Vol.2022-MBL-104 No.25
Vol.2022-UBI-75 No.25
Vol.2022-CDS-35 No.25
Vol.2022-ASD-24 No.25
2022/9/6

Android スマートフォンにおけるフォアグラウンドタスクの動作を考慮した CPU クロックレート制御

熊倉孝太[1]　神山剛[2]　小口正人[3]　山口実靖[1]

概要：スマートフォンにおいて消費電力の低減は重要な課題の一つである．また，CPU は大きな電力を消費する装置の一つであるため，適切な速度の制御が重要となる．CPU による消費電力の削減とユーザエクスペリエンスの改善はトレードオフの関係にある．すなわち，CPU クロックレートを低下させると，消費電力が改善するがユーザエクスペリエンスは悪化する．クロックレートを増加させると逆となる．よって，CPU 資源消費が大きくなる時にはクロックレートを増加させ，低くなるときは低下させることが好ましい．しかし，主要なスマートフォン OS の一つである Android を含め，多くのスマートフォン OS のカーネルでは，未来の CPU 資源消費量の予測は行っていない．CPU 資源消費の増加や減少を観察した後にクロックレートを上げ下げする後追い型のポリシーとなっており，必ずしも適切にクロックレートを制御できていない．本稿では，Android OS に着目し，フォアグラウンドアプリケーションの動作を観察して近い未来の CPU 使用率を予測し，その予測に基づき CPU クロックレートを制御する手法を提案する．そして，性能評価によりその有効性を示す．

キーワード：CPU クロック周波数制御, ART, Java Bytecode, Android アプリケーション，メソッドコール, JVM

1. はじめに

Android スマートフォンは全世界のスマートフォン OS において 70%以上のシェア率を誇るプラットフォームであり[1]，同プラットフォームの性能向上は重要なテーマである．スマートフォンにおける最も大きな課題の一つに，消費電力の多さが挙げられる[2]．スマートフォンを構成する装置の中でも，CPU は多くの消費電力を消費する装置である．また，端末の処理性能と消費電力の間にはトレードオフの関係性が存在する．CPU クロックレートを上げることは端末の処理速度向上(改善)につながるが，消費電力が増加(悪化)にもつながる．また，クロックレートを下げることは端末内の処理速度低下(低下)をもたらすが，スマートフォンの省電力化(改善)につながる．そのため，性能と消費電力の両方を考慮した適切なクロックレートの制御が望ましいと考えられる．

現在 Android OS には Linux カーネルが採用されている．同カーネルは CPU クロックレート制御を行うが，近い将来の CPU 使用率を予測せず，過去の端末の CPU 使用率の変化に追従したクロックレート制御を行う．よって，観察された過去の CPU 使用率と現在や近い未来の CPU 使用率に大きな乖離がある場合などには，適切な CPU 使用率の制御が行えないことがある[3]．また，Android スマートフォンには，端末全体の CPU 使用率がフォアグラウンドで実行されるアプリケーションの処理により支配的に決定されるという特徴がある[3][4]．よって，フォアグラウンドで実行されるアプリケーションが近い将来に使用する CPU 使用率を予測することができれば，端末全体の CPU 使用率をおおむね予測することができると考えられる．加えて，アプリケーションが消費する CPU リソースは，アプリケーション内にて呼び出されるメソッドと大きな関係性があり，アプリケーションのメソッド呼び出しを観察することにより近い未来のそのアプリケーションによる CPU 消費量を推定できると，我々は推測している．

本稿では，まず現在の Android スマートフォンにおけるクロックレート制御手法を紹介し，その課題を述べる．そして，フォアグラウンドアプリケーションのメソッド呼び出しの観察に基づく CPU クロックレート制御手法を紹介する．次に，Android アプリケーション配布サイトに配布されている実アプリケーション 15 個に対して，アプリケーション内の各メソッド呼び出しにおけるその実行時間と CPU 使用率を調査し，実アプリケーションおける呼び出されるメソッドと CPU 消費量の関係の調査結果を述べる．そして，本クロックレート制御を実装する手法を提案し，その実装を用いて動作の検証を行い，その動作の正しさや有用性について考察する．

