社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－ARC－170（9）
2006／11／29

リーク電力削減のための細粒度命令スケジューリング手法の検討

近　藤　正　章†　　中　村　　宏†

近年，リーク電流による消費電力の増加が問題となっている．本稿では，特に実行時のロジック部におけるリーク電流削減を目的に，細粒度命令スケジューリング方式を検討する．本方式は，演算器回路がアイドルの際に，パワーゲーティング手法によるリークエネルギー削減を行うことを前提とし，その効果を最大化するために，処理を空間的・時間的に閉じ込め，処理を行う際はできるだけ一度に大量の処理を，またストール時にはできるだけ長い間ストールする，というように命令実行を制御するものである．その具体的なスケジューリング手法の一つとして，本稿では 2 つ以上のキャッシュミス要求が発生した場合には，実行可能な命令があってもロード・ストアユニット以外の演算器部をスリープモードに移行するという命令実行方式を提案する．本手法を評価した結果，従来型の実行方式に比べパワーゲーティングによりリーク電流を削減できるサイクル数が増加し，効率的リークエネルギーを削減可能であることがわかった．

Fine-Grain Instruction Scheduling for Saving Leakage-Energy

Masaaki Kondo† and Hiroshi Nakamura†

As semiconductor technology scales down, leakage-power becomes dominant in the total power consumption of LSI chips. To reduce runtime leakage-power, we propose a new instruction scheduling strategy in which a set of processing are put into temporally and spatially packed regions to maximize leakage-energy saving by a power-gating technique. We focus on stall cycles caused by cache misses in an out-of-order processor. In the proposed strategy, if two or more cache misses are generated, the processor stops the instruction execution even when some of instructions are ready to execute. We evaluate the proposed strategy and the result reveals that the proposed method increases the power-gated cycles, and thereby more leakage-energy can be saved compared with a conventional processor.

1. はじめに

近年，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となっている．モバイル計算機のバッテリ駆動時間の延長という要求はもちろんのこと，ハイエンドプロセッサにおいても，放熱の問題から消費電力削減は必要不可欠である．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よりリーク消費電力が増大し，今後はチップ全体の消費電力の半分以上を占めるとさえ予測されている．そのため，リーク電流を削減するための技術開発が早急の課題となっている．

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は多く提案されており，それらを用いることで大幅にリーク消費電力を削減することができる．しかし，将来的な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，性能低下なく実行時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[1)~3)]．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[4)] 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．

リーク電流を削減するための回路手法としては，しきい値電圧を上げる，あるいは電源電圧の供給を停止する (パワーゲーティング) などが存在するが，一般

† 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

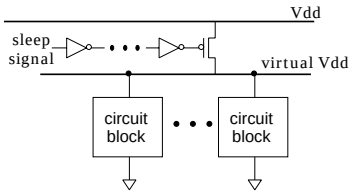

図 1 パワーゲーティングの概要

的にそれらの手法はリーク電流削減と，スイッチング速度の低下，あるいは動作や情報の保持が不可になるという間のトレードオフがある．したがって，性能低下なく実行時リーク電流を削減するためには，通常の動作を行うモード (高リークモード) と，リーク電流を削減するためのモード (低リークモード) を各ユニットの実行状況に合わせて動的に切り替えつつ実行を行う必要がある．

効率的な実行時リーク電流削減のためには，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望ましく，オーバヘッドが小さなパワーゲーティング手法が有望であるが，それでもモードの切り替え時には性能，およびエネルギー面において，ある程度のオーバヘッドが生じる．そこで本稿では，パワーゲーティング手法により効率的にリーク消費電力を削減するために考慮すべきアーキテクチャ的な課題についてまとめると共に，リーク電流削減効果を最大化するための手段の一つとして，細粒度命令スケジューリング手法について検討する．本稿で述べる細粒度命令スケジューリング手法は，処理を空間的・時間的に閉じ込め，処理を行う際はできるだけ一度に大量の処理を，またストール時にはできるだけ長い間ストールする，というように命令実行を行うものである．本稿では，そのためのアーキテクチャ上の工夫の一つについて述べ，リーク電流削減効果について評価を行う．

