スマートフォンにおける消費エネルギー可視化のための予測モデル

2010SE162 小川 詩織　2010SE175 大沢 沙希
指導教員：横山 哲郎

1　はじめに

近年，スマートフォンの利用者が急増していく傾向にある．世界における携帯電話の販売台数に占めるスマートフォンの比率は，2011 年には 26.6%だったものが 2015 年には 51.8%に増えると予測されている[1, p.161]．利用者が携帯電話からスマートフォンに乗り換えた理由の上位には，画面が大きくパーソナルコンピュータと同じようにウェブページを閲覧できたり，豊富なアプリケーションを自由に追加できたりすることが挙げられている[1, p.211]．このような利便性の高さから，より多くの利用者からスマートフォンが使われるようになると予測される．しかし，便利である分スマートフォンを使用する時間が長くなり，バッテリ持ちに不満を感じることがある．そこで，スマートフォンのバッテリ持ちを向上させることが重要であるといえる．

バッテリ持ちを向上させる方法のひとつに，消費電力を削減する方法がある．消費電力を削減するためには，まず，システムの稼動時に消費電力を可視化する技術が必要である．可視化をすることで，電力消費の原因となる機能やソフトウェアを明らかにすることができ，省電力ソフトウェア開発の効率化につながる．

文献[2]では，消費電力の見積もり手法や，消費電力モデルの生成手法の提案がされている．これらの研究では，シングルコアである Nokia 社の無線通信端末 N810 の 1 台を対象に実験が行われていた．そこでわれわれは，N810 以外の端末で再現実験を行い，提案手法の有効性を確認することを目指す．この手法が提案された当時はシングルコアの CPU を搭載した端末がほとんどであったが，本研究を開始した時点ではデュアルコアが当たり前になってきている[3]．そこで，われわれはデュアルコアの端末を対象として，消費電力の見積もりや消費電力モデルの生成を行う．具体的には，CPU 負荷率と消費電力の分布を実測により求め，CPU 負荷率から消費電力を見積もることのできるモデルを単回帰分析によって作成する．消費電力に対して線形で，かつ大きな影響を与えることが予測できるため，モデルパラメータには，CPU 負荷率を使用する．

ソフトウェアによる省電力化の方法は，動的な消費電力を抑えるための手段である DVFS と連携した手法がよく知られている．そこで，CPU 負荷量とエネルギー効率の関連を求め，ondemand governor を用いた理想的な応需戦略のモデルを作成する．

2　関連研究

2.1　消費電力の見積もり手法と消費電力モデルの生成手法

文献[2]では，消費電力の見積もり手法と消費電力モデルの生成手法が提案されている．また，Nokia 社の無線通信端末 N810（シングルコア）を用いて，消費電力モデルの生成と消費電力見積もり手法を適用した事例を示している．

消費電力モデルを作成するには，まず，複数のモデルパラメータに対応したテストベンチを用いて，電力測定器で消費電力値を測定する．次に，重回帰分析を用いて，パラメータと測定した消費電力値からモデルを作成する．モデルには，線形式

$$P_{estimated} = C_0 + \sum_{i=1}^{N} C_i \cdot P_i \qquad (1)$$

を用いた手法が提案されている．$P_{estimated}$ は消費電力の見積もり値，P_i は線形モデルのパラメータ，C_i は各パラメータの係数，C_0 は定数項，N はパラメータの数を表わしている．モデルが作成されると，パラメータ値を用いて端末の消費電力を見積もることができる．

一部のベンチマークを除き，誤差が 5%未満と非常に精度が良いと示している．

2.2　DVFS

動的に消費電力を制御する方法の中でも，周波数を制御しながら適切な電源電圧制御を行うことを DVFS（Dynamic Voltage and Frequency Scaling）という[4]．電圧と周波数を下げ，動作を遅くすることで消費電力を削減することができる．しかし，単に遅くすることが最適というわけではなく，設計の余地がある．設計を行うための情報を提供するデータが得られたことを4.2節で述べる．

