Vol.2009-ARC-184 No.14
2009/8/4

コンパイラによる細粒度スリープ制御のためのアーキテクチャ支援技術の検討

近藤正章[†1]　薦田登志矢[†2]
佐々木広[†3]　中村宏[†2,†3]

近年，リーク電流による消費電力増加が問題となっており，待機時だけでなく，アプリケーション実行中のリーク消費電力も無視できなくなっている．走行時パワーゲーティングは，演算器などのユニットがアイドルの際に，パワーゲーティング手法により電源供給を遮断することでリーク消費電力の削減を狙うものである．パワーゲーティング手法では，アクティブ／スリープモードの切り替え時に性能やエネルギー面でのオーバーヘッドが生じるため，効率的に実行時のリーク電力を削減するためには，オーバーヘッドの影響を考慮しつつ，モード切り替え制御を行なう必要がある．本稿では，インオーダ命令発行のプロセッサの演算器を対象に，コンパイラとアーキテクチャが連携しつつ，モード切り替えを制御する手法を提案する．提案手法を評価した結果，通常のプロセッサに比べ，0.43%程度の性能低下で約 80%の算器部のリーク消費エネルギーを削減可能であることがわかった．

An Architectural Support for Compiler-Directed Fine Grain Power-Gating

Masaaki Kondo,[†1] Toshiya Komoda,[†2]
Hiroshi Sasaki[†3] and Hiroshi Nakamura[†2,†3]

As leakage-power consumption becomes dominant in the total power consumption of LSI chips, it is necessary to reduce the leakage current not only for the standby period but also for the application running time. Run-time power gating (RTPG) is a technique to reduce leakage power during the program execution by turning on and off circuit components in much finer temporal/spatial granularity. The problem associated with RTPG is the time delay for restarting the execution and the dynamic power overhead for active-sleep mode transition. To address these issues, in this paper, we propose a compiler and architecture co-operative power-gating control technique. We evaluate the proposed technique and the results reveal that the proposed technique can control the power-mode effectively so that about 80% of leakage energy of the functional units is reduced with 0.43% performance penalty.

1. はじめに

消費電力・消費エネルギーの削減は LSI を設計する上での最も重要な課題の一つである．LSI チップの消費電力には，トランジスタのスイッチングによるダイナミック消費電力とリーク電流によるリーク消費電力があるが，半導体プロセスの微細化にともない，近年ではリーク消費電力がチップ全体の消費電力の多くを占めるようになっている．また，リーク消費電力は今後も増大すると予測されているため，その削減は重要な技術課題である．

従来より，特にモバイル用途のプロセッサにおいて，待機時やシステムアイドル時のリーク電流を削減するための手法が多く提案されている．例えば，パワーゲーティング (PG) 手法は，MTCMOS 技術を用い，動作させる必要のないロジックやメモリセルへの電源供給を閾値の高いトランジスタを用いて遮断することで，対象回路のリーク電流を削減するものである[9]．PG により，待機時のリーク消費電力を大幅に削減することができるが，今後のリーク電流増大を考えると，待機時やシステムアイドル時だけでなく，アプリケーション実行中のリーク消費電力も無視することはできない．したがって，性能低下なく走行時のリーク電流を削減するための手法が必要となる．

これまでにもキャッシュにおける実行時のリーク電流を削減する手法は多く提案されている[5),7)]．プロセッサではトランジスタの多くがキャッシュに費やされており，またリーク消費電力はトランジスタ数に比例して増大することから，キャッシュのリーク電流を抑えることはチップ全体の消費電力削減に有効である．一方，文献 2) のリーク消費電力モデルによると，演算器などの組み合わせ回路はトランジスタ数は少ないものの，特性の違いからトランジスタあたりのリーク消費電力が大きいとされている．したがって，キャッシュのみならず，演算器を含めたプロセッサ全体のリーク消費電力削減を考えることが重要である．

PG は，時間的・空間的に細粒度に電源電圧供給制御を行うことが可能であるが，通常の動作を行うモード（アクティブモード）とリーク電流を削減するためのモード（スリープモード）の切り替え時には，性能やエネルギー面でオーバーヘッドが生じる．そのため，効

†1 電気通信大学 大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications
†2 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†3 東京大学 先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo

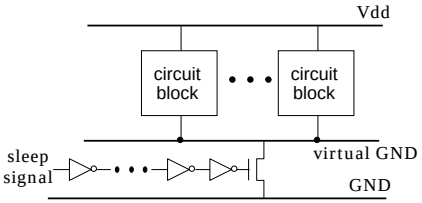


図 1 パワーゲーティングの概要

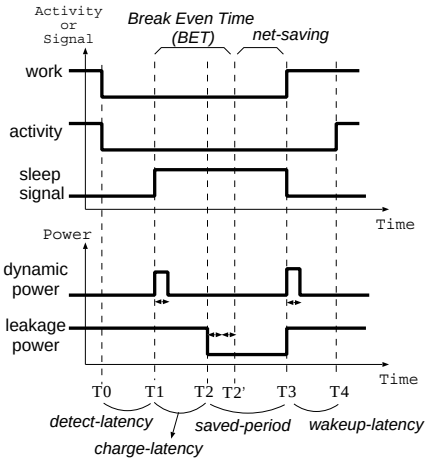


図 2 モード切り替え時のオーバーヘッド

率的に実行時リーク電力を削減するためには，オーバーヘッドの影響を考慮しつつ，モード切り替えの制御をする必要がある．

そこで本稿では，インオーダ命令発行のマイクロプロセッサの各演算器を対象に，コンパイラとアーキテクチャが連携しつつ，細粒度 PG のためのモード制御を行なう手法を提案する．提案手法は，コンパイラにより各演算器のアイドル期間を予測し，実行時にはコンパイラで生成したモード切り替え情報を利用して，ハードウェアがプロセッサ内の演算器のモードを制御するものである．本手法により，モード切り替えを実現するためのハードウェアコストを抑えつつ，オーバーヘッドの影響を抑えることが可能となり，効率的な走行時 PG が実現できると考えられる．

2. 走行時リーク電力の削減

2.1 パワーゲーティング手法

パワーゲーティング (PG) 手法は，動作させる必要のないロジックやメモリセルへの電源供給を遮断することで当該回路のリーク電流を削減するものである．この電源供給制御のために，スリープ信号により制御されるスリープトランジスタを電源線と回路ブロック間，あるいはグラウンド線と回路ブロック間，またはその両方に挿入する．図 1 はグラウンド線と回路ブロック間にスリープトランジスタを挿入した場合の PG 手法の概要を示したものである．

図のスリープ信号がアサートされると，グラウンド線 (Gnd) と仮想的なグラウンド線 (virtual Gnd: 以降では VGND と表記) 間のスリープトランジスタがオフになり，回路ブロックへの電源供給が遮断され，リーク電流を削減することができる．PG におけるモードの切り替え，すなわちスリープトランジスタのオン/オフを切り替える場合，スリープ信号の伝搬やスリープトランジスタの駆動，またスリープ期間中に VGND に溜まった電荷の放電などのために時間的・エネルギー的なオーバーヘッドが生じる．

図 2 はモード切り替え時のオーバーヘッドについて説明するために，横軸に時間経過を，縦軸に処理の有無や消費電力を示したものである．図は時刻 T0 において，ある回路ブロックにおける実行可能な処理 (work) がなくなり，同時にその回路ブロックでの処理 (activity) が行われずアイドル状態となる場合を示している．このアイドル状態を検出し，時刻 T1 でスリープ信号 (sleep signal) をアサートすることによりスリープモードに移行する．ここで，時刻 T1 ではまだ VGND に電荷が流れていくため，すぐに対象の回路ブロックでのリーク電流が削減できるわけではなく，一定の時間が過ぎた時刻 T2 よりリーク電流の削減が可能となる．次に時刻 T3 で再び実行可能な処理が現れ，スリープ状態から復帰する必要がある．ここで，スリープトランジスタをオンにして電源を供給しても，VGND に溜った電荷の放電に時間を要するため，すぐに実行が再開できず，時刻 T4 において処理が再開可能となる．

図の例では時刻 T3 で実行可能な処理が *wakeup-latency* (WL) サイクル分遅延させられている．これが PG を行う際の時間的なオーバーヘッドとなる．また，実行可能な処理がないアイドル期間は時刻 T0 から T3 の間であるのにもかかわらず，そのアイドルを検出してスリープモードに移行する遅延の *detect-latency*，およびスリープモードに移行してから実際にリーク電流が流れなくなるまでの遅延である *charge-latency* のために，リーク電力が削減できる期間は時刻 T2 と T3 の間の *saved-period* サイクル分の時間となる．さらにモード切り替え時のダイナミック電力 (図中の dynamic power) を考慮すると，実際に削減できるエネルギーはそのダイナミック電力の増加分を差し引いた *net-saving* 分のリーク電力となる．この，スリープモードに移行してから削減できたリーク電力とダイナミック電力のオーバーヘッドが釣り合うまでの時間 (時刻 T1 から T2')，すなわち損益分岐点となる時間を Break Even Time (BET) と呼ぶ．スリープモードに移行した際には BET 以上スリープしなければ逆に電力が増大してしまうため，BET を考慮した最適化が重要となる．この BET は半導体プロセスや温度，PG 対象回路の構成に依存して変化する．

2.2 走行時リーク削減のための課題

前述のオーバーヘッドがあるため，効率的に走行時リーク消費電力を削減するためには，以下の点を考慮してモード切り替え制御を行なう必要がある．

(1) WL の隠蔽:
スリープ状態からの復帰の際にあらかじめ処理が可能となる時刻を予測し，WL の時間分だけ前にスリープ状態から復帰させることで時間的なオーバーヘッドを隠蔽する必要がある．この時，saved-period の期間をなるべく長くするためには処理可能となるぎりぎりのタイミングでスリープ状態から復帰することが重要である．

