第57回 プログラミング･シンポジウム 2016.1.8-10

並行GC中のCPU周波数抑制による消費電力の削減

片岡 崇史[†1,a)]　鵜川 始陽[†1,b)]

概要：近年のスマートフォンは大きな電力を必要とすることから，無駄な電力の消費を減らして駆動時間を伸ばすことが課題となっている．省電力化の方法のひとつとしてCPUの動作周波数を下げることが挙げられる．動作周波数を下げることはアプリケーションの実行速度に大きな悪影響を与える．そこで，キャッシュミスの起こりやすいガーベジコレクション（以下GC）中の処理のみ動作周波数を下げて実行することで，実行速度に与える影響は小さいまま消費電力を削減する手法が提案されている．しかしこの手法を並行GCに適用する方法は自明ではない．並行GC中に周波数を下げると，GC以外の処理を行っているCPUコアの処理が遅くなるためである．そこで我々はGCを行うCPUコアを固定し，GC中はそのコアのみ動作周波数を下げる方法を提案する．本研究では，この提案手法をAndroidの持つ言語仮想機械であるARTに実装し，その効果を調査した．その結果，エネルギー効率の改善がみられた．

キーワード：DVFS， 消費電力，Android，並行GC，Linux

1. はじめに

近年，スマートフォンは広く普及し，通話だけでなく様々なアプリケーションの実行に使われている．アプリケーションの実行には大きな電力を消費するが，バッテリの容量は限られているため，無駄に消費される電力を削減して，駆動時間を延ばすことが課題になっている．CPUの周波数(以下，周波数)を下げれば消費電力を抑えることができる．一方で，一般には周波数を下げるとプログラムの実行は遅くなる．しかし，メモリの広範囲にアクセスする処理ではキャッシュミスが起きやすく，周波数を下げても実行時間への影響が小さい．キャッシュミスしてメインメモリにアクセスしている間は，周波数に関係なく，CPUは有効な仕事をできずに待たされるからである．このような性質をもつ処理のひとつにガーベジコレクション（以下GC）がある．GCは，実行中のプログラムのヒープ領域に空き領域が少なくなったときに，ヒープ全体から使用されていないメモリ領域を探し出し，それらを解放することで再利用できるようにする．そのためメモリの広範囲へのアクセスが必要となることから，GC中はキャッシュミスが発生しやすい．この性質を利用して，GC中だけ周波数を下げることで，実行速度の低下を抑えつつ，消費電力を削減する手法が提案されている[1]．Androidを搭載したスマートフォンでは，アプリケーションをJavaで記述し，GCを備えた仮想マシンで実行する．既存の研究では，

†1 現在，高知工科大学
Presently with Kochi University Of Technology
a) kataoka@pl.info.kochi-tech.ac.jp
b) ugawa.tomoharu@kochi-tech.ac.jp

GC 中に全てのアプリケーションのスレッド (以下，ミューテータ) を止めて GC を行うように設定した Android に，GC 中だけ周波数を下げる手法を実装し，効果を確認している．近年では，スマートフォンにもマルチコア CPU が搭載されており，それを利用してアプリケーションの停止時間を短縮する並行 GC が主流になっている．しかし，既存研究を並行 GC に適用する方法は自明ではない．並行 GC では，GC が専用のスレッドを持ちミューテータの実行と並行して動作する．GC 中もミューテータが動作しているため，周波数を下げるとミューテータの実行も遅くなってしまう．本発表では，GC スレッドを実行する CPU コアを固定して，そのコアだけ GC 中に周波数を下げることで，GC だけ低い周波数で実行する手法を提案する．GC スレッドにプロセッサアフィニティを設定することで，GC スレッドを実行するコアは固定できる．ミューテータも GC スレッドと同じコアで実行される可能性もあるが，空いているコアがあれば，ミューテータはそのコアで実行するようにスケジュールされるため，あまり問題にならないと考えられる．これによりミューテータの処理があまり遅くなることなく，消費電力を削減できることが期待できる．

本研究の貢献は次の通りである．

- GC を周波数を下げて実行することで消費電力を削減する手法を，並行 GC に適用した．
- Android 5.1.0 の言語仮想マシンである ART に実装し，4 コアの ARM プロセッサで動作する Nexus7 上で性能を評価した．

2. 関連研究

橘田らは，ミューテータの実行を一時的に完全に止めてから GC を行う，ストップザワールド GC (以下 STWGC) を搭載した仮想マシンにおいて，GC 中の周波数を下げたときの消費電力を調べ，この手法は効果があることを示している [1]．本研究は，これを並行 GC に適用する手法を提案するものである．

