Vol.2013-ARC-205 No.5
Vol.2013-OS-125 No.5
2013/4/25

Dalvik VMによる細粒度PG制御の動的コード生成

和田 基[1]　塚本 潤[1]　小林 弘明[1]　高橋 昭宏[1]　坂本 龍一[1]　佐藤 未来子[1]　天野 英晴[4]　近藤 正彰[2]
中村 宏[3]　並木 美太郎[1]

概要：細粒度 PG(Power Gating) 制御を行うプロセッサでは，温度やキャッシュのヒット率などの動的なスリープ要因を適切に反映した命令列を実行することが重要である．本発表では，JIT コンパイラの生成するコードに対して，動的要因として実行時のチップ温度を考慮した細粒度 PG 制御を最適化する方式を提案する．筆者らが研究している Geyser アーキテクチャの細粒度 PG 方式を QEMU ベースである AndroidEmulator によってシミュレートした評価実験において，PG 制御を行わない場合と比較し，VM と JIT コンパイラおよび生成されたコードを実行するプロセス全体で平均 6%，最大 22%でリーク電力を削減することができた．

キーワード：パワーゲーティング, 動的コード生成, Dalvik VM, Android

1. はじめに

近年，スマートフォンなどの高度なシステムの普及に伴い，システム LSI に求められる性能や機能は増す一方である．その一方で，高性能化の裏には消費電力の増大という問題があり，また回路技術の微細化によってリーク電力の増大が顕著である．パワーゲーティング技術 (以後 PG 技術) では，使用されていない回路への電源供給を制御することによってリーク電力を削減する．細粒度 PG 技術では，空間的にはプロセッサ内の演算ユニット単位，時間的にはクロック単位での電源制御が行われる．しかし，電源を供給する瞬間，遮断する瞬間には，それぞれ通常時より大きなオーバヘッド電力が発生するため，短い期間の電源遮断 (スリープ) は消費電力を増加させる要因となる．得られる省電力効果とオーバヘッド電力が釣り合うスリープ長を，損益分岐点 (Break Even Point) と定義する．

本研究では，MIPS R3000 を拡張した「Geyser」と呼ばれるプロセッサを扱う．Geyser は ALU，Shift，Mult，Div の四つの演算ユニットに対し細粒度 PG 技術を施しており，各演算ユニットが使用されるときにのみ電源を供給し，使用が終わると遮断する．また，Geyser の特徴として，ソフトウェアから PG を制御するためのインタフェースとして PG 制御付演算命令と PGStatus レジスタを備えている．ハードウェアによる自律制御だけでは，実行する命令列によっては BEP を下回るスリープが多発する可能性があるため，最大限の PG 効果を得るためには，ハードウェアとソフトウェアが協調し PG を制御する必要がある．

ソフトウェアから PG 制御を行う先行研究としては，文献 [7][8] にあげられる ARM プロセッサを拡張した研究が行われている．彼らの研究では PG の制御インタフェースとして PG 制御命令を導入している．PG 制御のみを行う命令を命令セットに追加し，コンパイラによる事前解析に基づき制御命令を挿入する制御手法が提案されている．

Geyser における PG 制御付演算命令を用いた先行研究として，プログラムをコンパイラにより静的に命令列を解析し, 適切な命令列を生成する手法が提案されている [1][2][3]. プログラムの命令列を予測される実行順に解析し，各演算ユニットの使用間隔 (アイドルサイクル) を予測する．BEP より短いアイドルサイクルは，PG 制御付演算命令を用いてスリープを抑制することで PG を制御する．文献 [4] ではこの手法に OS による実行支援を追加し，実行時のチップ温度に応じてオブジェクトコードを切り替える手法を提案している．この手法では，命令粒度での PG 制御を行うことができるが，事前解析によってスリープタイミングを決定するので，キャッシュミス率などの動的な要因に対応しきれない．しかし，最適な PG 制御を行うためには，これらの動的な要因を考慮した PG 制御を行う必要がある．そこで本研究においては，VM の JIT コンパイラに着目

[1] 東京農工大学
Tokyo Univ. of Agriculture and Technology
[2] 電気通信大学
The Univ. of Electro-Communication
[3] 東京大学
Univ. of Tokyo
[4] 慶応義塾大学
Keio University

© 2013 Information Processing Society of Japan

し，動的な要因を反映した PG の制御機構を提案する．

本研究では，JIT コンパイラをもつ VM において，事前解析を必要とせずに動的な要因を反映した PG の制御手法を提案する．JIT コンパイラを持つ VM では，プログラムの実行時に動的にオブジェクトコードを生成・実行する基盤が整っているため，実行時の動的要因を考慮したコード生成が可能である．JIT コンパイラが生成するオブジェクトコード (以後これを JIT コードと呼ぶ) に対し命令列解析を行い，各ユニットのアイドルサイクルを予測することにって，JIT コードの実行時に動的な要因を考慮した上でスリープタイミングを決定する．提案手法を Dalvik VM に構築し，提案手法の有効性の検証を行う．Dalvik VM はトレースベースな JIT コンパイラを持ち，少ないコード修正がプログラム全体に大きな影響を与えるため，提案手法を構築するのに最適である．評価実験としては，3 種類のベンチマークを実行し，プロセス全体における平均リーク電力を算出し PG 制御を行わない場合と比較する．本研究においてプロセス全体とは，JIT コードと VM 本体，JIT コンパイラを合わせた全体の実行を指す．