(2) アイドル期間の高精度な予測:
モード切り替えオーバーヘッド削減のためには，各ユニットのアイドル期間を正確に検出/予測することで，BET よりも長いアイドルの場合のみにスリープモードに移行することが重要となる．

(3) 実行時の状況への適応性:
BET はチップ温度などに応じて変化するため，実行時の状況に動的に適応可能である必要がある．

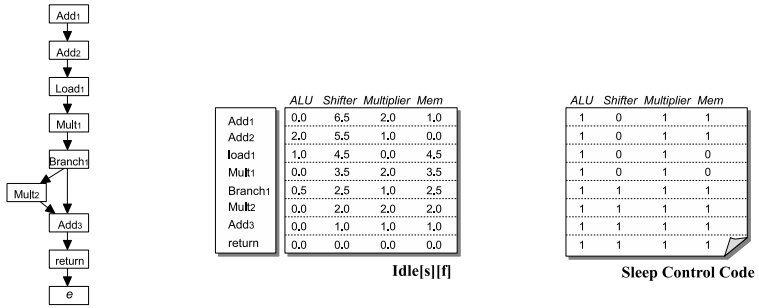
さらに，オーバーヘッドとなる消費エネルギーを低減するためには，モード切り替えを実現するためのハードウェアコストをなるべく抑える必要がある．次節では，これらの点を考慮しつつ，コンパイラとアーキテクチャで協調してモード切り替え制御を行なう手法を検討する．

3. コンパイラによるアイドル期間の予測

3.1 アイドル期間の予測

本稿で提案する細粒度 PG 手法では，コンパイラにより各演算器のアイドル期間を予測する．予測手法は，文献 8) の手法を基にしており，本節ではその概要について述べる．

アイドル期間予測のために，まずバイナリコードから手続き (procedure) 毎にコントロールフローグラフ (CFG) を作成する．図 3-(a) に CFG の例を示す．e 以外の各ノードはアセンブラでの 1 命令を，また，e は手続きからの出口を示す特殊なノードを表している．この CFG 上で，例えば Add_1 ノード時点での乗算器のアイドル期間を予測するためには，当該ノードの次の乗算命令 (図における $Mult_1$) 実行までのサイクル数を見積もる必要がある．対象のプロセッサが 1 サイクルに 1 命令を実行するインオーダ発行のプロセッサであり，Add_1 と $Mult_1$ 間の全命令が 1 サイクルで実行されると仮定すると，アイドル期間は



(a) Example CFG (b) Estimated Idle Cycle (c) Sleep Control Bits

図 3 各演算器のアイドル時間の解析

2 サイクルとなる．

一般に，プログラムコード中には分岐命令が多く存在するため，単純にストール期間を数え上げるのは難しい．そこで，文献 8) では，データフロー解析手法を応用したアイドル期間見積り手法が述べられており，本稿でもそれを用いる[⋆1]．なお，詳細については文献 8) を参照されたい．

なお，各演算器のアイドル期間は，キャッシュミスや分岐予測ミスにより影響を受ける．そこで，本稿では，プロファイリングにより各ロード命令のキャッシュミス確率と各分岐命令の予測ミス確率をプロファイリングにより求め，その情報を利用してより正確にアイドルサイクルを予測することも検討する．

3.2 スリープ制御コードの生成

各ノードにおける演算器毎のアイドル期間を見積もった後，各演算器のスリープモードをどのように制御するかを決定し，実際に制御用のコードを生成する必要がある．ここで，$Idle[s][f]$ をノード s における演算器 f のアイドル期間の予測結果とする．図 3-(b) の $Idle[s][f]$ のテーブルは，図 3-(a) の CFG における各演算器のアイドル期間予測結果を示している．さらに，$T_{src}[s][f]$ を，ノード s における演算器 f がアイドルとなった時点での予測アイドル期間と定義する．例えば，図において $T_{src}[Mult_1][Shifter]$ は 6.5 となる．

提案手法では，各命令 (ノード) に対し，演算器毎に 1 ビットを持つスリープモード制御用の情報として Sleep Control Bits (SCB) を付加する．Algorithm 1 は，$Idle[s][f]$ および $T_{src}[s][f]$ の情報を基に SCB を求めるためのアルゴリズムである．なお，$SCB[s][f]$ は

⋆1 当該文献には手続きを跨いだ解析についても述べられており，本稿ではそれも適用する．

Vol.2009-ARC-184 No.14
2009/8/4

Algorithm 1 Sleep Control Bit の生成

```
BET: Assumption of Break Even Time
WL: Assumption of Wake-up Latency

for all s do
  for all f do
    if Idle[s][f] ≤ BET + WL && T_src[s][f] ≤ BET + WL then
      SCB[s][f] ⇐ 1 /* turn-on FU f */
    else
      SCB[s][f] ⇐ 0 /* turn-off FU f */
    end if
  end for
end for
```

