする．一方，空きが無ければ Tail が指すエントリに登録されているオブジェクトを二次検索表に追い出し，その後当該オブジェクトを上書きする．なお，一次検索表にオブジェクトを登録する際には，(i) と同様に当該オブジェクトを LRU リストの先頭に挿入する．

これに対し，二次検索表にも登録されていなかった場合 (iii)，Victim Index が指すウェイ番号のエントリに当該オブジェクトを登録し，Victim Index の値をインクリメントする．なお，(ii) において一次検索表から追い出されたオブジェクトは，(iii) と同様の手順で二次検索表に登録する．

3.6　ハードウェア支援による性能向上

本節では，これまでに述べた手法の速度性能を評価した結果を示す．評価には，フルシステムシミュレータである gem5[12] を用いた．想定するシステム構成を**表 2** に示す．評価プラットフォームとしては DalvikVM を使用し，ベンチマークプログラムには，AOBench[9]，GCBench[10]，および SPECjvm2008[11] から 4 個の，計 6 個を使用した．なお，追加ハードウェアのサイズによる性能の違いを確認するために，一次検索表，および二次検索表のエントリ数を合計 3 通りの組み合わせで実装し，GC 実行サイクル数を計測した．なお，二次検索表のウェイ数は 4 で固定している．

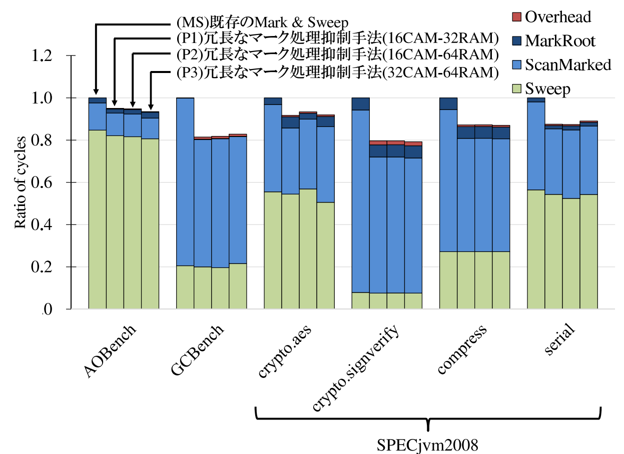

図 4　GC 実行サイクル数

図 4 に，各ベンチマークプログラムの実行サイクル数を示す．4 本のグラフはそれぞれ，

(MS)　既存の Mark & Sweep
(P1)　冗長なマーク処理抑制手法 (16CAM-32RAM)
(P2)　冗長なマーク処理抑制手法 (16CAM-64RAM)
(P3)　冗長なマーク処理抑制手法 (32CAM-64RAM)

において GC の実行に要したサイクル数を示しており，既存の Mark & Sweep (MS) におけるサイクル数を 1 として正規化している．なお，括弧内の'mCAM-nRAM' という表現は，一次検索表が m エントリ，二次検索表が n エントリで構成されていることを示している．図中の凡例はサイクル数の内訳を示しており，Overhead は専用表のアクセスレイテンシをオーバヘッドとして算出したサイクル数，MarkRoot はルート集合から直接参照されているオブジェクトにマークするために要したサイクル数，ScanMarked はマーク済みのオブジェクトが持つポインタを辿り，その参照先のオブジェクトにマークするために要したサイクル数，Sweep はオブジェクトの回収処理に要したサイクル数をそれぞれ示している．なお，本評価で使用した gem5 はフルシステムシミュレータであり，ベンチマークプログラム以外のプログラムも並行して実行されている．そのため，それらのプログラムから受ける影響による性能のばらつきを考慮する必要がある．したがって，本評価では各プログラムにつき試行を 10 回繰り返し，実行サイクル数が最小の値となった結果を，擾乱が最も小さい結果として採用した．