先行研究の実験対象は Android バージョン 4.2.2 の DalvikVM である．DalvikVM は Android バージョン 5.0 より前において主流の仮想マシンである．GC 中に周波数を下げる手法の基本的な効果を確認するために，ひとつの CPU コアのみ残してそれ以外をスリープさせ，GC には常に STWGC を用いる設定にして実験している．先行研究では，GC の前後に周波数を変更する処理を行い，GC の前には周波数を下げ，GC 後に周波数を元に戻す．これにより GC は周波数が低い状態で実行し，ミューテータの処理は周波数が高い状態で実行する．実験の結果，GC の平均時間が長いプログラムほど消費電力削減の効果が認められている．これは頻繁に GC が行われるプログラムの場合に効果が発揮されることを意味する．

このように，STWGC に GC 中に周波数を下げる手法は有効であることが示されている．一方で近年の Android では，通常は並行 GC が使われていることから，並行 GC への対応が望まれている．

3. 提案手法

GC がキャッシュミスを引き起こしやすい性質を利用して，並行 GC でも消費電力を削減するには，GC スレッドだけ選択的に低い周波数で実行し，ミューテータの実行には影響が出ないようにする必要がある．しかし並行 GC は，GC 中もミューテータが実行されており，全てのコアの周波数を下げるとミューテータの実行まで遅くなってしまう．GC スレッドが実行されるコアのみを周波数を下げれば良いが，GC スレッドがどのコアで実行されるかはわからない．また，ひとつのコアで GC スレッドとミューテータがインタリープして実行されている可能性がある．以上の理由より，GC スレッドのみを周波数を下げて実行することは困難である．

本研究では，GC を行うスレッドを特定の CPU コアに固定し，GC 時はその CPU コアのみ周波数を下げる方法を提案する．具体的には次のようにして実現する．GC を開始するときに，GC スレッドが

(1) 自身のスレッドの実行を特定のコアに固定し，
(2) そのコアのみ周波数を下げる．

GC が終了すると，動作周波数を元に戻し，GC ス

レッドの固定も解除する．

このようにすれば，GC スレッドは低い周波数で実行され，ミューテータへの影響は小さく抑えることができる．OS は，複数のスレッドを実行する際は，各コアに対して負荷が均等になるようにスレッドを割り振る．スレッド数が少ない場合は，ひとつのスレッドに対しひとつのコアが割り当てられるため，GC を行っているコアにミューテータが割り当てられることはない．スレッド数がコア数よりも多い場合は，GC スレッドとミューテータが同じコアに割り当てられ，ミューテータの処理も遅くなる恐れがある．

しかし，近年の CPU はコア数が多く，スレッドがコア数よりも多くなることは少ない．またスレッド数が多い場合であっても，大半のミューテータは GC とは異なるコアで実行されているため，全体としてミューテータに与える影響は小さいと考えられる．

したがって，提案手法では並行 GC であってもミューテータの処理が遅くなることなく，消費電力を削減できることが期待できる．

4. 性能評価

4.1 実装

本実験では，Android バージョン 5.1.0 の仮想マシンである ART を対象とした．Android バージョン 5.0 以降では，それまでの DalvikVM から置き換えられている．我々は ART に組み込まれている GC に対して今回の提案手法を実装し実験した．

動作周波数の制御については，先行研究と同様に Linux カーネルにより提供される API である cpufreq を利用した．GC スレッドを実行する CPU コアの固定には，プロセッサアフィニティの設定を行う API である，`sched_setaffinity` を利用した．GC の開始時と終了時に，これらの API で周波数の変更とスレッドの固定・解除を行った．

我々が今回使用したベンチマークプログラムでは，オブジェクトの生成と破棄を頻繁に行うことから，GC の実行時間が非常に長くなる．これにより，GC 中にメモリが枯渇し，ミューテータの実行が止まることがあった．GC はヒープ内の空きメモリが設定された閾値よりも少なくなったときに開始する．この GC 開始の閾値は，デフォルトでは 128kB から 512kB の間で動的に調整される．我々はこの閾値を 2MB に設定することで，早くから GC を行い，メモリが枯渇するまでに GC が終了するように変更した．

評価対象としては ARM プロセッサを搭載した Nexus7 を使用した．GC が実行される CPU コアの GC 時の周波数は，このプロセッサで設定可能な最低周波数である 384MHz とした．それ以外の周波数は最高周波数の 1512MHz とした．

4.2 評価方法

比較対象として，GC スレッドを特定のコアに固定せずに（以下「コア固定なし」）全てのコアを常に最高周波数で実行する ART の性能を調べた．

これに対し，我々はメモリアクセスが少なく，かつ GC が発生しやすいベンチマークプログラムを作成し，これによる性能評価を行った．このベンチマークプログラムは，4 つのワーカスレッドが暗号化の計算を繰り返し行い，その過程で一時的なバイト列オブジェクトを生成する．