ノード s に対して付加する演算器 f の SCB であり，本ビットが 1 の場合は当該演算器を動作モードに，0 の場合はスリープモードに制御することを意味する．図 3-(c) は，WL=1 サイクル，BET=2 サイクルを仮定した場合に，図 3-(b) のようにアイドル期間が予測された場合の SCB の値を示している．なお，スリープ制御対象の演算器が 4 つの場合は，SCB のビット幅は 4 bit となる．

全ノードに対して SCB を求めた後，それをコード全体でパッキングすることで Sleep Control Code (SCC) を作成する．SCC はプログラムのバイナリコードに含まれ，次節で述べるように命令と一緒に対応するビット列をプロセッサ内に読み込み，モード制御を行なう．なお，コンパイル時には複数の BET のパラメータに対して Alogorith 1 を適用し，あらかじめ複数バージョンの SCC を作成しておく．実行時には適切なバージョンの SCC を用いることで，実行時の BET の変動に対処可能である．

4. アーキテクチャ支援

4.1 SCC の読み込み

アプリケーションプログラムが起動された際には，プログラムローダが命令やデータと共に，全バージョン (あるいは一部のバージョン) の SCC をメモリ上に読み出す．図 4-(a) は，バイナリ中の各領域がメモリに読み出された場合のメモリアドレス空間の例を示している．

メモリから SCC をフェッチした際の記憶領域として，図 4-(b) に示すように，命令キャッシュ内に *SCC-cache* と呼ぶ新たな拡張領域を設ける．図は，1 ラインに 4 命令 (i0〜i3) を保持できる命令キャッシュの例であり，c0〜c3 は各命令 (ノード) に対応する SCB である．キャッシュミスが生じた際には，L2 キャッシュからミスしたラインの命令を読み込むとともに，該当命令の SCB を SCC-cache にロードする．SCC は仮想アドレスでアクセスされ，命令と SCC との対応関係は *SCC-register* と呼ぶレジスタで管理する．ミスした命令の下位アドレスと，SCC-register を組み合わせることで，対応する命令の SCB のアドレスを生成することができる．

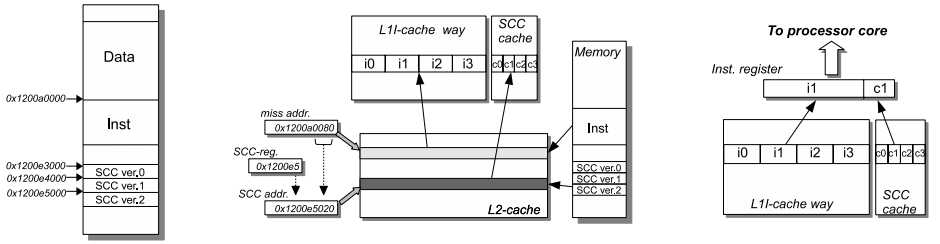


図 4 Sleep Control Code の管理

SCC-register は OS で管理することを想定している．OS がチップ温度やリーク電流センサーなどの値を基に BET を予測し，その BET を想定して生成されたバージョンの SCC が用いられるよう SCC-register に適切な値を設定する．これにより，実行時の状況へ適応することができるため，効率的にリーク消費電力が削減できると考えられる．

4.2 各演算器のモード制御

プロセッサが命令をフェッチする際には，命令とともに SCC キャッシュから当該命令の SCB をフェッチし，デコードステージで各演算器のスリープモードをコントロールするための信号を生成する (図 4-(c))．各演算器において SCB の該当するビットが 1 であれば，電力を供給し，0 であればスリープモードとする．また，デコードステージでは，当該ステージにいる命令が後段ステージで使用する演算器が WL サイクル以前からアクティブモードであったかどうかを判断する．もし，演算器が WL サイクル以前からアクティブであった場合には，すぐに演算を実行することができるが，そうでない場合には，演算器が使用可能になるまでパイプラインをストールさせる必要がある．したがって，3.1 節で述べたアイドル期間の予測が正確であれば，WL サイクル前に演算器をアクティブモードにするように SCC コードを生成できるため，WL を隠蔽することが可能である．

4.3 アーキテクチャ拡張にともなうオーバーヘッドの削減

提案するアーキテクチャでは，スリープモードの制御のために SCC を読み込む必要があるため，SCC を L2 キャッシュから読み込むまでの時間による時間的なオーバーヘッドと，SCC-cache による記憶領域 (面積) のオーバーヘッド，また SCC-cache アクセスのための電力オーバーヘッドが存在する．本節ではそれらオーバーヘッドの削減手法について述べる．