2. FG アプリケーションのメソッド呼び出しの観察による CPU クロックレート制御

1 章で述べた様に，Linux カーネルは CPU 使用率を観察し，その観察結果に基づき CPU クロックレートの制御を行う．すなわち，未来の CPU 使用率の予測は行わず，過去の観察結果に基づいて制御を行う．よって，過去と現在の CPU 使用率に乖離が生じ，クロックレート制御が不適切になることがある[4]．この問題に対して我々は過去の研究[5][6]にて，Android スマートフォンにおけるフォアグラウンドアプリケーション（FG アプリケーション）のメソッド呼び出しを観察し，この観察結果より近い未来の CPU 使用率

1 工学院大学
Kogakuin University
2 長崎大学
Nagasaki University
3 お茶の水女子大学
Ochanomizu University

ⓒ2022 Information Processing Society of Japan

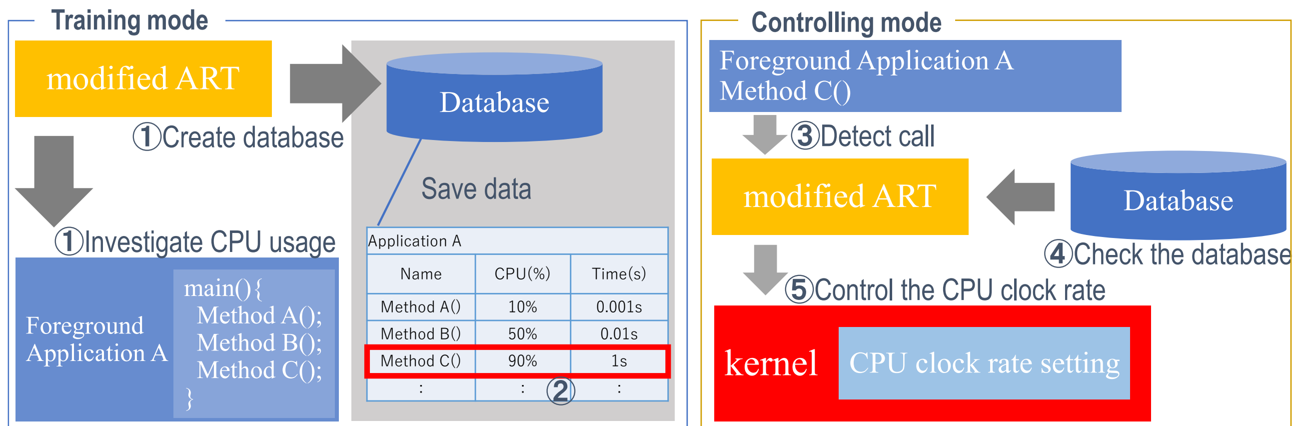

図 1　フォアグラウンドタスクの動作を考慮したクロックレート制御

を推定し，推定結果に基づき CPU クロックレートを制御する手法を提案した．本稿ではこの構想を基に，より最適なクロックレート制御について考察を行う．

この構想の概要を図 1 に示す．提案手法は Training mode と Controlling mode から構成される．Training mode ではフォアグラウンドで実行されるアプリケーションにおけるメソッドごとの CPU 使用率の測定を行う（図中①）．また取得したメソッド情報から，実行時間が長く CPU 使用率の大きいメソッドを特定する（図中②）．Controlling mode では Training mode で取得したデータを基にクロックレートの制御を行う．アプリケーション実行時に呼び出されるメソッドを観察し（図中③）CPU 使用率が高い・低いメソッドであった場合には（図中④），クロックレートを上げる・下げるといった処理を行う（図中⑤）．