評価結果を見ると，冗長なマーク処理抑制手法では，専

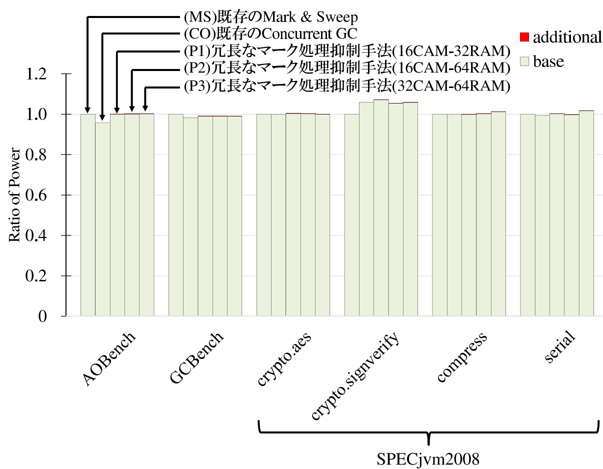

図 5　消費電力

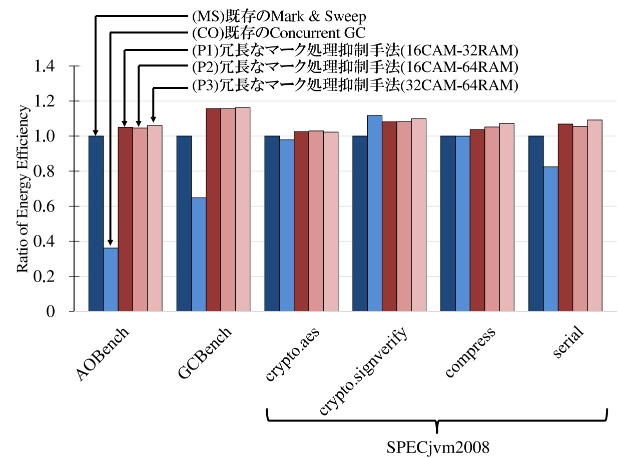

図 6　エネルギー効率

用表のエントリ数に関わらず既存の Mark & Sweep と比較してサイクル数を削減できていることが分かる．そのうちで一番追加ハードウェアのサイズが小さい (P1) では，既存の Mark & Sweep (MS) と比較して平均 13.5%，最大 22.3%サイクル数の削減が確認できた．

以上の結果より，冗長なマーク処理抑制手法は GC の速度性能の向上に有効であることが確認できる．しかしながら，DalvikVM などが動作するモバイル端末はバッテリ駆動が一般的であり，電力リソースが限られているため，速度性能だけでなく，エネルギー性能も GC の性能を議論する上で重要な要素となる．そこで本稿では，これまで述べた手法の消費エネルギーについて調査し，その結果について議論する．

4. 消費エネルギー評価

前章で述べた手法が消費エネルギーに与える影響を調査するために，シミュレーションによる評価を行った．

4.1 評価環境

評価は 3.6 節で示した環境およびベンチマークを用いて行った．また，電力評価には McPAT[13] と CACTI-P[14] とを使用した．本評価では，McPAT，および CACTI-P の出力情報から消費電力を算出し，エネルギー効率を ED 積の逆数として求めた．

4.2 評価結果

まず図 5 に，3.6 節で使用した各ベンチマークプログラムを実行した際の消費電力を示す．4 本のグラフはそれぞれ，3.6 節の (MS)，(P1)，(P2)，および (P3) に，

(CO)　既存の Concurrent GC

を加えた評価対象において，GC を実行した際の消費電力を示しており，(MS) における消費電力を 1 として正規化している．図中の凡例は，消費電力の内訳を示しており，additional が追加ハードウェアで消費された電力，base がベースユニットで消費された電力を示している．なお，(MS) と (CO) は追加ハードウェアが存在しないため，additional の値は 0 である．

評価結果を見ると，冗長なマーク処理抑制手法では，専用表のサイズに関わらず消費電力が (MS) と比較して増大していることが確認できる．(P1)，(P2)，および (P3) の中で最も消費電力の少なかった (P2) では，(MS) と比較して最大で約 5.4%，平均で約 0.85%，消費電力が増大していた．なお，消費電力全体に占める追加ハードウェアによる消費電力の割合は，ベンチマークプログラム全体の平均で，(P1)〜(P3) の全てにおいて約 0.031%と非常に小さかった．この結果から，電力増加分のほとんどは，専用表検索のための専用命令発行などで，ベースユニットにより消費されたものであること，また，一次検索表と二次検索表のサイズの違いによる追加ハードウェアの消費電力の変化は殆ど無いことが分かる．一方，Concurrent GC (CO) を見ると，既存の Mark & Sweep (MS) と比較して消費電力が減少しているベンチマークプログラムがある．これは，Concurrent GC はアプリケーションと並行動作させることで停止時間を短縮する手法であるが，並行動作によるスループットの低下，つまり時間あたりに実行される GC 処理の命令数の減少にその原因があると考えられる．