4.3.1 時間的オーバーヘッドの削減

命令キャッシュミスの際に，SCC を SCC-cache に読み込んでから命令フェッチを開始すると，L1 キャッシュミスレーテンシが長くなり，性能が低下する恐れがある．そこで，まず L2 キャッシュから命令を L1 キャッシュに転送し，SCC の読み込みを待たずに実行を再開する．この際，SCC の読み込みが完了するまでは，SCB のデフォルト値 (本稿ではすべてのビットが 0 とする) をスリープ制御に用いる．これにより，多少性能面に影響すると考えられるが，SCC を読み込むための時間的なオーバーヘッドはほぼ完全に隠蔽することができるようなる．

4.3.2 記憶領域・消費電力オーバーヘッドの削減

L1 命令キャッシュに追加して SCC-cache を設けるため，キャッシュの記憶領域が増加し，面積および電力消費面でのオーバーヘッドが生じる．これを削減するために，複数の SCB を統合することで，複数命令で一つの SCB を共有する．統合は各演算器のモード制御ビット毎に論理和演算を行なうことで実現する．これにより，モード制御ビットが 1，すなわちアクティブモードである時間が増加すると考えられるが，記憶領域・消費電力オーバーヘッドを大きく削減することが可能である．

5. 評　　価

5.1 評 価 環 境

本稿で提案するスリープモード切り替え制御方式を評価のために，SimpleScalar Tool Set[1] を用いてシミュレーションを行なった．命令セットは Alpha であり，1 命令は 32-bit 幅である．評価プログラムとしては SPEC CPU2000 ベンチマークプログラムを用い，プロファイリングには *test* インプットセット (あるいは，さらに問題サイズを縮小させた入力) を使用し，実際の評価では *ref* インプットセットを用いた．なお，最初の 10 億命令実行後の 2 億命令が評価対象である．

5.2 評価の仮定

評価では，一般的な 5 段ステージのパイプラインを持つインオーダ実行のプロセッサを仮定した．表 1 は，評価したプロセッサのパラメータである．なお，スリープ制御の対象は，表 1 中の整数加算器とロードストアユニットを除く 5 つの演算器とした．ここで，整数加算器とロードストアユニットを除いた理由は，それらの命令は頻繁に出現し，スリープモードにする機会がほとんどないためである．この場合，SCB は 5-bit あれば十分であるが，SCC-register を用いたアドレス計算を簡単化するため，余分な 3-bit を追加した 8-bit 幅とした．

表 1 評価したプロセッサの構成

Fetch & Decode width	1
Branch prediction	bimodal (2K-entry)
Mis-prediction penalty	3 cycles
Num. Functional Units	
- Int ALU	1
- Int Shift	1
- Int Mult	1
- FP Addr	1
- FP Mult	1
- FP Div	1
- (load/store)	1
L1 I-cache	32 KB, 32 B line, 2-way, 1-cycle latency
L1 D-cache	32 KB, 32 B line, 2-way, 1-cycle latency
L2 unified cache	512 KB, 64 B line, 4-way 10-cycle latency
Memory latency	80 cycles
Bus width	8 B
Bus clock	1/4 of processor core

6. 評 価 結 果

6.1 Base Result

図 5 提案手法を用いた場合における IPC (Instructions per Cycle) 低下率を示す．本評価では，BET は 10 サイクル，WL は 3 サイクルを仮定している．また，SCB の統合はせず，1 命令につき 1 つの SCB を持つ場合を評価した．図中，*Comp* および *Comp-prof* は提案手法の結果を示しており，それぞれ 3.1 節で述べたキャッシュミス，および分岐予測ミスに関するプロファイリング情報を用いなかった場合と用いた場合である．また，図には提案手法の他に，演算が終了しかつパイプライン中に当該演算器がなければすぐにスリープモードに移行する場合 (*Eager*)，さらに文献 15) で提案された動的に決定される閾値以上のサイクル数アイドル状態が続いた場合に PG モードに移行する手法 (*DSSG*) の結果も比較

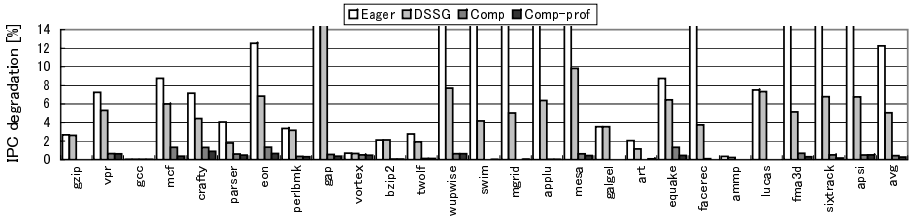


図 5 Performance degradation caused by wakeup-latency.