本稿では，まず 2 章で Geyser プロセッサの PG について詳しく述べ，3 章にて提案手法を試作する Dalvik VM の概要を述べる．4 章に研究の目標を示し，5 章，6 章にて提案手法の設計と実装を述べる．次いで 7 章で評価実験について述べ，8 章で本稿のまとめを述べて終わりとする．

2. Geyser プロセッサの PG

1 章で記したように，Geyser プロセッサはハードウェアによる PG の自律制御の他，PGStatus レジスタ，PG 制御付演算命令によってソフトウェアから PG 制御が行えることが特徴である．これらの制御インタフェースを用いて，BEP 未満のスリープをソフトウェアから抑制する．

PGStatus レジスタによって，自律制御の動作モード (以後 PG モードと呼ぶ) をソフトウェアから指定できる．各ユニットに対して，PG モードを次から指定できる．本研究では動的 PG モードが選択されていることを前提とする．

- 動的 PG - 各ユニットを使用する直前に電源を供給し直後に遮断する
- キャッシュミス時 PG - キャッシュミスが発生した場合にスリープさせる
- 常時 ON - PG を行わず常に電源を供給し続ける

PG 制御付演算命令は，MIPS の R 形式命令を拡張して提供される．通常 $(000000)_2$ であるオペコードを $(100111)_2$ とすることで，もとの命令の演算を行いつつ，動的 PG における演算終了後の電源遮断を抑制する．図 1 に PG 制御付演算命令を実行した際の電源遷移を示す．

PG 制御では，BEP 未満のスリープをいかにして抑制するかが重要となる．BEP 未満のスリープでは，図 2 に示すよう削減効果より電源遷移時のオーバヘッド電力が勝り，

図 1 PG 制御命令による電源遷移

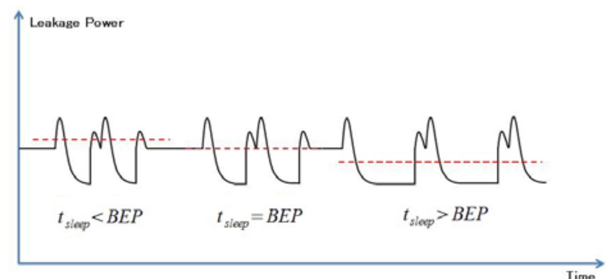

図 2 BEP とスリープ長

表 1 各ユニットの温度ごとの BEP 一覧

	25 ℃	65 ℃	100 ℃	125 ℃
ALU	124	38	18	12
SHIFT	160	50	22	14
MULT	118	44	44	34
DIV	58	14	6	2

リーク電力は増加してしまう．BEP は温度によって変化するため，実行時の温度を考慮した PG 制御が必要となる．またキャッシュミスなどの要因によって命令の実行時間が異なるため，これも考慮しなければならない．Geyser における BEP の値は解析シミュレーションにより表 1 に示す値であることが解析されている．一般に高温になるにつれ BEP は短くなり，温度をパラメータとして指数関数的に変化することが明らかになっている [11]．図 3 に指数近似曲線を示す．なお値の単位はサイクルであり，Geyser は設計上の最高である 200MHz で動作しているものとする．

3. Dalvik VM の概要

Dalvik VM は，米 Google 社が開発した組み込み向け OS「Android」に搭載されている Java ライクな仮想マシンである．Dalvik VM には，トレースベースの JIT コンパイラ (以下 Dalvik-JIT) が搭載されている．Dalvik VM では，バイトコード中の分岐命令間の命令列を実行単位 (ブロック) とみなし，アプリケーション実行中に頻繁に実行されるブロックをコンパイルする．Dalvik-JIT により生成された JIT コードは専用のメモリ領域に保存され，くりかえし実行することによって処理の高速化を図っている．アプリケーションは JIT コードとインタープリタによる実行処理を切り替えながら実行される．

© 2013 Information Processing Society of Japan

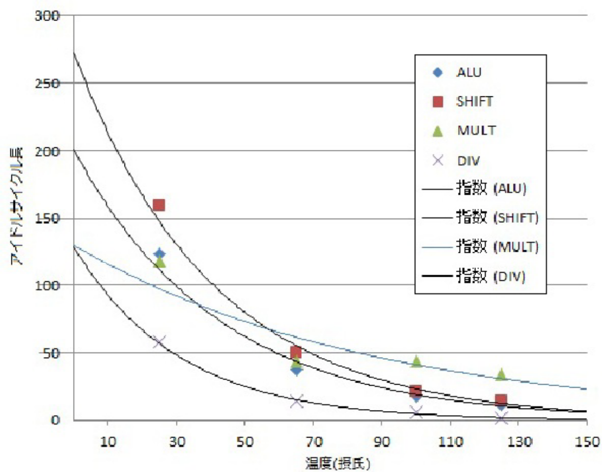

図 3　BEP の近似値

一度生成された JIT コードはアプリケーション実行中に何度も実行され，アプリケーションによってはインタープリタと比較して JIT コードの実行割合が高くなるほどである．そのため，JIT コードに対する小さな変更がアプリケーション全体に大きな影響を与える．