2. 実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の PMOS 間，あるいはグラウンド線と回路ブロックの NMOS 間に挿入する．図 1 は，PMOS 側にスリープトランジスタを挿入した場合のパワーゲーティング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電源線 (Vdd) と仮想的な電源線 (virtual Vdd) 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．

パワーゲーティング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，および virtual Vdd において減少した電荷の再充電などのために，時間的・エネルギー的なオーバヘッドが生じる．

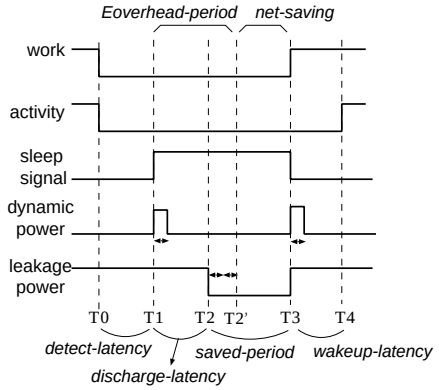

図 2 モード切り替え時のオーバヘッド

図 2 はモード切り替え時のオーバヘッドの概要を示したものである．図は時刻 T0 で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 virtual Vdd に電荷が存在するため，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，virtual Vdd への電荷の再充電などのために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* サイクル分遅延させられている．これが，パワーゲーティングを行う際の時間的なオーバヘッドとなる．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 *discharge-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．ここで，スリープモード中にもかかわらずリーク電力を削減できなかった期間 (時刻 T1 から T2') を，*Eoverhead-period* と呼ぶ．

上記で示したオーバヘッドが存在するため，比較的細粒度にモードの切り替えが可能なパワーゲーティングの場合でも，1 サイクル毎などの非常に短い時間単位でモードの切り替えを行うこと難しい．そこで，いつモードの切り替えを行うかなど，アーキテクチャ的に考慮すべき課題がいくつか存在する．

2.2 実行時リーク削減のための課題

モード切り替え時のオーバヘッドを低減しつつ，パワーゲーティングにより効率的に実行時リーク電力を削減するためには，以下の点を考慮してアーキテクチャの設計を行う必要がある．

(1) アイドルサイクル検出の効率化
(2) 一回のアイドル時間の最大化
(3) wakeup-latency の隠蔽

(1) はある決まったアイドル時間の中で detect-latency を最小化し，saved-period の最大化を図るものである．また，短いアイドルの場合にモードの切り替えを行うと，オーバヘッドによる時間的・エネルギー的な無駄が多いため，長い時間のアイドルを正確に検出することも重要である．例えば文献[5] では，通常は数サイクルアイドル状態が続いた場合に長い時間のアイドルと予測してスリープモードに移行するところを，分岐予測ミスが発生した場合は，ALU 部をすぐにスリープ状態に移行することで detect-latency を最小化する手法を提案している．

(2) は，短い時間のアイドルではスリープ状態に移行できない，あるいは仮にスリープ状態にしても，オーバヘッドにより無駄が大きいために，ストールする場合にはできるだけ長い時間ストールするように処理を時間的・空間的にある機能ブロックに閉じ込めるものである．これにより一回のアイドルの期間が長くなるためパワーゲーティングの機会が増大し，また相対的なオーバヘッドの縮小が期待できる．

(3) は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，wakeup-latency サイクル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ヘッドを隠蔽するものである．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．

