情報処理学会論文誌　コンピューティングシステム　Vol. 2　No. 3　83–95 (Sep. 2009)

Pipeline Blocking：走行時パワーゲーティングのための命令実行制御手法

近藤 正章[†1]　高木 紀子[†2]　中村 宏[†2,†3]

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．本論文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，Pipeline Blocking と呼ぶ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 1 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．その具体的な手法として本論文では，L1 データキャッシュミス発生の際の命令実行スケジューリング，および同時発行命令数の細粒度制御手法を提案する．本手法を評価した結果，従来手法に比べ，1.3%の性能低下で 7.3%程度パワーゲーティングにより整数演算器部のリーク電流を削減できるサイクル数を増加させることができ，効率的にリークエネルギーを削減可能であることが分かった．

Pipeline Blocking: Fine-grain Control of Instruction Execution for Run-time Power-gating

Masaaki Kondo,[†1] Noriko Takagi[†2]
and Hiroshi Nakamura[†2,†3]

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 execution control strategy called *Pipeline Blocking* in which a set of processing is put into a temporally and spatially packed region to maximize leakage-energy saving by a power-gating technique. We propose an execution control method when L1 data-cache misses occur and also propose a fine-grain issue-width control technique. We evaluate the proposed strategy and the result reveals that the proposed method reduces leakage energy by 7.3% for the integer unit with 1.3% performance degradation on average, compared with a conventional technique.

1. はじめに

近年，消費電力・消費エネルギーの削減は LSI を設計するうえでの最も重要な課題の 1 つとなっている．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．CMOS 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，将来的にはチップ全体の消費電力の半分以上を占めるという予測もある．したがって，リーク電流を削減するための技術開発が急務である．

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，性能低下なく走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[9),11)]．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 5) のリーク消費電力モデルによると，演算器などの組合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，キャッシュだけでなく，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，閾値電圧を上げる，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デメリットとの間のトレードオフがある．したがって，性能低下なく走行時リーク電力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

†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

（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する必要がある．

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替え時には性能や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．そのため，効率的に実行時リーク電力を削減するためにはモード切替えの頻度を抑制しつつ，低リークモードで動作させる時間を最大化することが重要となる．

そこで本論文では，パワーゲーティング手法によりリーク電力削減効果を最大化するためにパイプラインブロッキング（Pipline Blocking: PB）と呼ぶ細粒度命令スケジューリング手法を提案する．PB 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 1 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ように命令実行を制御するものである．本論文では，そのためのマイクロアーキテクチャの工夫として L1 データキャッシュミス発生時の命令実行スケジューリング，および同時発行命令数の細粒度制御手法を提案し，リーク電流削減効果について評価を行う．

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ヘッドが生じる．

図 2 はモード切替え時のオーバ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

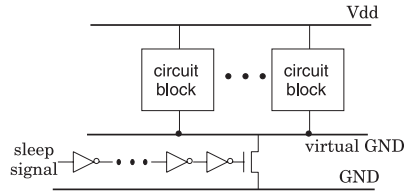

図 1　パワーゲーティングの概要
Fig. 1　Overview of power gating.

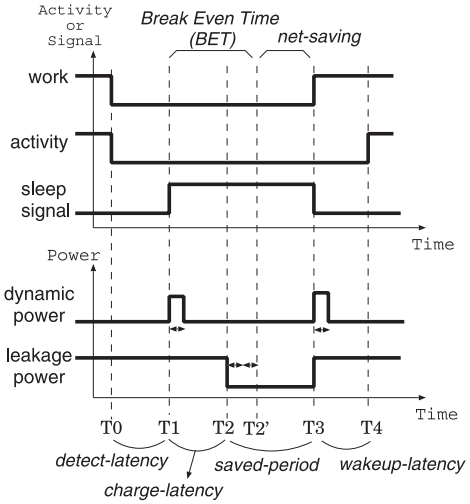

図 2　モード切替え時のオーバヘッド
Fig. 2　Overhead for mode transition.