2.3　ondemand governor

ondemand governor は多くの端末で用いられており，一定の負荷がかかると周波数を最も大きい値に上げ，負荷が下がると安定するまで段階的に下げる[5]．ondemand governor の閾値が設定可能である．仕事量とエネルギー効率の関連を求めることで，消費電力量を最適化する閾値の議論ができる．

3　問題解決方法の選定

3.1　実験機器

実験で使用したスマートフォン（以下「端末」）は，Samsung 社の Galaxy Nexus である．CPU は Texas Instruments OMAP4460 1.2GHz のデュアルコアであり，CPU クロック周波数は 350–1200MHz までの可変である．

端末への電力供給はバッテリではなく電源装置 PMC18-5A で行い，電力測定器 WT1600 によってその電圧・電流を測定した．測定電力の確度は読み値誤差 0.1%，測定レンジ誤差 0.2%であった．データ更新レート

は 1s に設定した.

3.2 条件

端末でアプリケーションを実行する際に, 端末の状態を一定にするため, 下記のような条件下で実験を行った.

(a) Bluetooth, Wi-Fi, 3G 通信, セキュリティソフトなど全てオフ
(b) アプリケーションの実行中にディスプレイを常に点灯(スリープ状態にしない)
(c) CPU のクロック周波数を固定(SetCPU アプリケーションを用いて設定)

3.3 実験方法

端末内にある SetCPU アプリケーションを用いて, クロック周波数を 350MHz, 700MHz, 920MHz, 1200MHz にそれぞれ固定し, CPU 負荷をかけて電力測定を行った.

実測値から, アプリケーション無実行時の消費電力の値, CPU 負荷率が 0%, 25%, 50%, 75%, 100%のときの消費電力の値を求める. ここで, アプリケーション無実行時とは, アプリケーションを起動し, sleep 周期と count の値を入力せずに消費電力を測定したものである. また, CPU 負荷率が 0%とは, CPU 負荷率がほぼ 0%となるような sleep 周期と count の値を入力し, アプリケーションを実行している間の消費電力を測定したものである. アプリケーションの影響がない状態のモデルを作成するために, CPU 負荷率が 0%, 25%, 50%, 75%, 100%のときの消費電力の値から無実行時の消費電力の値の平均値を減算する. 結果から, 2.1 節の式(1)を $N = 1$ に特殊化した線形式を用いて, CPU 負荷率から消費電力を算出できるモデルを作成する. 説明変数 P_1 には CPU 負荷率を用いる. CPU 負荷率と消費電力値を実験において計測して, 単回帰分析により係数を推定する.

作成したモデルと同じ条件下の消費電力を測定器で測定する. その結果とモデルを比較し, 精度を求める.

3.4 CPU 負荷率を計測するアプリケーション

3.4.1 CPU 負荷率の取得方法について

CPU 負荷率とは, CPU 稼働時間を実時間で割った値である. CPU 稼働時間は, テストベンチ実行前後の CPU の起動時間の値の差で, 実時間は, 実行前後の現在のシステム時間の値の差である.

CPU 起動時間と現在のシステム時間の値をそれぞれ計測する方法に, **/proc/stat** に書かれた情報を用いるものがある. Linux の**/proc/stat** には, システムの状態を表わす数値が格納されており, 1行目には, 図1のような CPU の状態を表わす数値が格納されている.

```
cpu 130120 1621 38979 3053995 ・・・
```

図 1 **/proc/stat** の 1 行目

図 1 に示した値は, 左からそれぞれ, user(ユーザーモードで消費した時間), nice(低い優先度のユーザーモードで消費した時間), system(システムモードで消費した時間), idle(タスク待ちで消費した時間)という意味がある. CPU 起動時間は user, nice, system の和, idle は CPU のアイドリング時間である. また, 現在のシステム時間は CPU 起動時間と CPU のアイドリング時間の和である. この方法では, **/proc/stat** で取得した値から CPU 起動時間とアイドリング時間, 現在のシステム時間をずれが生じることなく取得できる. また, 文献[2]でも, **/proc/stat** が用いられているため, 今回の実験と条件をそろえることができる. したがって, **/proc/stat** を使用し計測する方法で CPU 負荷率を求めることにした.