上記の 3 つの課題中，(2) はこれまであまり検討されてこなかった．そこで，本稿では一回のアイドル時間を最大化するための，アーキテクチャ手法について検討する．

3. リーク電力削減のための細粒度命令スケジューリング

3.1 概　　要

3.1.1 従来型の実行方式

従来のマイクロプロセッサでは，命令レベルの並列性抽出技術，あるいはレーテンシ隠蔽技術に代表されるように，できる限り暇なユニットが存在しないように，また各命令を早期に実行可能状態にし，かつ実行可能な処理はできる限りすぐに実行されるようにアーキテクチャ的な工夫や命令スケジューリングが行われている．図 3 は従来のプロセッサの実行の様子を示したものである．図は時間の経過にともなう 2 つのユニットの稼働率を表し，ユニット間の矢印は処理の依存関係を表している．従来型の実行では，Unit-B の命令実行に依存する Unit-A の命令は，依存関係が解決次第すぐに実行される．そのため，もし Unit-A で処理すべき命令が十分にない場合，すなわち命令レベル並列度が十分でない場合,「実行」と「ストール」フェーズをを短い時間間隔で繰り返すことになる．この時，オーバヘッドを考慮すると，短い時間のストールではスリープモードに移行することができず，パワーゲーティングによりリーク電力を効率的に削減することはできない．

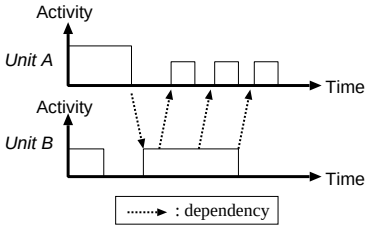

図 3 従来の実行方式

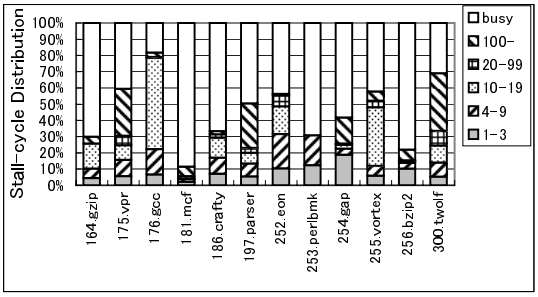

図 4 ALU 部のストールサイクル時間の分類

図 4 は，SPEC2000 整数ベンチマークの各プログラムについて，ALU のアイドルサイクル 1 回の長さ毎に，それらが全実行時間に占める割合を示している．例えば，図中の “1-3” は，1 回のアイドル時間の長さが 1，2，および 3 サイクルのものが，合計で実行時間中の何%を占めていたかを示している．また，“busy” は演算を行っていたサイクルの割合である．なお，評価環境については 4.1 節で述べる．図 4 より，20 サイクル以下のような比較的短い期間のアイドルが占める割合が多いことがわかる．これは，整数ベンチマークの場合，分岐予測ミスが多いこと，また L2 キャッシュミスは多くないが，L1 キャッシュミスが多いことが原因である．したがって，Eoverhead-period が 10