4.　目標

Geyser における PG 制御では，ソフトウェアによるスリープタイミングの制御がリーク電力の削減に重要であり，BEP の温度依存性を考慮した上でスリープタイミングを定める必要がある．Dalvik VM の JIT コンパイラでは動的に命令列を生成する基盤を有しているため，これを拡張し，動的要因を反映した PG 制御を行う JIT コードを動的に生成し，電力削減効果を高めることを目標とする．Dalvik VM において，一度生成された JIT コードはプログラムの実行中に繰り返し実行され，プロセス全体に占める JIT コードの実行割合は高くなる．この理由から電力削減に寄与するような JIT コードの修正はプロセス全体としてより良い効果を期待できるため，提案手法による効果は大きいと期待される．評価実験として三つのベンチマークを実行し，提案手法による PG 制御を行わない場合と，プロセス全体における平均リーク電力を比較し，有効性の検証を行った．

5.　提案手法の設計

5.1　設計方針

電源状態の遷移時に発生するオーバヘッド電力がゼロの場合，アイドルサイクルの長さに関わらずユニットが使われない間はスリープさせるのが理想的である．しかし，実際にはオーバヘッド電力が発生し，BEP 未満の長さのスリープは電力的に不利になってしまう．そのため，電力的に不利となる BEP 未満の長さのスリープを抑制するよう PG を制御することが求められる．

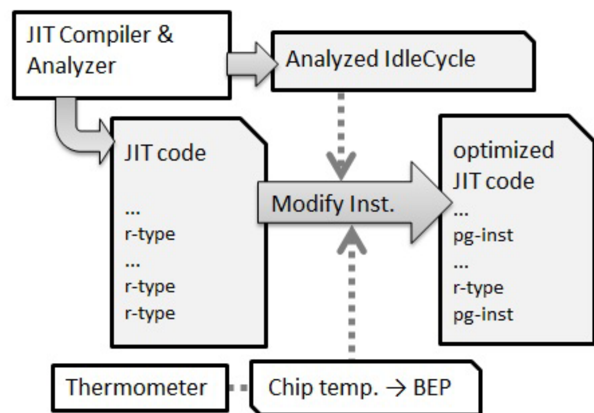

図 4　JIT コードの省電力化

しかしハードウェアによる PG 制御だけでは，実行する命令列によっては電力的に不利なスリープが多発してしまう．より効果的にリーク電力を削減するためには，ソフトウェアによる制御が必要不可欠である．しかし，ソフトウェアによる制御処理からオーバヘッドが発生しないよう，制御処理は簡潔であることが望ましい．Dalvik VM において，JIT コンパイラによってコンパイルされるバイトコードのメモリ量はプログラム全体でみたら少ないが，生成された JIT コードは何度も繰り返し実行されるため，実行時間に占める JIT コードの割合は高くなる．すなわち，JIT コードに対する変更は，プログラムの実行全体に大きく影響を与えることになる．JIT コンパイルに着目した PG 制御を行うことによって，少ないオーバヘッドで大きな効果が見込まれる．

5.2　省電力の命令列を生成する Dalvik VM の構成

提案方式では，JIT コンパイラの生成した JIT コードの実行を省電力化する．そのために，BEP 未満のスリープが起こらないよう JIT コード中の R 形式命令を PG 制御付演算命令に修正する．通常，Geyser では演算ユニットには使用されるときのみ電源を供給するが，PG 制御付演算命令の実行後は使用したユニットの電源供給を継続する．そのため，ユニットの使用間隔が狭く BEP 未満のスリープが発生すると予測される場合は，PG 制御付演算命令を用いて電力的に不利なスリープを抑制する．図 4 に JIT コード省電力化の全体像を示す．JIT コンパイラの生成した JIT コードに対して，チップ温度などの動的な要因を考慮した PG 制御情報を PG 処理付演算命令として付加し適切な電源制御を行う．

提案手法では，スリープサイクルを予測するために生成された JIT コードの電力解析を行う．解析は JIT コンパイルが行われ JIT コードが生成された直後に行われる．解析された情報をもとに，JIT コードの実行直前にチップ温度などの動的な要因を考慮して JIT コードの命令語修正を行い動的な要因に追従していく．

© 2013 Information Processing Society of Japan

提案手法では 2 段階の処理によって動的な要因に追従した PG 制御を行う．第一段階では事前準備処理として JIT コードの解析処理を行い，第二段階で実行準備処理としてチップ温度やキャッシュヒット率などの動的な要因を考慮し実行時の状況に最適化された省電力 JIT コードを生成する．

5.3 省電力 JIT コードの生成アルゴリズム

JIT コードはプログラムの実行中に何度も実行されるため，各実行時のチップ温度などの動的な要因を適時反映していかなければならない．提案手法では，一度生成された JIT コードを修正することで状況に応じたコードを生成する．そのため，ひとつのバイトコードブロックの JIT コンパイルは一度しか行わず，JIT コードとバイトコードブロックは常に一対一に対応した状態となる．省電力化は命令のオペコードだけの修正で可能である．