3.4.2 デュアルコア

N810 はシングルコアであるのに対し, Galaxy Nexus はデュアルコアであった. 片方のコアに負荷をかけるのではなく, 2 つのコアを同時に動かすことで, より現実に近い状態で計測を行うことができる. そこで Thread をたて, CPU1, CPU2 を同時に稼動させる. 仮に, CPU1 の処理が先に終わっても, CPU2 の処理が終わるまで待つようにし, CPU2 の処理の途中で次の処理を行わないようにした.

3.4.3 テストベンチ

今回作成したアプリケーションは, sleep 周期と count を入力して, CPU 負荷率を出力するものである. アプリケーション内のテストベンチ(図 2)の数値演算によって負荷をかける. ここで, count とは図 2 内の for 文を何回まわすかというもので, sleep 周期とは何回に 1 回処理を休むかというものである. 図 2 内の関数 **Thread.sleep()**とは特定の時間間隔(long 型 ミリ秒)休む関数である. 今回は, 10ms 休む設定にしている.

```
for (int j=0; j<count; j++) {
        数値演算
        //sleep 周期
        if (j%sleepcycle==0) {
                Thread.sleep(10);
        }
}
```

図 2 CPU 負荷率測定用テストベンチ

3.5 周波数ごとのエネルギー効率の逆数の計算方法

エネルギー効率の逆数は, CPU 負荷率とクロック周波数のかけた値である MIPS (Mega Instructions Per Second:CPU 負荷量)と, 消費電力を CPU 負荷量で割った値である W / MIPS (Inverse Energy Efficiency :エネルギー効率の逆数)の値を求めることで算出できる.

4 実験結果

4.1 CPU 負荷率と消費電力のグラフとその精度

クロック周波数が 350MHz, 700MHz, 920MHz, 1200MHz のそれぞれに対して, 無実行時の消費電力と, CPU 負荷率が 0%, 25%, 50%, 75%, 100%のときの消費電力の測定を行った. それぞれの値に対し 20 回ずつ測定し, その散布図を作成し, モデル式(線形近似の式)を

求めた. ただし, 縦軸の消費電力は, CPU 負荷率が 0%, 25%, 50%, 75%, 100%のときの消費電力の値から無実行時の消費電力の値の平均値を減算したもので, 横軸は CPU 負荷量である.

この結果から, 測定値と見積もり値との平均誤差, 最大誤差を求めた. ここで, x は CPU 負荷率(%), y は消費電力(W)とする. 350MHz の式 $y = 0.23x + 0.0154$ との誤差は, 平均 0.01W, 最大 0.03W, 700MHz の式 $y = 0.6x + 0.0314$ との誤差は, 平均 0.02W, 最大 0.07W, 920MHz の式 $y = 0.95x + 0.0623$ との誤差は, 平均 0.04W, 最大 0.12W, 1200MHz の式 $y = 1.65x + 0.0748$ との誤差は, 平均 0.11W, 最大 0.58W であった.

線形近似の式を 1 つにまとめたものを図 3 に示す.