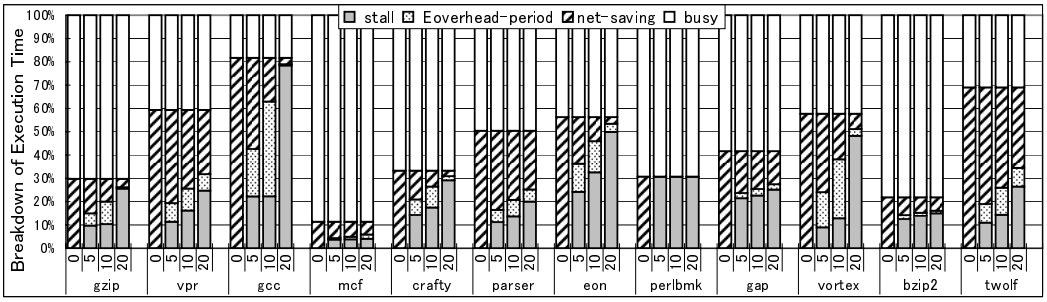

図 5 ALU 部にパワーゲーティングを適用した場合の実行時間の内訳

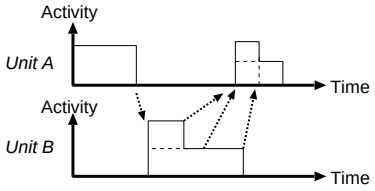

図 6 アイドル時間最大化のための実行方式

サイクル前後であると仮定すると，パワーゲーティングによるリーク電力削減効果は十部に得られない．

図 5 は，完璧なアイドルサイクルの検出が行え (detect-latency が 0 サイクルであり，Eoverhead-period 以上のストールのみをモード切り替えの対象とできる)，wakeup-latency も 0 サイクルであると仮定し，Eoverhead-period が 0，5，10，20 サイクルの 4 通りに変化させた場合の，ALU 部の実行時間の分類を示したものである．なお，実行時間は通常モード時のストール (stall)，スリープモード時のオーバヘッドサイクル (Eoverhead-period)，スリープモード時にリーク電力を削減できたサイクル (net-saving)，および演算を行っていたサイクル (busy) の 4 つに分類している．図より，Eoverhead-period が 0 であれば net-saving の部分が占める割合が大きいため，リーク電力を大きく削減できるが，Eoverhead-period が大きくなると，busy 以外のアイドルサイクルの中で net-saving の占める割合が減少し，リーク電力を効率的に削減できないことがわかる．実際の Eoverhead-period がどの程度になるかはプロセステクノロジや回路の種類・規模にも依存するため一概に言うことはできないが，5 サイクル程度でも大きく net-saving が減少してしまうものが多いことから，一回のアイドル時間が長くなるような実行方式が必要であると考えられる．

3.1.2 パワーゲーティングに適した実行方式

本稿で検討するパワーゲーティングに適した命令実行方式は，できる限り処理が時間的・空間的に集中するように制御を行う．図 6 は，図 3 の処理を時間的に閉じ込めるようにスケジューリングした場合の実行の様子を示している．この例では，依存が解決し実行の準備ができた命令もあえて実行せず，後でまとめて実行する．これにより，処理とストールのフェーズをはっきりと区別することができるようになり，より長い期間スリープモードに移行して電源供給を停止することができるため，リーク電流の大幅な削減が期待できる．また，スリープトランジスタのオン/オフの切り替え回数も少なくてすみ，モード切り替えにともなうオーバヘッドの影響も小さくなる．

次節では，このような命令実行方式の実現手法の 1 つとして，データキャッシュミスが生じた際の命令実行を制御することで，net-saving の時間を最大化する手法を提案する．

3.2 キャッシュミス時のスケジューリング

近年のマイクロプロセッサは，データキャッシュミス発生時にも依存がなく実行可能な命令を処理することで高性能化を狙うノンブロッキングキャッシュを採用するものが多い．この，ノンブロッキングキャッシュを採用するプロセッサでは，キャッシュミス解決のためのデータ転送中にも，新たにキャッシュミスが生じる可能性がある．ここで，一般的にはキャッシュ・主記憶間のデータ転送は，同時には一つのリクエストしか処理できないため，キャッシュミスによるデータ転送要求が積み重なると，プロセッサはストールする可能性が高い．

ここで，最初にキャッシュミスしたデータが下位のメモリ階層から転送されてくると，当該データに依存していた命令は実行することが可能となる．しかし，この時点では，次にキャッシュミスしたデータに依存する命令は実行できず，他に実行可能な命令がなくなると演算部は再びストールしてしまう．このように，複数のキャッシュミス要求が積み重なると，短い実行フェーズとストールフェーズが交互に表れ，効率的にパワーゲーティングを行うことができない．

そこで，キャッシュミスによるデータ転送要求が 2 つ以上積み重なった場合は，実行可能な命令がある場合でも，ALU や FPU などの演算器部をスリープ状態にすることを提案する．この際，性能低下を抑えつつ，またスリープから復帰した時にはストールせずにできる限り多くの命令実行ができるよう，ロード命令を処理するロード・ストアユニットはスリープ状態に

表 1 評価における仮定

Fetch & Decode & Commit width	4
Branch prediction	Combined bimodal (4K-entry) gshare (4K-entry) selector (4K-entry)
BTB	1024 sets, 4way
Mis-Prediction penalty	7 cycles
Instruction queue size	integer: 32, load/store: 32 floating-point 32
Issue width	integer: 2, load/store: 2 floating-point: 2
Number of ALU/FPU	ALU:2 FPU:2
L1 I-Cache	32KB, 32B line, 2way 1 cycle latency
L1 D-Cache	32KB, 32B line, 2way 2 cycle latency
L2 unified Cache	1024KB, 64B line, 8way 10 cycle latency
Memory latency	100 cycle
Bus width	8B
Bus clock	1/4 of processor core