JIT コードを省電力化するためには，実行時に BEP 未満のスリープが発生しないよう PG を制御する必要がある．2 章で示したように，R 形式命令のオペコードを書き換えることによって PG 制御付演算命令を利用できるため，R 形式命令の実行後に始まる BEP 未満のスリープが抑制可能である．そのために，JIT コード中に存在する各 R 形式命令の使用するユニットの使用される間隔，アイドルサイクルを解析する．Geyser の動的 PG モードにおいて，各ユニットへの電源供給はそのユニットが使用される直前に開始され，直後に停止するので，ユニットのアイドルサイクルとスリープサイクルは一致する．そのため，アイドルサイクルを解析することによって，実行時に各ユニットのアクティブ・スリープの挙動を予測することができる．

JIT コード中の各 R 形式命令に対して，各命令の実行直後のアイドルサイクルとユニットの BEP を比較し修正するか否かを決定する．BEP よりアイドルサイクルが短い場合はスリープさせると電力的に不利となるため，スリープを抑制するよう命令オペコードを $(100111)_2$ に修正する．逆にアイドルサイクルが BEP より長い場合，これはスリープさせるとリーク電力を削減できるため，命令オペコードを $(000000)_2$ として通常通りユニットのスリープが行われるよう制御する．図 5 に命令語修正の方法を図解する．ユニットの BEP は温度によって変動するため，実測したチップ温度をもとにユニットの BEP を算出する．

このように，JIT コード中に存在するすべての R 形式命令 (もしくは置換された PG 制御付演算命令) は，常にすべて同じ温度にむけて最適化された状態になる．JIT コード中の対象命令がどの温度に向けて最適化されているかは記録しておき，温度に追従するための指標として 5.5 節で述べる命令語修正を行うタイミングを決定するのに用いる．

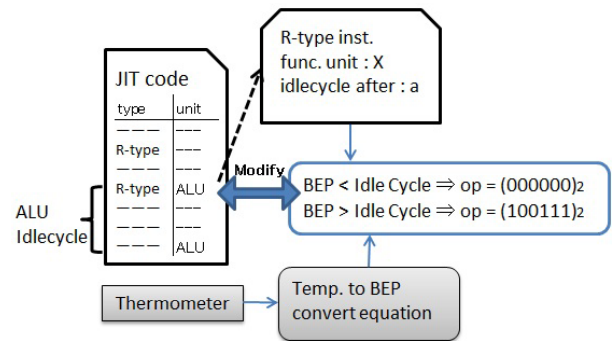

図 5 JIT コードの省電力化方法

5.4 Dalvik VM による電力制御のモニタ機構

5.3 節で示したように，JIT コードの省電力化のためには，各 JIT コードのアイドルサイクル情報，各ユニットの各温度における BEP，そして修正対象の候補となる JIT コード中の R 形式命令の位置を知る必要がある．提案手法では，JIT コンパイルが行われた直後に JIT コードの解析を行いアイドルサイクルを予測する．また，各ユニットの BEP は表 1 の温度と BEP の対応を用いて算出する．

5.4.1 アイドルサイクル解析

アイドルサイクルとは，PG 対象となる四つのユニットが使われる間隔をサイクル単位で表したものである．PG 制御付演算命令は R 形式命令のオペコードを修正して用いるため，R 形式命令が実行されてから，次に同じユニットを使用する命令が実行されるまでの間隔のみをアイドルサイクルとして数える．図 6 中の①，②は R 形式命令から始まっているため ALU のアイドルサイクルとしてとらえるが，③では先頭が I 形式命令であり実行時に PG 制御を行うことができないため，ここでは対象から外す．

図 7 に示すよう，アイドルサイクルの解析は JIT コンパイルが行われた直後に行われる．Geyser において抑制できるスリープは R 形式命令が実行された直後に発生するスリープのみである．そのため，アイドルサイクルの解析は R 形式命令から始まるアイドルサイクルに関してのみで十分である．JIT コード中に存在する各 R 形式命令に対して，その R 形式命令の使用するユニットを判定し，次に同じユニットが実行されるまでに実行されると予測される命令数を計測する．