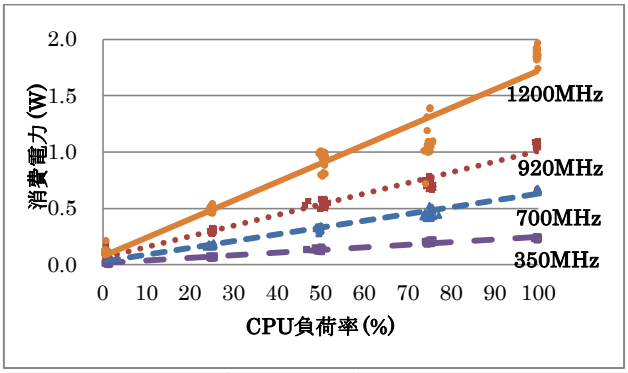

図 3 CPU 負荷率と消費電力のグラフ

4.2 周波数ごとのエネルギー効率の逆数のグラフ

3.5 節から, CPU 負荷量とエネルギー効率の逆数の値を求めた. これらを用いて, 周波数ごとのエネルギー効率の逆数のグラフを図 4 に示す. 横軸は CPU 負荷量で, 縦軸はエネルギー効率の逆数である.

図 4 のグラフ上の点は, 原点に近い方から CPU 負荷率 25%, 50%, 75%, 100%である. 他の値に比べ誤差が大きく精度が低いため CPU 負荷率 0%を省略した.

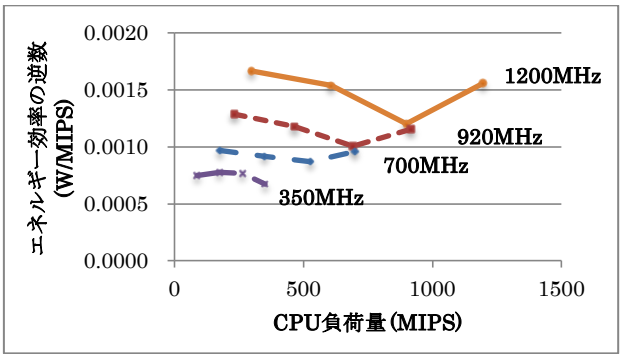

図 4 周波数ごとのエネルギー効率の逆数

5 考察

5.1 モデル式

図 3 より, 各 CPU 負荷率に対して消費電力が一番小さいのはクロック周波数が 350MHz のときのモデル式である. よって, 図 3 のグラフからだけでは, クロック周波数 350MHz が最適なモデル式だと考えられる. しかし, クロック周波数によって単位時間あたりに処理できる命令数は異なる. したがって, 1 命令あたりの消費電力に対するモデルを考えることで, さらに最適なモデルを作成することができる. 作成するには, CPU 負荷率と消費電力だけでなく, CPU 負荷量も考慮する必要がある. CPU 負荷量を考慮したものが図 4 のグラフである.

5.2 周波数ごとのエネルギー効率の逆数

図 4 は, 横軸は数字が小さいほど仕事量が少なく, 縦軸は数が小さいほどエネルギー効率が良いことを表わしている. よって, CPU 負荷量が 0 MIPS から 350MIPS 未満のときはクロック周波数 350MHz, CPU 負荷量が 350 MIPS から 700MIPS 未満のときはクロック周波数 700MHz, CPU 負荷量が 700MIPS から 920MIPS 未満のときはクロック周波数 920MHz, CPU 負荷量が 920MIPS 以上のときはクロック周波数 1200MHz を使用することで, エネルギー効率が良くなると読み取ることができる. すなわち, これらの曲線の下包絡線によって最適エネルギー消費が求まる.

実測では CPU 負荷率が 100%より大きな値になりえないため, 図 4 の線の長さがクロック周波数によって異なる. 図 4 から, クロック周波数によって処理できる単位時間あたりの命令数が異なることがわかった. クロック周波数が小さければ CPU 負荷量は少なく, 大きければ CPU 負荷量は多くなる. また, クロック周波数が上がるほど, 処理時間は短くなるがエネルギー効率は悪くなる.

図 4 から, エネルギー効率が最も良いのは, CPU 負荷量が 349.3073 MIPS でエネルギー効率の逆数が 0.000676 W / MIPS のときであり, エネルギー効率が最も悪いのは, CPU 負荷量が 298.7862 MIPS でエネルギー効率の逆数が 0.001667 W / MIPS のときであると読み取れる. その差は 0.000991 W / MIPS である.