しない．これにより，演算部では，処理とストールのフェーズをはっきりと区別することができるようになり，効率的にパワーゲーティングを行うことができると考えられる．なお，スリープ状態からの復帰は，すべてのキャッシュミス要求が解決した際に行う．

4. 評価環境

4.1 評価環境

本稿での命令実行方式によるリーク電力削減の効果を調べるため，SimpleScalar Tool Set[6] を用いたサイクルレベルシミュレーションにより評価を行なう．評価プログラムは，SPEC CPU2000 の整数ベンチマークプログラムを用いる．コンパイラは，Alpha 用の命令セットを生成する DEC C コンパイラを用い，オプションは “-arch ev6 -fast -O4 -non_shared” である．なお，SPECCPU2000 ベンチマークには *ref* インプットセットを用い，最初の 10 億命令実行後の 200 万命令を評価した．

4.2 評価の仮定

表 1 に評価におけるプロセッサの仮定を示す．なお，本評価では ALU 部のみをパワーゲーティングの対象とする．ALU 数は 2 であり，両 ALU がアイドル時のみスリープモードに移行できると仮定する．

また，実行方式の違いによる net-saving 時間の変化に着目して評価を行うため，アイドルサイクルの検出 (detect-latency) は 0 サイクルで可能，また Eoverhead-period サイクル以上のストールのみをモード切り替えの対象とでき wakeup-latency も 0 サイクルであると仮定する．なお本評価では，評価指標としてリークエネルギー自体ではなく net-saving のサイクル数と，性能 (IPC) を用いる．

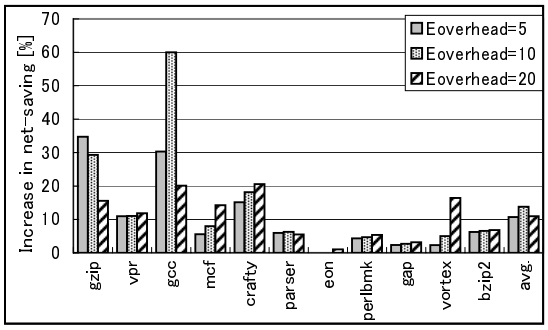

図 7 リーク電力削減可能なサイクル数の改善率

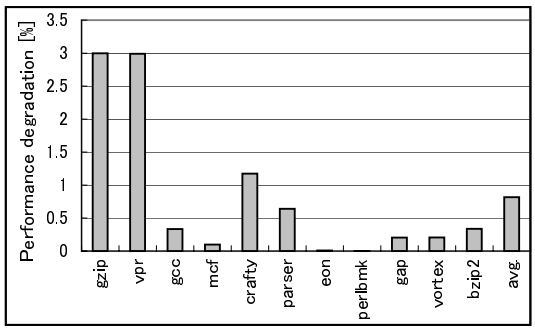

図 8 性能低下率

5. 評価結果

図 7 に，通常のプロセッサの実行方式に対し，3.2 節で述べた 2 つ以上のキャッシュミス要求が発生した場合には，実行可能な命令があっても ALU 部をスリープモードに移行するよう実行方式を変更した場合の，ALU 部のリーク電力削減可能なサイクル数 (net-saving) の増加率を示す．なお，Eoverhead-period については 5，10，20 サイクルの 3 種類の場合について評価した．

図より，多くのプログラムで net-saving のサイクル数が，通常の実行方式の場合に比べて大きく増加していることがわかる．この結果より，従来のプロセッサに比べ実行フェーズとストールフェーズがはっきり切り分けられるようになっていると考えられる．

なお，アイドル時間に占める net-saving サイクルの割合は，従来型の実行方式に比べて本提案実行方式では最高では 7%，平均で 3%増加している．net-saving サイクルの増加は，直接リークエネルギー削減に結び付くため，提案手法により ALU 部のリークエネルギーが大きく削減できると期待できる．