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 に溜まった電

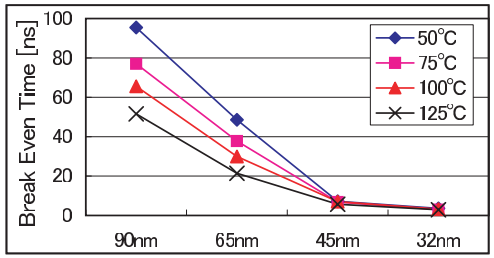

図 3　Break Even Time の評価結果
Fig. 3　Break even time of an inverter-chain circuit.

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* 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　BET の評価

前節で述べたように，PG を行う際には BET を考慮する必要がある．そこでおよその BET の値を知るために，簡単な回路を対象に予備評価を行った．**図 3** に，PG 対象の回路ブロックとして 16 段のインバータチェインを，スリープトランジスタとして 8 個の閾値電圧の高い NMOS トランジスタを用い，図 1 の構成で PG を行った場合の半導体の各プロセス世代における BET を示す．なお，スリープトランジスタの駆動のためのバッファは 1 つとした．この 8 個のスリープトランジスタおよびバッファの駆動に必要な消費電力をモード切替え時のダイナミック消費電力とする．

BET は温度や対象とする回路の構成，トランジスタの特性などのほかに，入力ベクトルにも依存するが，インバータチェインは入力値によってほとんどリーク電流が変わらず，基本データの取得に適していると考え，本論文では対象回路に用いた．また，8 個のスリープトランジスタを挿入するのは，バッファ1 つで 8 個程度のトランジスタを駆動するのが適当であり，また 16 段のインバータチェインに対して遅延の増大を引き起こさないのに十分な個数であると考えたためである．

評価には hspice を用い，各プロセスのパラメータとしては半導体プロセスの予測モデルである Predictive Technology Model（PTM）[1] を用いた．電源電圧やトランジスタの閾値電圧などは ITRS のロードマップ[2] に従い，回路ブロックは *High Performance*，スリープトランジスタは *Low Standby Power* のトランジスタで構成している．このとき，スリープトランジスタがオンの場合（動作モード）に対してのスリープトランジスタがオフ（スリープモード）時のリーク電流はプロセス世代や温度によって異なるが 2.8%から 0.62%となる．

図 3 より，プロセス世代が進むにつれて BET が短くなることが分かる．これは，将来的なプロセスではダイナミック電力に比べてリーク電力の割合が増大するため，PG の際のオーバヘッド電力とリーク電力の削減分が釣り合う時間が短くなるためである．同様の理由で対象回路の温度が高くなるにつれ BET が短くなる．将来的に BET の値は非常に短くなり，本評価環境の例では 45 nm プロセスの場合では 5〜8 ns 程度，32 nm プロセスでは 3〜4 ns 程度になる．たとえば 2 GHz の周波数を仮定した場合，45 nm プロセスでは 10〜16 サイクル，32 nm プロセスでは 6〜8 サイクルとなり，マイクロプロセッサの L1 キャッシュミスや分岐予測ミスなどによるストールサイクル中にも PG により消費エネルギーを削減できる可能性がある．

2.3　走行時リーク削減のための課題

将来的に BET が短くなるとしてもある程度のサイクル数が必要であるため，モード切替え時のオーバヘッドを低減させつつ PG 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：
モード切替えオーバヘッド削減のためには，BET よりも長い時間のアイドルを素早くかつ正確に検出/予測することで detect-latency を最小化し，saved-period を最大化する必要がある．たとえば文献 14) では，アイドル状態がある閾値以上のサイク

ル数続いた場合にスリープモードに移行するアルゴリズムにおいて，その閾値を動的に決定する手法を提案している．

(2)　1 回のアイドル時間の最大化：
短い時間のアイドルではスリープ状態に移行できない，あるいは移行してもオーバヘッドが大きく効率的でないために，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・空間的に各機能ブロックに閉じ込めることが重要となる．

(3)　wakeup-latency 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，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．このとき，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．

上記の課題中，BET を考慮しつつ効率的な PG 手法を構築するためには (1) と (2) が特に重要となる．そこで素早く比較的長いアイドルサイクルを検出し，かつ 1 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を次章で提案する．