3. 関連研究

3.1 Android OS における CPU governor

Android OS にて CPU クロックレートを制御する CPUgovernor について説明する．Android OS には Linux カーネルが採用されており，同カーネルはクロックレートを動的に制御する．governor は端末の CPU 負荷に応じてクロックレートを変更する Linux カーネル内の機能である．クロックレートの変更ポリシーは governor によって違いがある．本章では，本稿の実験端末(Pixel XL 及び Pixel 3a)内に実装されている governor について説明する．

本稿の実験端末内には “schedutil”, “powersave”, “performance”, “userspace” という 4 種類の governor が採用されている．schedutil は端末内で標準(初期設定)起動している governor であり，スケジューラによって CPU クロックレートが選択される．スケジューラが取得した CPU 使用率に基づきクロックレートを変更するといった特徴がある．powersave governor は積極的にクロックレートを下げるように動作し，端末の消費電力を最も抑える governor である．消費電力を低く抑えるが，処理性能は落ちるという特徴がある．performance governor は積極的にクロックレートを向上させる governor であり，処理性能を向上させる代わりに消費電力が大きくなる傾向がある．userspace governor は CPU クロックレートをユーザが設定した値に固定し，クロックレートの動的制御を行わない governor である．本稿では，提案手法と，schedutil, powersave governor との性能を比較し，評価を行う．

3.2 Android アプリケーションの動作観察に関する研究

私たちの過去の研究で，JVM(Java virtual machine)環境を改変することでアプリケーションメソッドの呼び出しを観測可能にし，CPU クロックレートを制御する手法のコンセプトを提案した[5][6]．改変した JVM 環境では，メソッドの呼び出しと終了ごとに端末内の */proc/[pid]/stat* ファイルを読み込むことで，それぞれの時点における累積 CPU 使用時間を測定した．しかし，実用的なアプリケーションにおける各メソッドの CPU 使用率の観察には至っていない．そのため，提案された手法が実際に適用可能かどうかは未確認である．本稿では，提案されていたこのコンセプトと手法を実装して実現している．

先行研究[7][8]では ART(Android Runtime)環境を改変することでアプリケーションライフサイクルの動作の観測を可能にした．しかし提案された観測手法では，アプリケーションのメソッドごとの詳細な実行時間や CPU 使用率の測定は達成されていない．

3.3 Android 端末の CPU クロック周波数制御についての研究

文献[4]にて，Android スマートフォンに採用されている governor の動作に着目し，クロックレートをより最適に制御した研究が行われている．標準 governor に比べ，クロックレートを上げる際には小さく，クロックレートを下げる際には大きく変動させるように governor を改変し，端末性能をほとんど落とさずに消費電力を大きく削減することに

ⓒ2022 Information Processing Society of Japan

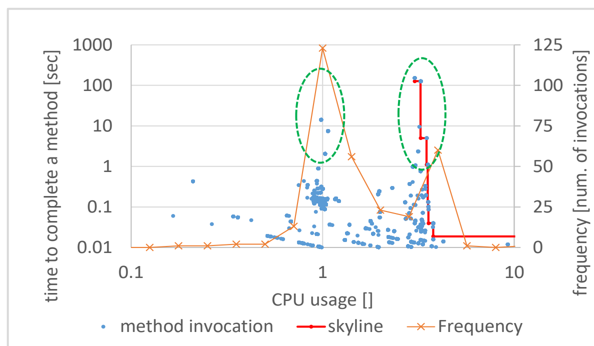

図 2　各メソッドの CPU 使用率と実行時間とスカイライン（アプリケーション O)

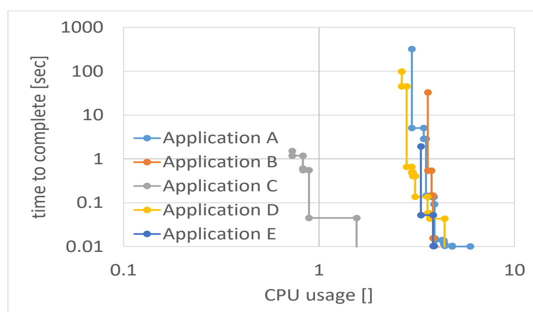

図 3　各メソッドの CPU 使用率と実行時間のスカイライン（アプリケーション A～E)