次に，図 7 に通常の実行方式に比べての提案方式の性能低下率を示す．図より，ほとんどのプログラムで従来型に比べ性能が低下していることがわかる．提案方式では実行可能な処理を後回しにすることがあるた

め，その命令が性能上クリティカルであった場合は，性能に悪影響を及ぼすことが原因である．しかし，最も性能低下が大きいベンチマークでも低下率は 3.0%程度であり，平均すると 0.8%と非常に小さい．複数のキャッシュミスが生じた場合は，演算部はストールし易く，多少の命令実行を後回しにしても，多くの場合であまり性能に影響しないことがわかる．

6. 関 連 研 究

半導体プロセスの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メモリ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[1)~3)]．また，演算器等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，近年では演算器などのロジック部を対象にした手法も多く開発されている[5),7)~9)]．

文献[7)] 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，さらにそのモデル基づいたモード切り替えの戦略を提案している．文献[8)] 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[5)] は，パワーゲーティング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[9)] 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，細粒度なパワーゲーティング手法を提案している．また，設計時にパワーゲーティング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．

上記の研究は本研究と同様に，実行時のロジック部のリーク消費エネルギー削減を目的としている．しかし，実行フェーズとアイドルフェーズをはっきり区別できるように実行方式を改良し，一回のアイドル時間を最大化することで効率的にパワーゲーティングを行う点については考えられていない．この点で，本稿で提案する手法での新規性が高いと考えられる．

7. まとめと今後の課題

本稿では，パワーゲーティング手法により効率的にリーク消費電力を削減することを目的とした細粒度命令スケジューリング手法について検討した．この手法は，処理を空間的・時間的に閉じ込め，処理を行う際はできるだけ一度に大量の処理を，またストール時にはできるだけ長い間ストールする，というように命令実行を行い，一回のアイドル時間を最大化することで効率的にパワーゲーティングを行うものである．

細粒度命令スケジューリング手法の一手法として，本稿では 2 つ以上のキャッシュミス要求が発生した場合には実行可能な命令があっても ALU 部をスリープモードに移行するような実行方式を評価した．評価結果より，従来型の実行方式に比べスリープモード時にリーク消費エネルギーを削減できるサイクル数が増加することがわかった．

今後は，他のスケジューリング手法を検討すると共に，ALU 部以外へパワーゲーティング手法を効率的に適用するためのアーキテクチャを考えることが課題である．また，アイドルサイクル検出の効率化や，スリープモード復帰時のレーテンシを隠蔽する手法を検討する必要もある．さらに，プロセッサ全体でダイナミックエネルギーも含めた合計の消費エネルギーを評価することも今後の課題である．

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次世代超低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) S. Kaxiras, et al., “Cache decay: exploiting generational behavior to reduce cache leakage power”, *In Proc. the 28th ISCA*, pp.240–251, 2001.
2) K. Flautner, et al., “Drowsy caches: simple techniques for reducing leakage power” *In Proc. the 29th ISCA*, pp.148–157, 2002.
3) 小宮礼子 他, “待機ラインへの参照密度に基づく低リーク・キャッシュの高性能化” *In Proc. SACSIS2006*, pp.3-12, 2006.
4) J.A Butts and G.S. Sohi, “A static power model for architects” *In Proc. the 33rd MICRO*, pp.191–201, 2000.
5) Z.Hu, et al., “Microarchitectural Techniques for Power Gating of Execution Units” *In Proc. ISLPED’04*, pp.32-37, 2004.
6) T. Austin, et al., “SimpleScalar: An Infrastructure for Computer System Modeling”, *IEEE Computer*, Vol. 35, No. 2, pp.59–67, Feb. 2002.
7) S.Dropsho, et al. “Managing Static Leakage Energy in Microprocessor Functional Units”, *In Proc. the 35th MICRO*, 2002.
8) S.Rele, et al. “Optimizing Static Power Dissipation by Functional Units in Superscalar Processors”, *In Proc. ICCC2002*, 2002.
9) K.Usami and N.Ohkubo, “A Design Approach for Fine-grained Run-Time Power Gating using Locally Extracted Sleep Signals”, *In Proc. ICCD2006*, 2006.