3. パイプラインブロッキング

3.1 概　　要

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やレーテンシ隠蔽技術に代表されるように，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ことが多い．**図 4** はそれら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」フェーズを短い時間間隔で繰り返すことになる．BET 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しても電力削減効果が十分に得られないため効率的でない．

図 5 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期間別にそれらが全実行時間に占める割合を示している．たとえば図中の “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 5 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分かる．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．この場合，BET が 10 サイクル前後であると仮定すると，そのままでは PG によるリーク電力削減効果は十分に得られないと考えられる．

本論文で提案するパイプラインブロッキング（PB）は，できる限り処理が時間的・空間的にまとまるように制御を行うものである．**図 6** は，図 4 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実行をブロックすることであえて実行せず，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

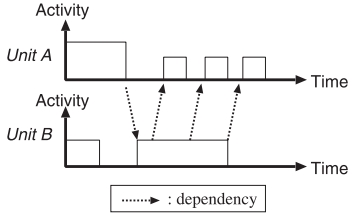

図 4　従来の実行方式
Fig. 4　Execution in a conventional pipeline.

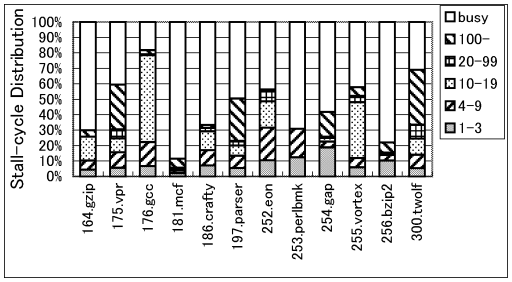

図 5　ALU 部のストールサイクル時間の分類
Fig. 5　Stall cycle distribution in an ALU.

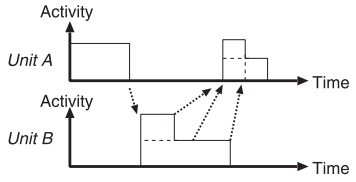

図 6　パイプラインブロッキングの実行方式
Fig. 6　Execution in Pipeline Blocking.

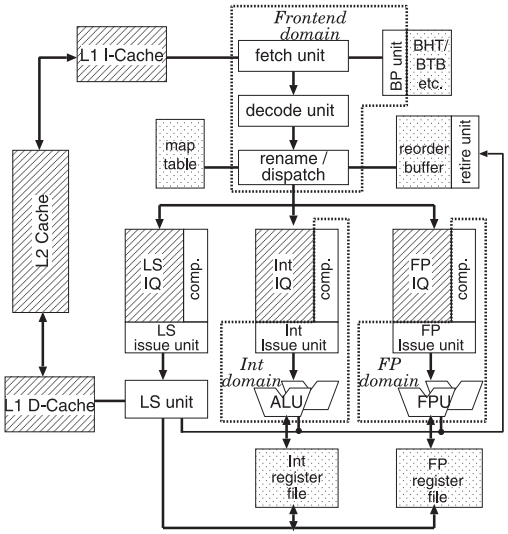

図 7　仮定するプロセッサの構成
Fig. 7　Organization of the assumed processor.

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/オフの切替え回数も少なくて済み，モード切替えにともなうオーバヘッドの影響も小さくなる．

3.3 節以降では，効率的な走行時 PG のために処理を時間的・空間的に閉じ込める具体的な PB 手法を提案する．

3.2　パワーゲーティング対象のユニット

本論文では，**図 7** に示すような，動的命令スケジューリングを行うスーパスカラプロセッサを仮定して PB を提案する．PG を行う際にはいくつかのユニットをまとめて制御の対象とし，図のようにフロントエンドドメイン（*Frontend-domain*），整数ドメイン（*Int-domain*），浮動小数点ドメイン（*FP-domain*）の 3 つのドメインを設ける．Frontend-domain における PG 対象の回路は，フェッチ（fetch unit），デコード（decode unit），リネーム/ディスパッチ（rename/dispatch），分岐予測（BP unit）を行うユニット中の組合せ回路と，それらに関連するパイプラインレジスタである．また，Int-domain および FP-domain における PG 対象回路は，命令キューにおいてウェイクアップを行う際の比較器（comp.）[⋆1]，および命令セレクトなどを含む命令発行のための組合せ回路（issue unit），各演算器，およびそれらに関連するパイプラインレジスタである．命令キューの RAM や CAM のメモリ部は情報を保持する必要があるため，PG を行わない．