このベンチマークプログラムを連続で 1 時間 30 分実行した後，デバイスのバッテリ残量と処理を行った回数を確認する．この計測結果からエネルギー効率を求める．

4.3 実験結果

3 つの実験について，消費電力の比較を図 1 に，実行速度の比較を図 2 に，バッテリ効率の比較を図 3 に示す．図 1 の縦軸はバッテリ残量の割合で，横軸が時間である．図 2 の縦軸はベンチマークの処理カウンタで，横軸は時間である．図 3 の縦軸はバッテリ残量の割合で，横軸はベンチマークの処理カウンタである．これらのグラフ上の workload は，ベンチマークプログラムによる暗号化の処理が行われた回数である．この回数が多いほど処理性能が高い．

図 1 より，消費電力量が大きく削減されたことが分かる．更に，図 2 より GC 中の周波数を下げることによる実行速度への影響は小さいといえる．

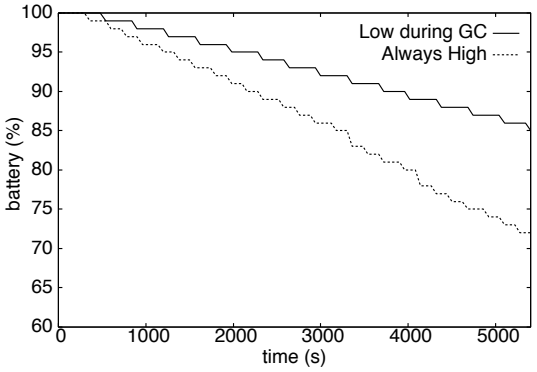

図 **1** 消費電力の比較

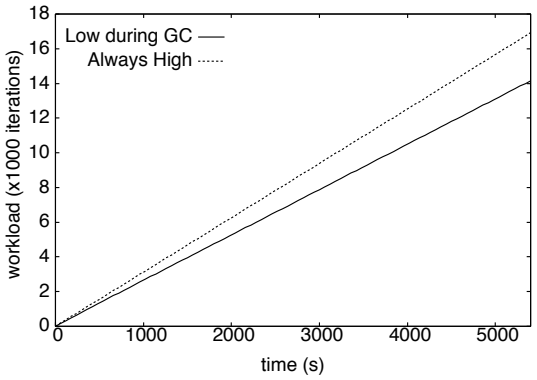

図 **2** 実行速度の比較

また，図 3 より，同じ計算を少ない電力で完了できることが分かり，電力効率が改善されたことが確認できた．

近年のスマートフォンの多くは並行 GC が使われているため，提案手法を適用することにより消費電力量の改善が期待できる．特に今回の実験で使用したベンチマークプログラムのような，オブジェクトの生成と破棄を多く繰り返すアプリケーションでは，提案手法による効果は大きくなると考えられる．

5. まとめ

本稿では，並行 GC において，GC を実行する CPU コアをひとつに固定し，そのコアの周波数を落とすことにより，ミューテータの実行速度を遅くすることなく消費電力の削減をする手法を提案した．また，この提案手法を ART に実装し，効果を調査した．その結果，実行速度の低下は抑えつつ，消費電力を大きく削減することができた．この手法を適用することで，Nexus7 以外のモバイル端末にも同様に消費電力を削減することが期待できる．

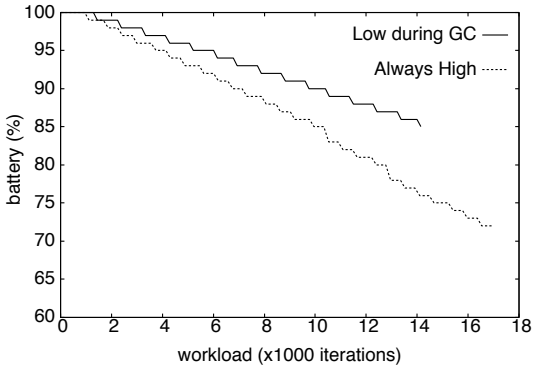

図 **3** バッテリー効率の比較

今後の課題としては，よりアプリケーションに近い処理を行うベンチマークプログラムで性能を評価することが挙げられる．今回使用したベンチマークプログラムは，提案手法の効果をわかりやすくするために，GC が頻繁に発生するものを設定した．実際のアプリケーションの実行では今回のベンチマークプログラムほど GC が発生しないため，これに対して得られる効果を調べる必要がある．また，今回は Nexus7 上の Android バージョン 5.1.0 で実験を行ったが，バッテリー内の電力量の減り方は端末によって異なるため，他の様々な環境でこの手法の効果を調べることが挙げられる．

参考文献

[1] 橘田頼之，鵜川始陽，岩崎英哉: 携帯端末における仮想機械での CPU 周波数抑制による消費電力の削減. 第 55 回プログラミング・シンポジウム予稿集, pp. 75–81 (2014)