CPU 負荷量が大きくなるにつれて, 他のクロック周波数とのエネルギー効率の差が小さくなっていることがわかる. また, クロック周波数が 700MHz, 920MHz, 1200MHz のときのグラフの CPU 負荷率 75%から 100%にかけてエネルギー効率の逆数の値が増加していることから, クロック周波数 700MHz の CPU 負荷量が 525.7497 MIPS, 920MHz の CPU 負荷量が 691.354 MIPS, 1200MHz の CPU 負荷量が 897.5345 MIPS より大きくなると, エネルギー効率は良くならない.

図 4 の包絡線から, 理想化したモデルの状態遷移図と理想的な応需戦略の状態遷移図を作成することで, CPU 負荷率に応じた, 適切なクロック周波数が明らかになる.

5.3 理想化したモデルと理想的な応需戦略

1 章で述べたように, 負荷に応じて適切なクロック周波数にかえることのできる理想化したモデルの状態遷移図を図 4 から考える. 理想化したモデルを求めるため, CPU 負荷率 100%以上も考慮する.

図 5, 6 の丸の中の数字はクロック周波数を表わす.

[ ]の中の数字は CPU 負荷率を表わし, 次の 2 つの算出方法で求めた. 1 つめは, クロック周波数から CPU 負荷率を求める算出する方法である. 基準にするクロック周波数の CPU 負荷率が 100%のとき, 他のクロッ

ク周波数の CPU 負荷率がそれぞれ何%に相当するかを求めるものである. 3.5 節より, CPU 負荷量は, CPU 負荷率とクロック周波数をかけた値である. 他のクロック周波数を *x* MHz, 求める CPU 負荷率を *y* %とおくと, *x* MHz * *y* % =（基準にするクロック周波数）となる. 2 つめは, CPU 負荷量から CPU 負荷率を算出する方法である. 基準にするクロック周波数の CPU 負荷率が 100%のとき, 基準より小さいクロック周波数が何%に相当するかを求めるものである. 基準より小さいクロック周波数の CPU 負荷率が 100%のときの CPU 負荷量を *a* MIPS, 求める CPU 負荷率を *b* %とおくと, （基準としたクロック周波数の CPU 負荷量の最大 / *a* MIPS ）* 100 = *b* %となる.

丸に遷移するというのは, CPU 負荷率の値が[ ]に当てはまるとき矢印の先にあるクロック周波数にかえるということである.

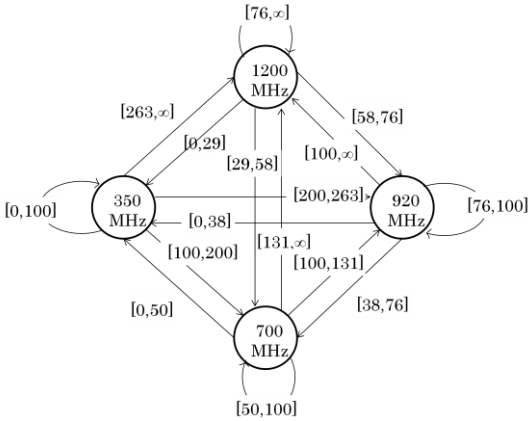

図 5　理想化したモデル

理想化したモデルのメリットは, CPU 負荷率に応じて最適なクロック周波数に一気に変えることが可能なことである. デメリットは, CPU 負荷率が 100%以上は計測不可能なため, 別の方法で見積もる必要があることである.

ondemand governor をもとに, 図 4 から理想的な応需戦略の状態遷移図を考える.

クロック周波数を最も大きい 1200MHz にするため, 他のクロック周波数から 1200MHz に向かう遷移の矢印がある. また, クロック周波数を下げるときは段階的に下げるため, 1200MHz から 920MHz, 920MHz から 700MHz というような矢印がある. CPU 負荷率の値は, 図 5 を参考にした. 以上の結果を図 6 に示す.