なお，ロード命令はクリティカルパス上にあることが多く，ロード命令の実行をブロックすることは性能低下につながりやすいと考えられるため，本論文ではロード・ストア命令を実行するユニット（LS unit）は PG の対象にしない．また，BHT や命令キュー，キャッシュなどのメモリ構造のユニットについては，従来から不必要なエントリの電源供給を遮断するなどのリーク電力削減手法が提案されており，PB 手法とは独立して用いることができるため，本論文では扱わないこととする．

3.3　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後続のメモリアクセス命令がキャッシュにアクセスできるようにして，依存がなく実行可能な後続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．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．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 1 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比較的長い期間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現れ効率的に PG を行うことができない．

⋆1 ロード命令の実行によるウェイクアップ動作に必要な比較器は対象外とする．

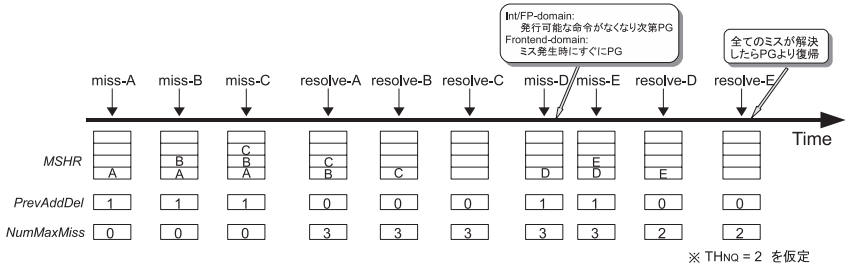

図 8　キャッシュミスの積み重なり数予測器の動作
Fig. 8　An example behavior of the cache miss accumulation predictor.

そこで，L1 データキャッシュミスによるデータ転送要求がいくつか積み重なるような場合には，Int-domain や FP-domain，さらには Frontend-domain を PG することを提案する．この際，先に述べたようにロード命令を処理するロード・ストアユニットは PG モードにしないため，Int-domain などが PG 中であっても，積み重なったキャッシュミスを処理することができ，スリープから復帰したときにはストールせずにできる限り多くの命令が実行できると期待される．

PG を行う効果があるか否かは，どの程度の期間ストールするかに依存する．そこで，キャッシュミス時に今後どの程度の期間ストールするかを判定するために，L1 データキャッシュミスが発生した際に，近い将来にいくつのキャッシュミスが積み重なるかを判定する予測器を導入する．

一般的にノンブロッキングキャッシュを採用するプロセッサは，複数のミスを管理するために未解決のキャッシュミス要求を保持する *MSHR*（*Miss State Holding Register*）[8] などのキューを持つ．本論文では，この MSHR を前提として説明する．この，MSHR に存在する有効なエントリ数が，その時点で積み重なっているキャッシュミス数となる．予測のために，まず MSHR へのエントリの追加/削除の状態を示す *PrevAddDel* と呼ぶ 1 ビットのフラグを設ける．これは，MSHR へのアクセスが追加（新規ミスの発生）であったか削除（ミスの解決）であったかを判断するものである．ここで，直前のアクセスが追加であり現在のアクセスが削除であった場合，その時点でのエントリ数が直近に積まれたミス数の最大値を示すことになる．