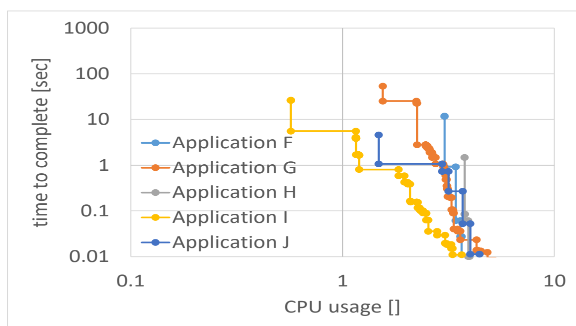

図 4　各メソッドの CPU 使用率と実行時間のスカイライン（アプリケーション F～J)

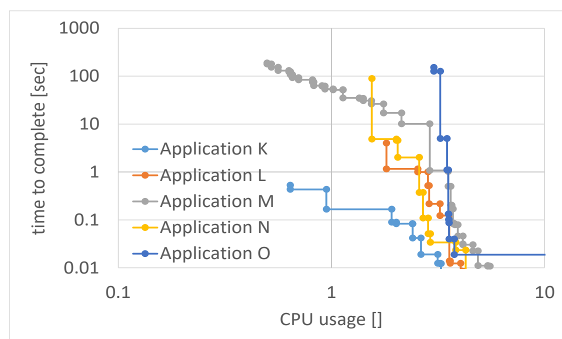

図 5　各メソッドの CPU 使用率と実行時間のスカイライン（アプリケーション K～O)

成功している．

また文献[9]にて，アプリケーションから取得したフィードバック情報を基に最適なクロックレート制御を行うことで，必要とされるパフォーマンスを満たしつつ消費電力を抑える手法が提案されている．

3.4 Linux カーネル内 CPU クロック周波数の最適化に関する研究

Linux 環境において動的にクロックレートを制御するための研究が存在する．Rao らは，オフラインプロファイリングとオンライン制御の 2 段階から構成されるクロックレートの制御手法を提案している[10]．本稿における提案手法も同様に 2 段階の制御理論から構成される．

また Bao らは, CPU クロックレート選択に対するコンパイル時アプローチを提案している[11]．そして，60 のベンチマークと 5 つのマルチコア CPU を用いた評価により，Linux が提供する governor である“powersave”を上回りつつ性能の向上を果たしている．しかしこれらの先行研究では，アプリケーションの動作を考慮したクロックレートの制御や，メソッド呼び出しに基づく制御については考察されていない．

4. 実アプリケーションにおけるメソッド呼び出しごとの実行時間と CPU 使用率

文献[3]の提案手法は，アプリケーションにて CPU クロックレートの制御（レートの上昇と低下）をするべき特徴的なメソッド呼び出しが存在することを前提としている．具体的には, CPU 使用率が高く実行時間が短くないと予想されるメソッドの呼び出しが存在し，そのメソッドの開始時に CPU クロックレートを上昇させ，その終了時に低下させることを想定している．クロックレートの変更に時間を要するため，頻繁なクロックレートの変更は好ましくないと予想され，そのメソッド呼び出しは CPU 使用率が高いのみでなく，実行時間が短くないことも必要となる．本章では，実アプリケーションにおけるメソッド呼び出しごとの CPU 使用率と実行時間を計測し，これらの前提（CPU 使用率が高くかく，実行時間が短くない）を満たすメソッド呼び出しの存在について調査を行う．これらの内容は前章の Training mode に該当する．

我々の実験環境に実アプリケーションをインストールしてアプリケーションを実行し，メソッドごとの CPU 使用率と実行時間を取得，分析を行った．調査対象のアプリケーションは, 2022 年 4 月 28 日時点にて Google Play Store における無料アプリケーションのダウンロード数ランキング上位 15 位のアプリケーションとした．これらをアプリケーション A～O と呼ぶ．実験端末は Pixel XL であり，CPU コア数は 4 である．