次に，(MS) の結果で正規化したエネルギー効率の評価結果をを図 6 に示す．これを見ると，(P1)，(P2)，および (P3) はすべてのベンチマークプログラムにおいて (MS) と比較してエネルギー効率が向上していることが確認できる．なお，いくつかのプログラムでは，Concurrent GC においてエネルギー効率が大きく悪化している．これはスループットの悪化により，GC の実行時間が増加したことが要因であると考えられる．

最後に，追加ハードウェアが消費するエネルギー全体に占める一次検索表および二次検索表の消費エネルギーの割合を計測した．その結果，(P1)，(P2)，および (P3) において，すべてのベンチマークプログラムにおいて，97%以上

© 2017 Information Processing Society of Japan

のエネルギーを二次検索表で消費していることが確認できた．この結果より，リーク電力や読み書きに要する電力は RAM よりも CAM の方が大きいにも関わらず，追加ハードウェアにおける消費エネルギーの大半が二次検索表によるものであることが確認できた．以上の結果より，消費エネルギーの削減には二次検索表で消費されるエネルギーを抑制することが有効であると結論づけられる．

4.3 考察

本節では，前節の評価結果に基づき，追加ハードウェアが消費するエネルギーをどのように抑制するかについて検討する．前節で述べたように，追加ハードウェアが消費するエネルギーの大半は二次検索表によるものであり，この消費エネルギーを抑制することが有効である．

二次検索表の改良について議論するために，3.6 節で示したベンチマークプログラムを用いて二次検索表が消費するエネルギーの内訳を計測した．その結果，すべてのベンチマークプログラムにおいて表からの読み出しに要するエネルギーが二次検索表が消費するエネルギー全体の 50%以上を占めていることを確認できた．二次検索表からの読み出しは，マーク対象のオブジェクトが二次検索表に登録されているか否かを確認するために必要な，エントリの検索処理時に発生する．その際，ウェイ数分のエントリに対して読み出しが発生するため，ウェイ数が増えるに従って 1 回の読み出しに要する電力も増加する．以上より，二次検索表の消費エネルギーを抑制する方法として，二次検索表のウェイ数を削減することが考えられる．

ウェイ数の削減が追加ハードウェアの消費エネルギー抑制にどの程度有効であるかを調査するために，サイズを固定した二次検索表のウェイ数と行数を 2 通りの組み合わせで実装し，3.6 節で示した環境およびベンチマークを用いて消費エネルギーをそれぞれ計測した．消費エネルギーは，

(MS) 既存の Mark & Sweep
(W1) 冗長なマーク処理抑制手法 (4way-8line)
(W2) 冗長なマーク処理抑制手法 (2way-16line)

において，GC を実行した際に消費したエネルギーを求めた．まず，消費エネルギー全体に占める追加ハードウェアが消費するエネルギーの割合を求めた．その結果，(W1) で約 0.031%，(W2) が約 0.016%であり，ウェイ数を減らしたことで追加ハードウェアの消費エネルギーを抑制できていることが確認できる．

次に，(MS) の結果で正規化したエネルギー効率の評価結果を図 **7** に示す．これを見ると，GCBench では，(W2) は (W1) と比較してエネルギー効率が約 13.5%向上しており，crypto.signverify でもエネルギー効率が向上している．この二つのベンチマークプログラムは，GC 実行サイクル数全体に占めるマーク処理に要するサイクル数の割合が大きいプログラムである．これらのようにマーク処理が多く発生するプログラムの場合，マーク処理の省略回数が他のプログラムと比較して多くなり，実行時間の短縮率も高くなる．しかしながら，マーク処理の回数が多い場合には，専用表へのアクセス回数も多いため，追加ハードウェアで消費されるエネルギーは他のプログラムよりも多くなる．ここで，ウェイ数を減らした場合には，実行時間の短縮率は殆ど変化せずに消費エネルギーを抑制でき，エネルギー効率が向上したと考えられる．よってこれらと似た性質を持つプログラムでは，ウェイ数を減らすことによりエネルギー効率の向上が期待できる．