図 8 にキャッシュミス，およびキャッシュミスの解決が発生した際の本予測器の動作を示す．キャッシュミスが生じると，MSHR に該当ミス情報を保持するエントリが割り当てられると同時に，PrevAddDel はエントリが追加された事を示す “1” にセットされる．miss-C が発生した段階で MSHR には 3 つのキャッシュミスが積み重なっていることになる．resolve-A において最初のミスが解決すると，MSHR から miss-A のエントリが削除され，PrevAddDel が 0 に切り替わるが，その際の直前のエントリ数が直近に積まれたミス数の最大値を示していることになる．そして，この直近に積み重なったミス数の最大値を *NumMaxMiss* と呼ぶレジスタに記憶し，ミスが発生した際に参照することで，これからいくつのミスが積み重なりそうかを予測する．本予測器は，MSHR に積み重なるミス数は，前回に積み重なったミス数と同じであるという仮定に基づいている．これは，キャッシュミスとなる同一メモリアクセス命令のシーケンスが繰り返されるときにあてはまる．データキャッシュミスを引き起こす命令は極少数であるといわれており[6]，妥当な仮定であると考えられる．

MSHR が空のときに L1 データキャッシュミスが発生した際には，NumMaxMiss がある閾値（TH_{NQ}）以上の値であればそれだけの数のミスが重なる，すなわち，ある期間以上のストールが期待されると予測し PG モードに移行する．TH_{NQ} の値は，どの程度以上のストールを期待するかで決めるパラメータであり L2 キャッシュミス率が低い場合は，およそ L2 キャッシュヒットレイテンシ $\times TH_{NQ}$ が期待されるストール期間となる．ここで，TH_{NQ} の値を大きくするとスリープモードに移行する機会が減少し，小さくすると短いストール期間の場合でもスリープモードに移行してしまい，スリープ期間が BET に届かずに消費電力が増大してしまう可能性がある．そのため，期待されるストール期間が BET 以上になる範囲で最も小さな値を TH_{NQ} とするのがよいと考えられる．

Frontend-domain は，NumMaxMiss が TH_{NQ} 以上で L1 データキャッシュミスが発生した際にすぐに PG モードに移行するが，Int-domain や FP-domain においては，性能低下を極力避けるためにそれぞれの命令キューに発行可能な命令がある間は実際に PG は行わず，発行可能な命令がなくなり演算器がストールしたときに PG を行う．図 8 の例では，miss-D が発生した際には NumMaxMiss $= 3$ であり，閾値として $TH_{NQ} = 2$ を仮定すると NumMaxMiss $\geq TH_{NQ}$ が成立するため，PG モードに移行する．PG モードに移行しても，ロード・ストアユニットは PG の対象ではないため，PG 中にもミスが積み重なることが多い．

なお，前節で述べたように，本論文では命令キューの RAM などは PG を行わないと仮定しているが，もしエントリごとに個別にスリープモードの管理ができれば，有効な情報を保持していないエントリもこの時点でスリープモードに移行させることで，さらにリーク電力を削減できる可能性がある．この細粒度なモード管理については今後の課題である．

PG モードからの復帰は，最初のミスが解決し実行可能な処理が発生した時点では行わず，Int-domain，FP-domain，Frontend-domain ともに，すべてのキャッシュミス要求が解決した際に動作モードに復帰する．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，効率的に PG を行うことができると考えられる．なお，本予測器は L1 データキャッシュに対し 1 つ設けるのみである．

3.4 同時発行命令数の変更

1 サイクルに複数命令の同時実行が可能なスーパスカラプロセッサは当然複数の演算器を持つが，命令レベル並列性（ILP）の低いプログラムでは 1 サイクルにすべての演算器を使いきれないことが多い．このような場合に命令発行幅を動的に制御することで，たとえ同時に複数命令が実行可能であってもいくつかの演算器を PG モードにしてリーク電力を削減することができる．

同時発行命令数を動的に制御するために，本論文ではインターバルベースの単純なアルゴリズムを用いる．このために IPC に対する 2 つの閾値 $THiw_{high}$ と $THiw_{low}$ を導入する．あるインターバル（$Itvl_{iw}$）ごとに IPC を観測し，そのインターバルの最後で観測された IPC と閾値の値を比較する．$THiw_{low}$ よりも IPC が低い場合は次のインターバルでは命令発行幅を 1 個減らして実行する．逆に $THiw_{high}$ よりも IPC が高い場合は命令発行幅を 1 個増やして実行する．なお，プログラム実行のために，命令発行幅は最低でも 1 以上にしておく必要がある．