アプリケーションごとに全てのメソッド呼び出しが完了するまでの時間と，メソッドごと CPU 使用率を計測し

Vol.2022-MBL-104 No.25
Vol.2022-UBI-75 No.25
Vol.2022-CDS-35 No.25
Vol.2022-ASD-24 No.25
2022/9/6

図 6　各メソッドの CPU 使用率と実行時間 (アプリケーション A～N)

た．CPU 使用率は使用した CPU 時間を経過した実時間で割った比率である．各 CPU コアの CPU 使用率は最大で 1.0（100%）であるため，4 つの CPU コアでは，最大 4.0（400%）の CPU 使用率となる．図 2 はメソッド完了までの平均時間が最も長かったアプリケーション O のメソッドごとの実行時間と CPU 使用率の関係を示したものである．ただし，グラフにはメソッドの実行時間が 10ms 以上のデータのみを表示している．青い各点は実行されたメソッドを意味し，横軸の値が CPU 使用率を，縦軸の値が実行時間を意味する．赤線は全呼び出しのスカイライン[12]を意味する．オレンジ色の線は CPU 使用率でグループ分けしたメソッド呼び出しの頻度を示している．

図 2 から，一部のメソッド呼び出においては，メソッドの実行時間，その間における CPU 使用率ともに大きくなっていることが分かる．このことは，これらのメソッドを呼び出す前に CPU クロックレートを向上させることが適切であることを示唆していると言える．オレンジ色の線により表されている頻度に着目すると，CPU 使用率が 1.0 と 4.0 付近にあるメソッド呼び出しが多いことが分かる．このことから，CPU コアを 1 つ最大限に活用する(その CPU の使

ⓒ2022 Information Processing Society of Japan

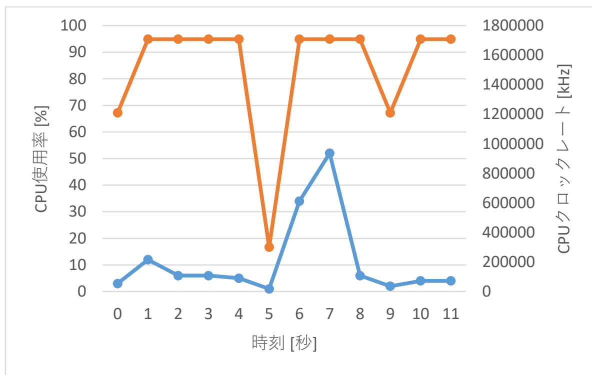

図 7　CPU 使用率と CPU クロックレートの推移 (提案手法)

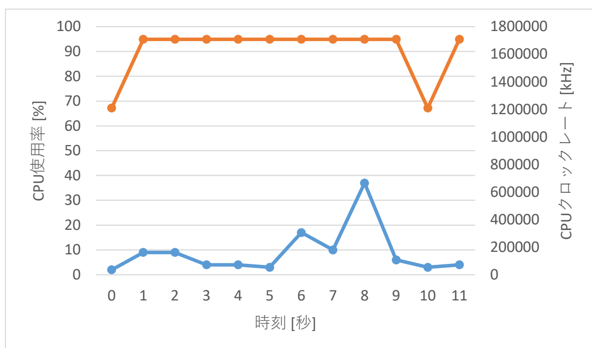

図 8　CPU 使用率と CPU クロックレートの推移 (schedutil governor)