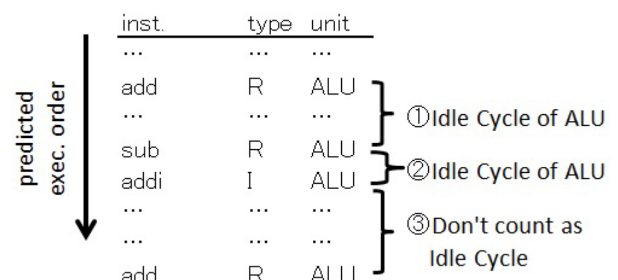

図 6 アイドルサイクルの例

© 2013 Information Processing Society of Japan

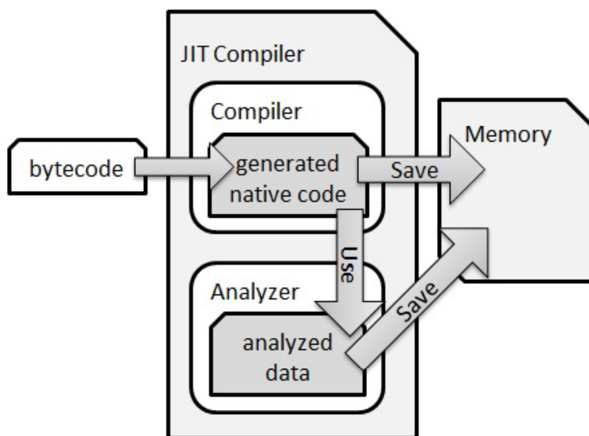

図 7　JIT コンパイルから解析の流れ

通常プロセッサではプログラムカウンタの示すアドレスの命令を実行し，インクリメントして命令の実行順を定める．そのため実行順はアドレス番地の低い順にすればよい．しかし，分岐命令などによって実行順が一意に定まらない場合も多い．そこで，提案手法においては，分岐するか否かを確率的に定める．分岐の統計情報などから分岐確率を考慮することが望ましいが，本研究では最初の実装として双方向に等しい確率で分岐先を決定することとする．

また，一般にキャッシュやパイプラインの状態によって，ある命令の実行にかかるサイクル数は変動する．パイプラインが理想的に流れキャッシュミスが起きない場合，命令は 1 サイクルごとにフェッチされ見かけ上 1 サイクルで実行される．しかし，パイプラインがストールする場合やキャッシュミスが発生すると，命令の実行にかかるサイクル数は伸びていく．そのため，命令数を数えるだけではなく，実行にかかるサイクル数を推定することが望ましい．しかし本研究では，1 命令の実行にかかるサイクル数は 1 サイクルと仮定して解析を行い，キャッシュなどの要因によるアイドルサイクルの伸びは JIT コード実行直前の命令修正時に補正する方式を考える．統計情報を用いた分岐予測とキャッシュミス予測は今後の課題とする．

5.4.2　BEP の算出

回路の BEP は，その回路規模や温度によって変動する．そのため，各ユニットに対して，チップ温度から BEP への変換式を用意して BEP を算出する．BEP は温度に対して指数関数的に増減するため，表 1 に示した 4 つの温度における BEP をもとに，指数関数として近似式を作成する．Geyser の各ユニットに対して，この近似式は次のようになる．

$$BEP_{ALU}(t) = 200.85 \times e^{-0.023t} \tag{1}$$

$$BEP_{SHIFT}(t) = 272.62 \times e^{-0.024t} \tag{2}$$

$$BEP_{MULT}(t) = 130.14 \times e^{-0.011t} \tag{3}$$

$$BEP_{DIV}(t) = 128.74 \times e^{-0.033t} \tag{4}$$

JIT コード中の R 形式命令を修正するときはチップ温度を計測し，その命令の使用するユニットに応じて式 (1) から式 (4) を用いて BEP を算出する．近似式は浮動小数点演算を含むため，Geyser のように浮動小数点演算ユニットのない環境においては計算処理に時間がかかってしまう．そこで，常温近辺の温度における値は事前に計算しておき，その結果を参照することによって高速化する．

5.5　命令語修正のタイミング

命令語修正は，各 JIT コードに対して独立に行われ，生成された JIT コードが初めて実行される直前に必ず行われる．しかしチップ温度などの動的な要因を適時反映させていくためには，命令語修正を随時行っていく必要がある．修正された JIT コードは，チップ温度などの動的な要因が変化しない間はそのまま使用し続け，変化があった場合に再度修正を行い動的な要因に追従していく．

命令語の修正は，既存の (修正された)JIT コードでは BEP 未満のスリープが抑制しきれないと判断された場合に行う．BEP は温度によって変動するので，チップ温度が一定以上変動した場合がこれにあたる．既存の JIT コードが最適化されているチップ温度を T_{opt} としたとき，再修正が不必要なチップ温度 T を次の式で定義する．

$$|\overline{BEP}(T) - \overline{BEP}(T_{opt})| \leq n \tag{5}$$

ここで n は自然数とする．また $\overline{BEP}(t)$ は全 PG 対象ユニットの BEP の平均を表し，Geyser においては次式となる．

$$\overline{BEP}(t) = 163.89 \times e^{-0.02t} \tag{6}$$

式 (5) が成り立たなくなった場合，最適化温度における BEP と現在の温度における BEP の差が大きくなりすぎたと判断し，命令語の修正を行う．n の値を小さくすることによって，命令語修正を頻繁に行い温度への追従を高めることができる．しかし，命令語修正を行う回数が多いと処理オーバヘッドにつながり，性能が低下してしまう．処理オーバヘッドを削減することはどのようなシステムにおいても重要である．Android などの組み込み機器においては，処理性能の低下はエネルギー性能の低下を引き起こすことが文献 [5][6] にて示されており，エネルギー面でも高い削減効果を得るためにはオーバヘッドの削減が重要となる．実行するアプリケーション特性に合わせた値を設定す

ることで，オーバヘッドを最小限に抑えエネルギー効率を向上させたうえで，温度へ追従した PG 制御が可能となる．

修正の必要の有無は，その JIT コードが実行される直前に確認する．実行直前に行うことによって，無駄な修正をおさえた上で，実行時のチップ温度などに最適化された省電力 JIT コードの実行が可能となる．