なお，上記の手法は動的電力削減の目的でスーパスカラのウェイ数を動的に制御する *Pipeline Balancing* [4] と近い手法であるが，リーク電力削減を目的とする点が異なる．

4. 評価環境

4.1 評価環境

本論文で提案する実行制御方式の効果を調べるため，SimpleScalar Tool Set [3] を図 7 に示すプロセッサが評価できるように拡張したものを用いて，サイクルレベルシミュレーションにより評価を行う．評価プログラムとしては，SPEC CPU2000 の整数ベンチマークプログラムを Alpha 用の命令セットを生成する DEC C コンパイラにより “-arch ev6 -fast -O4 -non_shared” のオプションでコンパイルしたものを用いる．なお，SPEC CPU2000 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 2 億命令を評価した．

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．また，アルゴリズムで用いるパラメータとしては以下の値を用いる．

表 1　評価におけるプロセッサの仮定
Table 1　Processor configuration.

Fetch & Decode & Commit width	4
Branch prediction	Combined bimodal (4 K-entry), gshare (4 K-entry) selector (4 K-entry)
BTB	1,024 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	32 KB, 32 B line, 2way, 1 cycle latency
L1 D-Cache	32 KB, 32 B line, 2way, 2 cycle latency
L2 unified Cache	2 MB, 128 B line, 8way, 10 cycle latency
Memory latency	100 cycle
Bus width	8 B
Bus clock	1/4 of processor core

- TH_{NQ}：2
- $Itvl_{iw}$：4,096 サイクル

ここで，$THiw_{high}$，$THiw_{low}$ については種々の値に変更して評価を行う．なお，BET の値は 2.2 節の結果を基に 10 サイクルを仮定する．$TH_{NQ} = 2$ としているため，演算部のストールを BET より大きい L1 miss penalty $\times\, TH_{NQ} = 10 \times 2 = 20$ サイクル程度の粒度にまとめることができると予想される．

なお，2.3 節で述べたアイドルサイクル検出の効率化と 1 回のアイドル時間の最大化に関する評価に注力するため，wakeup-latency については理想的な場合を仮定し，スリープ状態からの復帰は 0 サイクルで行えるとした．

5. 評価結果

5.1 キャッシュミス時のスケジューリングの評価結果

図 9 に，3.3 節で述べたキャッシュミス時のスケジューリング手法を用いた場合の Int-domain おける総実行サイクルに対する PG によりリーク電力を削減できるサイクル数の割合を示す．リーク電力を削減できるサイクル数は，charge-latency と saved-period を足したものから BET を引いた net-saving サイクル数の合計である．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

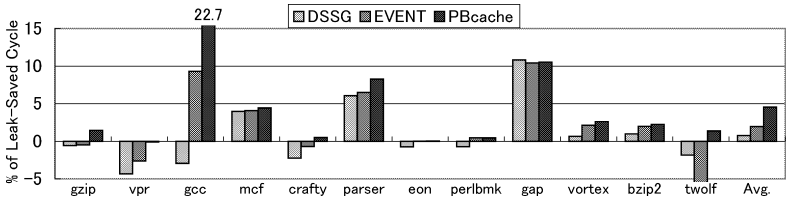

図 9　Int-domain におけるリーク電力削減可能なサイクルの割合
Fig. 9　Percentage of leakage saving cycles to the total execution time in Int-domain.