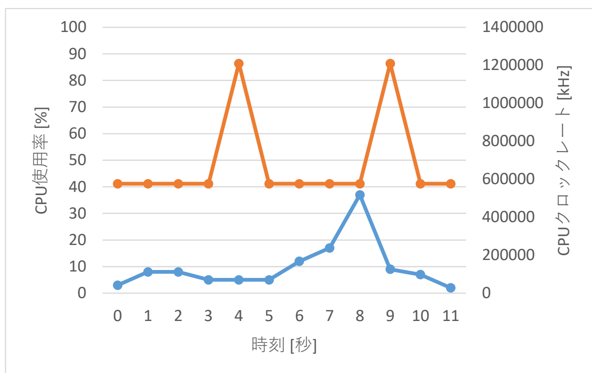

図 9　CPU 使用率と CPU クロックレートの推移 (powersave governor)

用率が 100%になるまで活用する)メソッド呼び出しや，CPU コア 4 つを最大限に活用するメソッドが多いことが読み取れる．緑色の丸で示した実行時間が 1 秒以上のメソッド呼び出しに注目すると，CPU 使用率が 100%もしくは 400%近辺になっていることが分かる．つまり，この実験結果より，これらの比較的長い時間を消費するメソッドは多くの，CPU リソースを多く消費していた(1 コアあるは 4 コアの CPU を最大限に消費していた)ことが判明した．

図 3～図 5 は，計測した 15 個のアプリケーションにおけるメソッド完了までの時間と CPU 使用率のスカイラインを示したものである.この結果から，多くのアプリケーションの実行にはメソッド完了までの時間が長く，CPU 使用率の高いメソッド呼び出しが存在していることが分かる．このことは，本稿で注目しているフォアグラウンドアプリケーションのメソッド呼び出しの観察による動的な CPU クロックレート制御[4]が，多くの Android アプリケーションに対し適切に機能することを意味していると期待できる．一部のアプリケーションでは実行に 10 秒以上要するメソッドがあり，それらのメソッド呼び出しは CPU 資源を極端に多く消費していた．このような場合には特に本手法による動的制御が効果的であると期待される．

図 6 に，アプリケーション A からアプリケーション N におけるメソッドごとの実行時間と CPU 使用率の関係を示す．

5.　実装手法

本章にて，アプリケーションのメソッド呼び出しごとの実行時間と CPU 使用率の観察に基づく CPU クロックレートの制御手法を Android OS に実装する簡易な方法について述べる．これらの内容は 2 章の Training mode に該当する．

ART においてインタプリタの実行対象のアプリケーションのメソッド呼び出し(invocation)と復帰(return)の観察は，文献[6]にて述べられている手法により実現できる．すなわち，インタプリタのメソッドのコールスタックにおける push と pop を観察することにより実現できる．また，push 時と pop 時の時刻と累積 CPU 時間を記録して，それらの push 時と pop 時の差を計算することにより，そのメソッド呼び出しに要した時間と, 要した CPU 時間を取得することができ，これらの比より CPU 使用率を計算することができる．

Training mode にてメソッド呼び出しを観測し，実行時間が長く CPU 使用率が大きいと考えられるメソッド(このメソッドは Controlling mode にて特定される)が呼び出された際に CPU クロックレートを最大にし，メソッド呼び出しが終了した際にはクロックレートを最低にすることにより，解像度が低いが簡易な CPU クロックレートの制御を実現できる．CPU クロックレートは，アプリケーションの権限(一般ユーザ権限)にて動作する ART プロセスにて変更する必要がある．これは，一般ユーザ権限で読み込みが可能であり，読み込みを行うとカーネル内にて CPU クロックレートの変更(最高化や最低化)が行われる /proc ファイルを Linux カーネルに追加することにより実現できる．そして，ART にて対象メソッドの呼び出しが観察された際に当該 /proc ファイルの読み込みを ART が行うことにより CPU クロックレートの動的制御を実現できる．

ⓒ2022 Information Processing Society of Japan

Vol.2022-MBL-104 No.25
Vol.2022-UBI-75 No.25
Vol.2022-CDS-35 No.25
Vol.2022-ASD-24 No.25
2022/9/6