6. 提案手法の実装

提案手法の有効性を検証するため，Dalvik VM 上に提案手法を構築し，Geyser プロセッサの PG モデルに対応させた Android Emulator(QEMU) 上で動作させる．構築した環境を表 2 に，構成図を図 8 に記す．図 8 において，星のついている機能を提案手法の一部として Dalvik VM に追加した．

通常 Dalvik VM では，JIT コードは OS の提供する ashmem(anonymous shared memory) 領域に確保されたコードキャッシュ (図 8 中の dalvik-jit-code-cache) に JIT コードを保存する．アイドルサイクル解析による解析データも同様に ashmem 領域に保存するため，そのための領域として PG 情報キャッシュ (図 8 中の pg-info-cache) を確保した．

JIT コンパイルが行われた直後に，生成された JIT コードのアイドルサイクルを解析する機能を追加した．解析対象の JIT コード中に存在する全 R 形式命令のアドレスと使用ユニット，直後のアイドルサイクル情報を PG 情報キャッシュに登録する．また，その JIT コードは絶対零度未満の値など，システムが存在しえない温度に最適化されていることにし，初回実行時に必ず命令語修正が行われるようにした．

命令語修正を行うために，処理が JIT コードに移行する直前に 5.5 節で述べた修正の必要の有無を判定するよう

表 2 構築環境

項目	製品名等
評価環境	Android Emulator (QEMU)
CPU アーキテクチャ	MIPS32 rev2
細粒度 PG 方式	Geyser アーキテクチャ
OS カーネル	Android Kernel 3.0.8
OS ユーザランド	MIPSANDROID 4.0 (ICS)

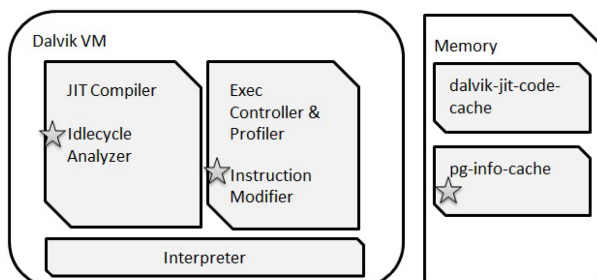

図 8 システムの構成図

表 3 開発評価環境

項目	製品名等
OS	Fedora 17 (Linux Kernel 3.3)
CPU	Intel Xeon E5606 x2
Toolchain	GNU Toolchain by Google (gcc 4.5.2)

表 4 コードの変更量

	追加/挿入行数
解析機能	500
命令語修正機能	300

記述した．修正が必要ないと判断つされた場合はそのまま JIT コードを実行し，必要な場合は修正処理を行ったのちに JIT コードを実行する．コードキャッシュは通常，JIT コンパイラが動作している時以外は書き込みに制限がかけられているため，コード修正前に書き込み権限を追加し，修正後に書き込み権限を破棄する処理を行う．解析されたアイドルサイクルと BEP を比較し，アイドルサイクルの直前の R 形式命令をその長さによって制御付演算命令に置換する．アイドルサイクルが BEP より短い場合オペコードを $(100111)_2$ に書き換え，長い場合は $(000000)_2$ に書き換えるよう処理を追加した．修正後はキャッシュフラッシュを行い主記憶上に変更を反映させている．

7. 評価実験

提案手法の評価として，ベンチマーク実行時のプロセス全体の平均リーク電力を，PG 制御を行わない場合と比較する．評価実験は QEMU ベースである Android Emulator 上で行うため，Android Emulator を Geyser に対応するよう拡張した．

7.1 評価環境の構築

Geyser の提供する PG 制御付演算命令を実行するため，命令セットに Geyser 用 PG 制御付演算命令を追加した．また，全命令は 1 命令 1 サイクルで実行されるものと仮定し，実行命令のトレース機能を追加した．この命令トレースで，各ユニットのアイドルサイクルを調べた．アイドルサイクルを各スリープとしてカウントし，評価実験を行う．この評価用機能への制御インタフェースとして，制御命令も追加した．命令列中に MIPS R3000 の命令セットに存在しない「$0x5a5aa5a5$」というワードで命令列を囲むことによって，その命令列の実行中に発生するスリープの分布を得ることがてきる．

7.2 平均リーク電力の算出

実行時の平均リーク電力は，Android Emulator に構築した命令トレース機能を用いてスリープ分布を求め，次の計算式によって算出することができる．

$$\overline{P} = \frac{\overline{P_{sleep}} \times T_{sleep} + \overline{P_{active}} \times T_{active}}{T_{total}} \quad (7)$$

ここで，T_{total} は実行総サイクル数，$\overline{P_{sleep}}$ はスリープ時平均リーク電力，T_{sleep} は総スリープサイクル数，$\overline{P_{active}}$ はアクティブ時平均リーク電力，T_{active} は総アクティブサイクル数である．スリープ時平均リーク電力とアクティブ時平均リーク電力は，文献 [9][10] で計算されている値を用いる．スリープ時平均リーク電力に関しては，各スリープサイクル毎の平均リーク電力のデータを用いて，評価機構により得られたスリープ分布をもとに次式によって算出した．

$$\overline{P_{sleep}} = \frac{\sum_i (P_{sleep_i} \times T_{sleep_i})}{\sum_j T_{sleep_j}} \quad (8)$$