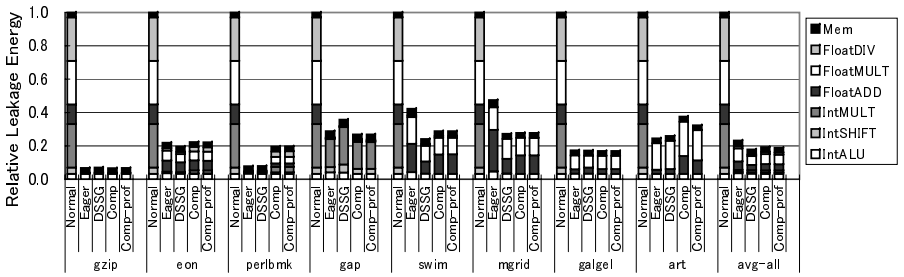


図 6 Leakage energy saving.

のために示している．

図より，Eager の性能低下が非常に大きいことがわかる．これは，演算処理を行なおうとした場合に演算器がまだスリープモード中であり，デコードステージで演算器を動作モードに移行させても，WL サイクル分パイプラインがストールしてしまうことが多いためである．DSSG では，短いアイドル期間の場合はスリープモードに移行しないように制御されるため，Eargerほど性能低下率は大きくないが，それでもベンチマーク平均で 5.0%と無視できない性能低下率となっている．一方，提案手法である Comp は性能低下率が非常に小さく，最悪の場合で 1.3%，平均で 0.43%である．これは，コンパイラにより演算器が使用されないアイドル期間を予測し，WL サイクルの時間分だけ前にスリープ状態から復帰させることで，PG にともなう性能低下を抑制できることを示している．なお，プロファイリング情報を利用した Comp-prof では，アイドル期間をより正確に予測できるため，性能低下率は平均で 0.28%と，Comp よりもさらに小さい．

図 6 は，いくつかのベンチマークにおける，PG を行なわなかった場合 (Noraml) に対しての，各手法を用いた場合の相対的なリーク消費エネルギーを示している．*avg-all* は，

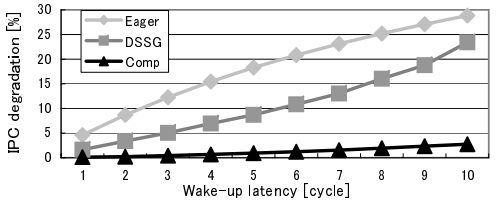


図 7 WL を変化させた場合の性能

SPEC2000 の全プログラムの平均を示している．図の相対値は演算器合計の値で計算されており，各棒グラフは演算器毎の内訳も示している．各演算器のリーク消費電力はトランジスタ数に比例するものと仮定し，各演算器のトランジスタ数は文献 3) を参考にした[⋆1]．なお，charge-latency+saved-period が BET より短い場合には net-saving サイクルがマイナスになり，かえって電力が増加するが，本評価ではその増加分も考慮している．また，実行時間増加によるリーク消費エネルギー増加の影響も含めている．

図 6 より，どの手法でも 20%程度までリーク消費エネルギーを削減することができ，走行時 PG が演算器のリーク消費エネルギー削減に有効であることがわかる．走行時 PG 手法同士を比較すると，優位性の傾向はプログラム毎に異なっている．例えば，`gap` では Comp あるいは Comp-prof が他の手法に比べて消費エネルギーが削減されているのに対し，いくつかのプログラムでは DSSG が最も削減率が高い結果となっている．また，`gzip` のように手法間でほとんど結果が同じものもある．提案手法でのアイドル期間予測の正確さや，DSSG での性能低下 (実行時間増) によるリーク消費エネルギー増加の影響などにより，結果に差がでるものと考えられる．平均すると DSSG の削減率がやや大きいが，Comp と大きな差はなく，性能低下で勝る Comp は，DSSG に比べて有効な手法であると考えられる．

6.2 WL や BET の影響

WL や BET の値が，走行時 PG 手法に与える影響を調べるために，それぞれを変化させつつ性能低下率，およびリーク消費エネルギーを評価した．図 7 は WL を 1 サイクルから 10 サイクルまで変化させた際の IPC 低下率を，図 8 および図 9 は，BET を 3 サイクルから 50 サイクルまで変化させた際の IPC 低下率および走行時 PG を行なわない場合に対す

⋆1 文献 3) には，整数乗算器と浮動小数点除算器のデータが掲載されていないため，それらの演算器のトランジスタ数は浮動小数点乗算器と同じと仮定した．

Vol.2009-ARC-184 No.14
2009/8/4

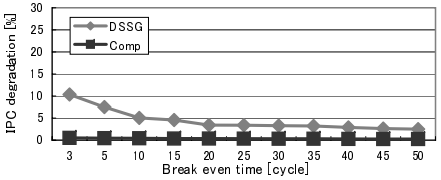


図 8 BET を変化させた場合の性能

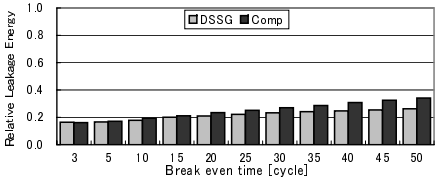


図 9 BET を変化させた場合のリーク消費エネルギー

る相対リーク消費エネルギーを示したものである[*1]．なお，すべての評価結果は，全プログラムの平均値である．

まず，WL を変化させた場合であるが，図 7 を見ると WL のサイクル数が大きくなるにつれ，Eager および DSSG では性能低下が大きくなっている．当然，WL によるストールの影響が大きくなるためである．一方，提案手法の Comp は，WL が大きくなっても，性能低下率はあまり大きくならない．コンパイラによるモード制御により，WL を隠蔽する効果が現れている結果である．