6. 動作検証

提案した動的 CPU クロックレート制御手法を，5 章の方法により試作した．本章にて，その動作の検証と有効性についての考察を行う．本試作実装では，Training mode は Pixel XL と Android 9.0.0 にて動作させ，Controlling mode は Pixel 3a と Android 12.0.1 で動作させた．ただし，これら OS の ART および Linux カーネルは，5 章で示した修正が行われているものである．検証のために評価用アプリケーションを実装し，そのアプリケーションにより動作の確認と検証を行った．自作の評価用アプリケーションは，GUI (Graphical User Interface)上に 2 つのボタンが配置されており，それぞれのボタンをタップすると自作のメソッド `method0()`と `method1()`が呼ばれる．前者のメソッドは多くの CPU 資源を消費し，後者のメソッドは CPU 資源をほぼ消費しない．両メソッドともメソッドの実行時間は短くなく，前者の実行時間は約 1 秒程度であり，CPU クロックレートに依存する．後者の実行時間は CPU クロックレートに依存せず 1 秒である．評価実験では `method0()`と `method1()`を交互に呼び出した．アプリケーションの動作の統一するため，プログラムにより指定時間にスクリーンタップイベントを発生させてアプリケーションの動きを制御し，そのときの OS の動作を観察した．

提案手法，schedutil governor，powersave gover にて動作検証を行ったときの，CPU 使用率と，CPU クロックレートの推移を図 7，8，9 に示す．図より，提案手法ではメソッド終了時に CPU クロックレートの最低への低下を行ったことを確認することができる．また，CPU 使用率が上昇すると同時に CPU クロックレートの上昇を行ったことを確認できる．これは提案手法が，CPU 消費が大きいメソッドの呼び出しを検出し，その開始時に CPU クロックレートを上昇させた結果であると予想される．一方で schedutil governor では，CPU 使用率が低い状態での積極的な CPU クロックレートの低下は確認できない．また，CPU 使用率の低下に遅れて CPU クロックレートの低減を行っていることが確認できる．powersave governor においては，ほとんどの時間を非常に低い CPU クロックレートとしており，消費電力の低減と，低い性能の利点と欠点が明確に確認できる結果となっていることが分かる．

以上の結果より，Training mode で学習したメソッド呼び出しを Controlling mode で検出し，検出に伴い CPU クロックレートと制御する手法が正しく実現されていることが確認できる．また，その制御は既存の schedutil governor より適切である可能性が示されているが，消費電力や性能(アプリケーション実行の時間)の定量的な計測による評価が必要であると考えられる．

7. 考察

本稿では，CPU クロックレートの制御手法について考察を行った．既存の CPU governor との比較について考察を行う．Linux にて初期設定で選択されている schedutil governor は，クロックレートが高い状態にあることが多いことが分かっている[4]．また，6 章の評価結果においても，多くの時間において CPU クロックレートが高い状態にあることが分かる．よって，より積極的に CPU クロックレートを下げる CPU governor と本手法の併用がより効果的になることが期待できる．より積極的に CPU クロックレートを下げる CPU governor の例としては，powersave governor が考えられるが，当該 governor は過剰に下げると予想されるため，初期設定の governor を積極的に改変したもの[4]が適切であると考えられる．

また，実行時間が短くないメソッド呼び出しを CPU クロックレートの制御の対象とする必要があると考えているが，その定量的な考察も重要であると考えられる．このためには，CPU クロックレートの変更に要する時間とそれに消費される電力の定量的な調査が重要であると考えられる．1 回の CPU クロックレートの変更に消費される電力量が定量的に分かれば，CPU クロックレートの低減によりえら得る消費電力量と，その制御により消費される電力量を比較し，break-even-time を特定でき，より効果的な制御可能になると期待される．

8. おわりに

本稿では，Android OS および Linux カーネルの CPU クロックレート制御に着目し，フォアグラウンドアプリケーションのメソッド呼び出しの観察による近い未来の端末の CPU 使用率の推定と，その推定値による CPU クロックレートの制御手法について考察した．具体的には，実アプリケーションにおけるメソッド呼び出し時の実行時間や CPU 使用率を調査し，CPU クロックレートの上昇や低下を行うべきメソッド呼び出しが存在していることを確認した．そして，本 CPU クロックレート制御手法の実装方法を示し，その実装例の動作の検証を行い有効性について考察を行った．