7.3　評価ベンチマーク

Caffeine Mark v3 より，Sieve，Method，Loop ベンチマークを実行する．処理内容はそれぞれエラトステネスのふるいによる素数判定，関数の再起呼び出し，クイックソート等である．65 ℃，100 ℃，125 ℃の環境として評価実験を行う．現時点では温度変化を伴うシミュレーションは行えないため，実行時はチップ温度が変動しない環境での評価とした．

7.4　評価結果

図 9 に実験結果を示す．Loop ベンチマーク，Sieve ベンチマークにおいては，全温度で平均リーク電力の削減が確認できた．Method ベンチマークにおいては，全温度で平均リーク電力が増加してしまっている．全体では平均 6%，最大 22%のリーク電力削減に成功した．ユニット別に見ると，100 ℃における Sieve ベンチマークで Div ユニットが最大 47%削減している．次節にその考察を示す．

7.5　JIT コード実行時のリーク電力

大きな削減効果が得られた 125 ℃における Sieve ベンチマークと効果が得られなかった 100 ℃における Method ベンチマークについて，JIT コードの実行中の平均リーク電力を計測した．図 10 にその結果を占めす．

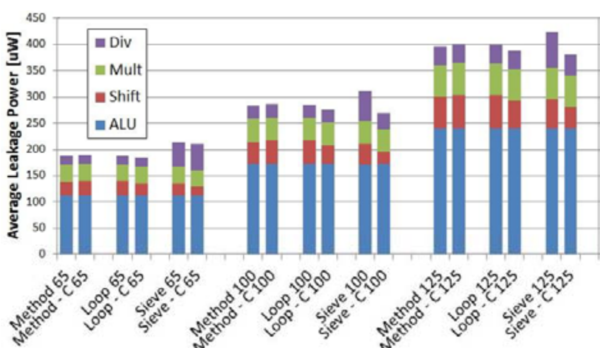

図 9　プロセス全体の平均リーク電力

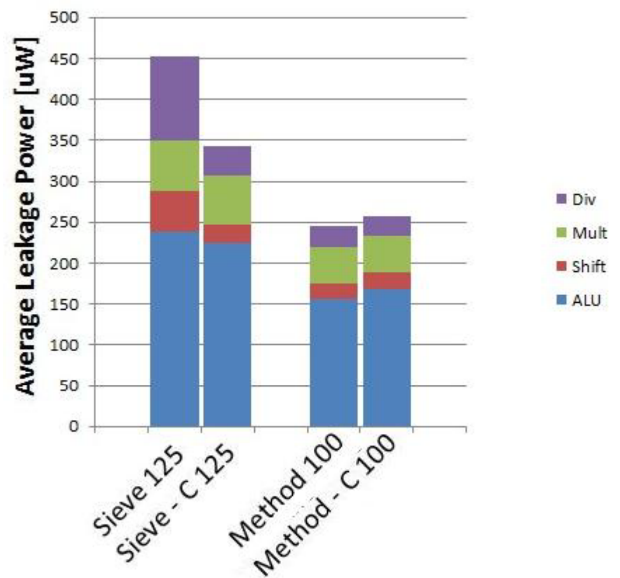

図 10　JIT コード実行中の平均リーク電力

図 9, 図 10 からわかるよう，プロセス全体で見たときのリーク電力の増減は，JIT コードのみ見たときの平均リーク電力の増減と一致している．Method ベンチマークにおいて JIT コード実行時の平均リーク電力が増加しているが，これは BEP 以上のスリープを抑制してしまったことが原因である．BEP 未満であると解析されたアイドルサイクルが実際は BEP 以上であった場合，このスリープは不利と判断し抑制されリーク電力を増加させてしまう．アイドルサイクルを解析する際，分岐命令が存在する場合は分岐先を当確率で確率的に選択するため，分岐命令が多いと解析精度が落ちる．解析精度を向上させることによって，本実験で効果の得られなかったベンチマークに対しても効果が見込める．

7.6　JIT コードの実行割合

提案手法において，PG 制御が行われるのは JIT コードの実行中のみである．そのため，全体の実行のうち JIT コードの占める割合が高いほど，提案手法による効果が大きくなる．Sieve，Loop，Method ベンチマークについて，総実行命令数に占める JIT コード命令の割合を図 11 に示す．

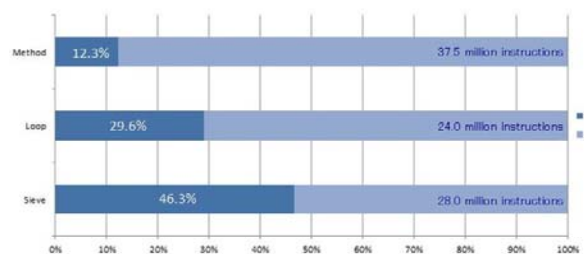

図 11　JIT コードの占める実行割合

JIT コードのしめる実行割合は Method, Loop, Sieve の順に低く，全温度平均で見たときの平均リーク電力削減率

の大小関係と一致する．JIT コードのしめる実行割合は，バイトコード中の数値演算系の命令が多いほど高い傾向がみられた．Method ベンチマークでは関数を再帰的に呼び出す処理が多く，数値演算は少ない．それに対し Sieve ベンチマークでは，素数判定で整数四則演算を多く行っているため，JIT コードの占める実行割合が高くなった．