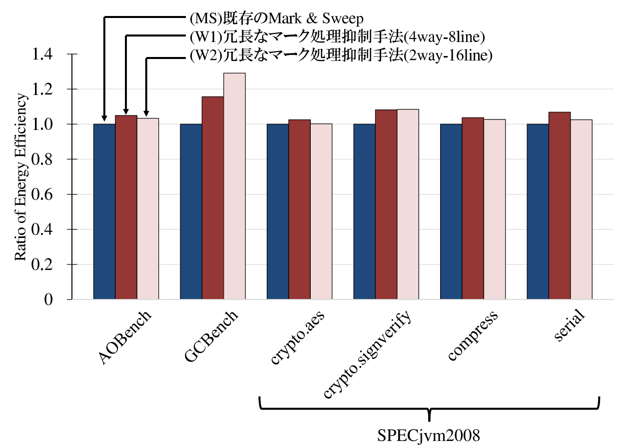

図 **7** RAM の構成を変更した場合におけるエネルギー効率

この結果より，ウェイ数の削減が，追加ハードウェアが消費するエネルギーの抑制に有効であると分かった．なお，ウェイ数の変更により消費エネルギーだけでなく，GC 実行サイクル数も変化するが，既存の Mark & Sweep (MS) と比較して (W1) が平均約 12.9%のサイクル数削減であるのに対し，(W2) は平均約 13.0%の削減となり，サイクル数の削減率が最も大きい crypto.signverify では，(W1) が約 20.4%であるのに対し，(W2) は約 22.2%と，ウェイ数を減らしたことによる速度性能の低下は発生しないことが分かった．

5. おわりに

本稿では，我々が提案している，GC における冗長なマーク処理を抑制するハードウェア支援手法を対象として，その消費エネルギーを評価した．また，各追加ハードウェアのサイズを変更し，それぞれの結果について既存手法との比較を行った．その結果， 既存の Mark & Sweep と比較して消費電力が平均で約 0.85%増大していることを確認した．また，この評価結果から，追加ハードウェアによる消費エネルギー増加の抑制について検討した．

今後の課題として，本稿の評価結果に基づき，追加ハードウェアによる消費エネルギー量の増加を抑えた，新しい GC のハードウェア支援手法を考案することが挙げられる．また，GC 性能の向上のために，ハードウェア支援を前提

© 2017 Information Processing Society of Japan

とした，新たな GC アルゴリズムを考案することも挙げられる．

参考文献

[1] 里見優樹，井手上慶，津邑公暁，松尾啓志：GC におけるポインタ探索高速化のためのハードウェア支援手法，情処研報，Vol. 2013-ARC-207, No. 27, pp. 1–9 (2013).
[2] Ideue, K., Satomi, Y., Tsumura, T. and Matsuo, H.: Hardware-Supported Pointer Detection for common Garbage Collections, *Proc. 1st Int'l Symp. on Computing and Networking (CANDAR'13)*, pp. 134–140 (online), DOI: 10.1109/CANDAR.2013.26 (2013).
[3] Kawamura, S. and Tsumura, T.: Hardware Supported Marking for Common Garbage Collections, *Proc. 4th Int'l Workshop on Computer Systems and Architecture (CSA'16)*, pp. 381–387 (2016).
[4] McCarthy, J.: Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I, *Communications of the ACM*, Vol. 3, pp. 184–195 (1960).
[5] Minsky, M.: A LISP Garbage Collector Algorithm Using Serial Secondary Storage, Technical report, Massachusetts Institute of Technology (1963).
[6] Collins, G. E.: A Method for Overlapping and Erasure of Lists, *Communications of the ACM*, Vol. 3, pp. 655–657 (1960).
[7] 中村成洋，相川　光，竹内郁雄：ガベージコレクションのアルゴリズムと実装，秀和システム (2010).
[8] Bornstein, D.: Dalvik Virtual Machine Internals, Google I/O 2008 (2008).
[9] Fujita, S.: Ambient Occlusion Benchmark, http://code.google.com/p/aobench/.
[10] Boehm, H.: An Artificial Garbage Collection Benchmark, http://hboehm.info/gc/gc_bench.html.
[11] Shiv, K., Chow, K., Wang, Y. and Petrochenko, D.: SPECjvm2008 Performance Characterization, *Proc. SPEC Benchmark Workshop on Computer Performance Evaluation and Benchmarking*, pp. 17–35 (2009).
[12] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D. and Wood, D. A.: The gem5 Simulator, *ACM SIGARCH Computer Architecture News*, Vol. 39, pp. 1–7 (2011).
[13] Li, S., Ahn, J.-H., Strong, R., Brockman, J., Tullsen, D. and Jouppi, N.: McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures, *42nd Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO-42)*, pp. 469–480 (2009).
[14] Li, S., Chen, K., Ahn, J. H., Brockman, J. B. and Jouppi, N. P.: CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques, *ICCAD: International Conference on Computer-Aided Design*, pp. 694–701 (2011).

© 2017 Information Processing Society of Japan