次に BET を変化させた場合について議論する．図 8 の性能低下率を見ると，DSSG では，BET の値が大きい場合は性能低下率は大きくないが，BET の値が小さくなるにつれ性能低下率が大きくなる．DSSG では，BET の値が小さい時は積極的に演算器をスリープモードに移行させようとするためである．一方，Comp では BET に関わらず，ほとんど性能低下は見られない．また，図 9 を見ると BET が大きくなるにつれ，DSSG，Comp ともにリーク消費エネルギーが多少増加している．それでも，PG を行なわない場合に比べると大きく消費エネルギーを削減できており，走行時 PG が有効であると言える．

6.3 SCC キャッシュのオーバーヘッド

本節では，4.3 節で述べた，SCC-cache の追加にともなう記憶領域・消費電力のオーバーヘッド削減のための SCB 統合の効果を評価する．

図 10 は，何命令分の SCB を統合するか (統合度，あるいは Merge-degree と表す) を変化させた場合の，IPC 低下率 (右軸) と相対リーク消費エネルギー (左軸) を示したものである．SCB を統合し，複数の命令で 1 つの SCB を共有すると，スリープモードに移行し難くなるため，統合度を増加させると性能低下率は小さくなるがリーク消費エネルギーは増加してしまう．ただし，SCB を統合することで面積オーバヘッドは大きく削減でき，統合度が 1 の場合は面積オーバーヘッドが 25% であるのに対し，統合度が 8 になると 3.1%まで削減できる．

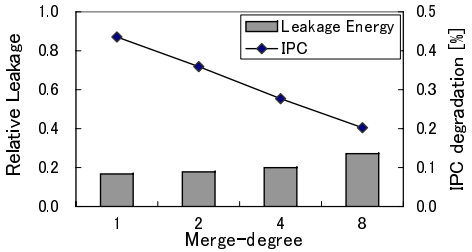


図 10 SCB 統合が IPC 低下率とリーク消費エネルギーに及ぼす影響

表 2 SCC-cache にともなう消費電力のオーバーヘッド

Merge-degree	Relative Access Energy	Relative Lekage power
1	1.837	1.577
2	1.618	1.434
4	0.873	0.840
8	1.031	1.026

表 2 は，統合度を変化させた場合の，1 命令を命令キャッシュおよび SCC-cache からフェッチする際のダイナミック消費エネルギーと，合計のリーク消費電力である．なお，値は SCC を持たない通常のプロセッサに対して正規化している．これらの値は，CACTI 5.3 を用いて評価した．

図より，統合度を増加させるとアクセスあたりの消費エネルギーや，リーク消費電力が大きく削減される傾向にある．今回の評価では，統合度を 4 以上にすることで，SCC-cache 追加による消費電力面でのオーバーヘッドはほとんど無視できるほどになることがわかった．

7. 関 連 研 究

半導体プロセス微細化によるリーク消費電力増大への対処を目的に，これまでにもリーク電流削減のための研究が多く行われている．特に，プロセッサにおいて大きな面積を占めるキャッシュにおけるリーク消費エネルギーを削減する手法は多く研究されている[5),7)]．また，演算器などのロジック部のリーク消費エネルギーも無視できないため，近年ではそれらを対

⋆1 WL が変わってもリーク消費エネルギーはあまり変化しないため，WL を変化させた際のリーク消費エネルギー評価結果は省略した．

象にした手法も多く開発されている．

文献 4) では，ドミノ回路で構成される演算器を対象にスリープモード移行の際のオーバヘッドを考慮した解析的なエネルギーモデルを作成し，さらにそのモデルに基づいたモード切り替え戦略を提案している．文献 6) は，PG によるリーク消費エネルギー削減効果の可能性をシミュレーションにより明らかにし，またステートマシンベースと分岐予測ベースのモード切り替え戦略を提案している．文献 13) では，既に存在するクロックゲーティング信号をモード切り替えのためのスリープ信号として利用する細粒度な PG 手法を提案している．また，設計時に PG の対象とするクロックゲーティング領域を判断するための解析的なモデルも提案されている．文献 12) では，細粒度な走行時 PG を MIPS3000 プロセッサに適用した LSI を設計し，リーク消費電力削減効果などを評価している．