図 11 で示したように，今回実験を行ったベンチマークでは，JIT コードの実行割合は最大でも 47%であり，実行全体の半分以上がインタープリタと VM の実行が占めていた．提案手法では JIT コード実行中のみ PG 制御を行い，JIT コード以外の VM の実行時は PG 制御を行わない．さらに高い省電力効果を得るためには，インタープリタを含む VM 自体でも PG 制御を行う手法も適用可能である．VM 自体の実行時も PG 制御を行うためには，文献 [4] で提案されているコンパイラによる静的解析を行う従来の PG 制御手法を適応する．Dalvik VM の事前解析を行うことで，JIT コンパイラ自身の省電力化が可能となる．

8. まとめ

本研究では，JIT コンパイラに着目し，事前解析の必要のない細粒度 PG の制御手法を提案した．JIT コンパイラの生成するコードに対し PG の制御コードを埋め込むことによって，動的な要因を考慮したコード生成を可能とした．また提案手法を Dalvik VM 上に構築し，提案手法の有効性の検証を行った．

評価においては, Geyser に対応させた Android Emulator 上で，三つのベンチマークを実行し平均リーク電力を算出した．PG 制御を行わない場合と比較し，提案手法を用いた場合，平均リーク電力を平均で 6%，最大 22%削減することに成功した．また今回構築した Dalvik VM では，アイドルサイクル解析の精度が低く JIT コード実行時の平均リーク電力を増加させる結果となってしまった．アイドルサイクル解析の精度を向上させ，PG 制御の精度を向上させることの必要性が明らかになった．

今後の課題としては，温度変化を伴う環境での評価実験が必要である．QEMU を用いた実験では，温度変化を伴う実験はデータ不足で困難である．温度変化を伴う環境で評価実験を行うためには，Geyser の実環境での実験が必要である．そのために，まず Geyser 実環境への Android およびに Dalvik VM の移植を行う．現在，Geyser 実環境での Linux の動作が確認されている．これをもとに Android システムの移植を行い，Geyser 実環境上で提案手法を適用した Dalvik VM を動作させ，実際に電力を計測することによって実験を行う．

謝辞 本研究は，科学技術振興機構「JST」の戦略的創造研究推進機構「CREST」における研究領域「革新的電源制御による次世代超低消費電力高性能システム LSI の研究」によるものである．

参考文献

[1] 薦田登志矢，佐々木宏，近藤正章，中村宏，“リーク電力削減のためのコンパイラによる細粒度スリープ制御”，SACSIS 2009, pp.11-18 (May 2009).
[2] 木村一樹，2008 年度卒業論文 “省電力 MIPS プロセッサコア評価のための計算機システムの FPGA による試作”，東京農工大学 (Jan 2009).
[3] 茂木勇，2009 年度卒業論文 “省電力 MIPS プロセッサ評価ボードへの Linux の移植”，東京農工大学 (Feb 2010).
[4] 小林弘明, 2010 年度卒業論文 “OS における細粒度パワーゲーティング向けオブジェクトコードの実行時管理機構の研究”，東京農工大学 (Jan 2011)
[5] S. Borkar. ”Microarchitecture and Design Challenges for Gigascale Integration.” In The Proceedings of 37th Annual IEEE/ACM International CSymposium on Microarchitecture, pp.3-3 (2004)
[6] Tapas Kumar Kundu, Kolin Paul, ”Improving Android Performance and Energy Efficiency”. In The Proceedings of 24th Annual Conference on VLSI Design, 2011, pp.256-261
[7] S. Roy, N. Ranganathan, and S. Katkoori, “A Framework for Power-Gating Functional Units in Embedded Microprocessors”, IEEE Transaction of Very Large Scale Integeration Systems vol.17, pp.1640-1649 (Nov 2009)
[8] Roy, Soumyaroop, ”A compiler-based leakage reduction technique by power-gating functional units in embedded microprocessors” (2006). Graduate School Theses and Dissertations. (http://scholarcommons.usf.edu/etd/2682)
[9] 中田光貴, 白井利明, 香嶋俊裕, 武田清大, 宇佐美公良, 関直臣, 長谷川揚平, 天野英晴, “ランタイムパワーゲーティングを適用した回路での検証環境と電力見積もり手法の構築”, 電子情報通信学会信学技報 VLD2007-111, pp.37-42 (Jan 2008).
[10] 白井利明，香嶋俊裕，武田清大，中田光貴，宇佐美公良，長谷川揚平，関直臣，天野英晴，“ランタイムパワーゲーティングを適用した MIPS R3000 プロセッサの実装と評価”，信学技報，vol.107，no.414，vld2007-111，pp.37-42 (Jan 2008).
[11] 宇佐美公良，橋田達徳，”細粒度パワーゲーティングにおける損得分岐時間の温度依存性モデルと温度適応型制御”，電子情報通信学会信学技報 VDL2010-8, pp.74-78 (Jan 2010)