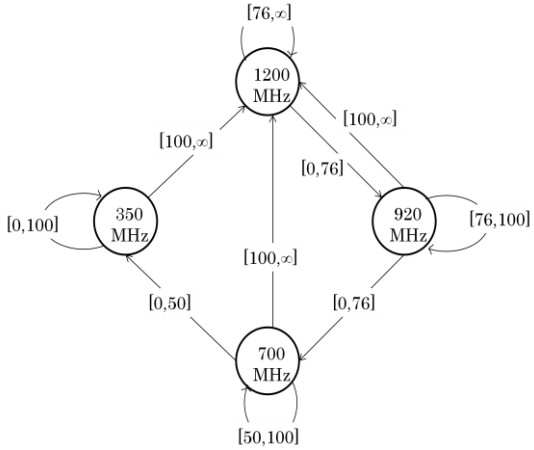

図 6　ondemand governor を用いた理想的な応需戦略

理想的な応需戦略のメリットは, 一定の負荷がかかると 1200MHz に上げてから処理を行うので, 処理時間が早いことである. デメリットは, 一定の負荷がかかると必ず 1200MHz に上げるので, 1200MHz 以外のクロック周波数で十分に処理できるとしても 1200MHz を通ってから下がるため, 余分な動作があることである.

3.4.2 節より, 今回は 2 つのコアに負荷を分散するという条件で実験を行い, 実験結果をもとに図 5, 6 を作成した. 片方のコアだけに負荷が集中した状態で図 5, 6 を用いた場合, エネルギー消費が最適な制御ができない. 例えば, クロック周波数 1200MHz の CPU 負荷率が 40%になったとき, 図 5 では 1200MHz から 700MHz に遷移することが最適である. しかし, 片方のコアだけに負荷がかかった場合, クロック周波数を下げるよりも 1200MHz で処理を行った方が良いため, 図 5 は不適切といえる.

6　おわりに

本稿では, スマートフォンのバッテリ持ちを向上させるために, システムの稼動時に消費電力を可視化し, 消費電力を削減する方法に着目した. クロック周波数を 4 つの値に固定し, CPU 負荷率を増減させるためのベンチマークを作成し, 負荷を 2 つのコアに分散させる条件下でデュアルコアの端末で再現実験を行った. 実測により CPU 負荷率と消費電力の散布図を求め, 仕事量と消費エネルギーのトレードオフを得ることができた. 文献[2]で提案されていたモデル式がデュアルコアにおいても有効であることを実験によって確認できた. 仕事量とエネルギー効率の関連を求め, 理想的な応需戦略と理想化したモデルを設計し, 作成したモデルと測定値の誤差が平均 0.05W, 最大 0.58W であることを確認した.

今後の課題として, クロック周波数を可変にしたものと CPU 負荷率以外の対象としなかった機能の消費電力モデルを生成することが挙げられる.

参考文献

[1] 総務省（編）: 平成 24 年版情報通信白書（2012）.

[2] 石原亨, 奥平拓見, 久住憲嗣ほか: OS から解析可能な無線通信端末の消費電力モデルとその生成手法, 電子情報通信学会技術研究報告, Vol.108, No.463, pp.25–30（2009）.

[3] 岩元直久: スマートフォン 秋冬モデル 最新購入ガイド, 日経 PC21 2013 年 01 月号, pp.14–19（2013）.

[4] 戸川望（編著）, 高田広章, 枝廣正人, 沢田篤史ほか（共著）: 組込みシステム概論, CQ 出版（2008）.

[5] IBM: The ondemand governor, IBM (online), available from 〈http://pic.dhe.ibm.com/infocenter/lnxinfo/v3r0m0/index.jsp?topic=%2Fliaai.cpufreq%2FUnderThecpufreq_base_dir.htm〉 (accessed 2013-12-06).