上記のようなハードウェアベースの PG 手法に加え，コンパイラによりモードの切り替えを行なう手法も検討されている．文献 10) は，各演算器の長期間のアイドルをコンパイラにより判断し，命令によりモードの切り替えを行う手法を提案している．文献 14) では，コンパイラにより各ユニットの動作状況を見積るためのフレームワークや，スリープ命令のスケジューリングポリシーが提案されている．ループ構造のコードを対象に，組み込みプロセッサを対象としたアイドル期間の予測手法について検討している論文もある[11)]．

上記の研究は本研究と同様，ハードウェアやコンパイラにより実行時のロジック部のリーク消費エネルギーを削減することを目的としている．しかし，コンパイラとアーキテクチャで協調し，wakeup-latency の隠蔽も含めた走行時 PG 手法を検討している研究はこれまにない．この点で，本稿で提案する手法の新規性は高いと考えられる．

8. まとめと今後の課題

本稿では，走行時パワーゲーティング (PG) 手法におけるモード切り替え時のオーバーヘッドの影響を抑え，効率的に演算器部のリーク消費電力を削減することを目的に，コンパイラとアーキテクチャが連携しつつ，モード制御を行なう手法を提案した．本手法は，コンパイラにより各演算器のアイドル期間を予測し，実行時にはコンパイラから与えられた情報を利用して，ハードウェアがプロセッサ内の演算器のモードを制御するものである．1 命令発行のインオーダプロセッサを対象に提案手法を評価した結果，通常のプロセッサに比べ，0.43%程度の性能低下で約 80%の算器部のリーク消費エネルギーを削減可能であることがわかった．

今後の課題としては，提案手法をスーパースカラプロセッサへ適用して評価することや，実行時に動的に変化する BET に適応してモード制御用のコードを選択した場合の評価を行なうことなどがあげられる．

謝辞 本研究の一部は，科学技術振興機構・戦略的創造研究推進事業 (CREST) の研究プロジェクト「革新的電源制御による超低電力高性能システム LSI の研究」の支援によって行われた．

参 考 文 献

1) T. Austin, et al., “SimpleScalar: An Infrastructure for Computer System Modeling”, *IEEE Computer*, Vol. 35, No. 2, pp.59–67, Feb. 2002.
2) J.AButts and G.S.Sohi, “A static power model for architects” *In Proc. the 33rd MICRO*, pp.191–201, 2000.
3) T.M. Conte, et al. “System-level Power Consumption Modeling and Tradeoff Analysis Techniques for Superscalar Processor Design”, *In IEEE Tr. on VLSI Systems*, Vol.8, No.2, pp.129–137, Apr. 2000.
4) S.Dropsho, et al. “Managing Static Leakage Energy in Microprocessor Functional Units”, *In Proc. the 35th MICRO*, 2002.
5) K. Flautner, et al., “Drowsy caches: simple techniques for reducing leakage power” *In Proc. the 29th ISCA*, pp.148–157, 2002.
6) Z. Hu, et al., “Microarchitectural Techniques for Power Gating of Execution Units” *In Proc. ISLPED’04*, pp.32-37, 2004.
7) S.Kaxiras, et al., “Cache decay: exploiting generational behavior to reduce cache leakage power”, *In Proc. the 28th ISCA*, pp.240–251, 2001.
8) 薦田登志矢 他, “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”, *In Proc. SACSIS2009*, pp.11–18, 2009.
9) S.Mutoh, T.Douseki, Y.Matsuya, T.Aoki, S.Shigematsu, and J.Yamada, “1-V power supply high-speed digital circuit technology with multi threshold-voltage CMOS”, *IEEE J. Solid-State Circuits*, vol.30, pp.847–854, Aug. 1995.
10) S.Rele, et al. “Optimizing Static Power Dissipation by Functional Units in Superscalar Processors”, *In Proc. ICCC2002*, 2002.
11) S.Roy, et. al., “A compiler based leakage reduction technique by power-gating functional units in embedded microprocessors”, *Proc. the 20th VLSID*, pp.215–220, 2007.
12) N. Seki, et al. “A Fine Grain Dynamic Sleep Control Scheme in MIPS R3000”, *In Proc. the XXVI ICCD*, pp.612-617, Oct. 2008.
13) K.Usami and N.Ohkubo, “A Design Approach for Fine-grained Run-Time Power Gating using Locally Extracted Sleep Signals”, *In Proc. ICCD2006*, 2006.
14) Y.-P. You, C.Lee, and J.K. Lee, “Compilers for leakage power reduction”, *ACM Tr. on Design Automation and Electronic Systems*, Vol.11 No.1, pp.147–164, 2006.
15) A. Youssef, M. Anis, and M. Elmasry, “Dynamic Standby Prediction for Leakage Tolerant Microprocessor Functional Units”, In Proc. *the 39th MICRO*, pp371.–381, Dec. 2006.