今後は，端末の消費電力やアプリケーションの性能の評価，CPU クロックレートの最大と最小以外も含む解像度の高い制御の実現，実アプリケーションでの評価をやっていく予定である．

謝辞 本研究は JSPS 科研費 21K11854, 21K11874 の助成を受けたものである．

参考文献

[1] Mobile Operating System Market Share Worldwide, July 2021 - July 2022. https://gs.statcounter.com/os-market-share/mobile/worldwide.

[2] P. K. D. Pramanik et al., "Power Consumption Analysis, Measurement, Management, and Issues: A State-of-the-Art Review of Smartphone Battery and Energy Usage," in IEEE Access, vol. 7, pp. 182113-182172, 2019, doi: 10.1109/ACCESS.2019.2958684.

[3] 関屋拓司，栗原駿，福田翔貴，濱中真太郎，小口正人，山口実靖, "アプリケーションの動作と消費電力を考慮したスマート

フォン CPU クロック周波数制御", 情報処理学会 第 79 回全国大会講演論文集, 2017 巻, 1 号, pp. 127-128, 2017.

[4] Y. Sato, M. Oguchi and S. Yamaguchi, "Mobile Application Aware Smartphone CPU Clock Frequency Optimization," in 2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW), Takayama, Japan, 2018 pp. 564-566. doi: 10.1109/CANDARW.2018.00112

[5] K. Kumakura, A. Sonoyama, T. Kamiyama, M. Oguchi and S. Yamaguchi, "Observation of Method Invocation in Application Runtime in Android for CPU Clock Rate Adjustment," 2021 Ninth International Symposium on Computing and Networking Workshops (CANDARW), 2021, pp. 481-483, doi: 10.1109/CANDARW53999.2021.00090.

[6] K. Kumakura, T. Kamiyama, M. Oguchi, S. Yamaguchi, "Stack-based Method Invocation and Return Monitoring in ART for CPU Clock Rate Adjustment," 2022 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), 2022.

[7] K. Nagata and S. Yamaguchi, "An Android application launch analyzing system," 2012 8th International Conference on Computing Technology and Information Management (NCM and ICNIT), 2012, pp. 76-81.

[8] K. Nishinaka, A. Sonoyama, T. Kamiyama, A. Fukuda, M. Oguchi and S. Yamaguchi, "Monitoring System for Optimization based on Analyzing Android Application Launching Behavior," 2020 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan), 2020, pp. 1-2, doi: 10.1109/ICCE-Taiwan49838.2020.9258290.

[9] K. Nagata, S. Yamaguchi and H. Ogawa, "A Power Saving Method with Consideration of Performance in Android Terminals," 2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing (IEEE ATC 2012), Fukuoka, 2012, pp. 578-585. doi: 10.1109/UIC-ATC.2012.133

[10] K. Rao, J. Wang, S. Yalamanchili, Y. Wardi and H. Ye, "Application-Specific Performance-Aware Energy Optimization on Android Mobile Devices," 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017, pp. 169-180, doi: 10.1109/HPCA.2017.32.

[11] W. Bao, C. Hong, S. Chunduri, S. Krishnamoorthy, L. N. Pouchet, F Rastello, and P. Sadayappan. 2016. Static and Dynamic Frequency Scaling on Multicore CPUs. ACM Trans. Archit. Code Optim. 13, 4, Article 51 (December 2016), 26 pages. DOI: https://doi.org/10.1145/3011017

[12] S. Borzsony, D. Kossmann and K. Stocker, "The Skyline operator," Proceedings 17th International Conference on Data Engineering, 2001, pp. 421-430, doi: 10.1109/ICDE.2001.914855.

ⓒ2022 Information Processing Society of Japan