加するが，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．

図には提案手法（*PBcache*）のほかに，文献 14) 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），および提案手法と同じく NumMaxMiss が TH_{NQ} 以上の場合に PG を行うが，1 つでも L1 データキャッシュミスが解決し，実行可能な処理が発生した場合はすぐに通常モードに移行して演算処理を実行する場合（EVENT）の結果も示している．なお，DSSG の場合は複数ある演算器ごとに PG の制御ができるが，ここでは 2 つの整数演算器を合計してリーク電力をセーブできるサイクル数を示している．

図 9 より，PBcache では，ほとんどの場合においてリーク電力を削減できることが分かる．また，`mcf` や `gap` を除いて DSSG や EVENT よりもリーク電力を削減できるサイクル数が多い．これは，L1 データキャッシュミスが生じ，ミスリクエストが MSHR に多く積まれそうな場合に，実行すべき処理がなくなったらすぐに PG モードに移行することや，実行可能な処理があった場合でも通常モードに移行せずにすべてのキャッシュミスが解決するまで PG を行ったため，実行フェーズと PG を行うストールフェーズがはっきり切り分けられ，効率的に PG が行えるようになったためである．

一方，比較対象の DSSG や EVENT ではリーク電力を削減できるサイクル数がマイナスの値になっていることも多く，PG を行うことで消費エネルギーが増大してしまう可能性があることが分かる．これは，PG を行ったものの，スリープ時間が BET に届かずに通常モードに移行してしまい，PG の際の電力のオーバヘッドがリーク電力削減分のサイクル数よりも多くなってしまうためである．

図 10 は，3.3 節の実行制御手法を適用した場合の Frontend-domain の結果である．前述の評価と同様に PBcache は提案手法を，DSSG は文献 14) の手法を示している[⋆1]．

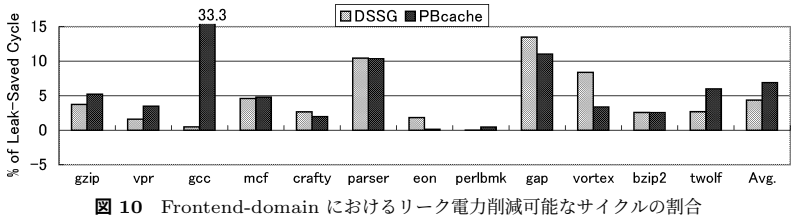

図 10　Frontend-domain におけるリーク電力削減可能なサイクルの割合
Fig. 10　Percentage of leakage saving cycles to the total execution time in Frontend-domain.

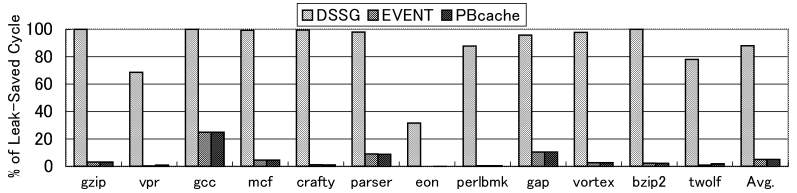

図 11　FP-domain におけるリーク電力削減可能なサイクルの割合
Fig. 11　Percentage of leakage saving cycles to the total execution time in FP-domain.

図 10 より，Int-domain の場合に比べ，DSSG ではリーク電力を削減できるサイクル数が増えていることが分かる．しかし提案手法の PBcache では実行フェーズと PG モードのフェーズがはっきり切り分けられたため，長い時間 PG できる機会が多く，DSSG よりもリーク電力を削減できることが分かる．特に `gcc` ではリーク電力削減可能なサイクルが多く，33.3%も Frontend-domain のリークエネルギーが削減できている．これは，`gcc` は L1 データキャッシュミスが多いためであり，特に本手法の効果がはっきりと現れている．

次に，**図 11** に FP-domain に各手法を適用した結果を示す．図より，FP-domain の場合は，Int-domain とは異なり，DSSG が EVENT や PBcache よりもリーク電力を多く削減できていることが分かる．また，EVENT と PBcache はほとんど同じ値となっており，提案手法の効果は現れていない．これは，評価対象としたプログラムが整数ベンチマークであり浮動小数点演算がほとんどなく，キャッシュミス時の命令実行スケジューリングを工夫しても何も効果が得られないためである．DSSG では，長いアイドル状態が続いた場

⋆1 Frontend-domain は，フェッチできる命令があっても L1 データキャッシュミスが積み重なる場合には PG を行うものであり，L1 データキャッシュミスの解決が実行可能な処理を発生させるイベントはならないため，EVENT の結果はない．

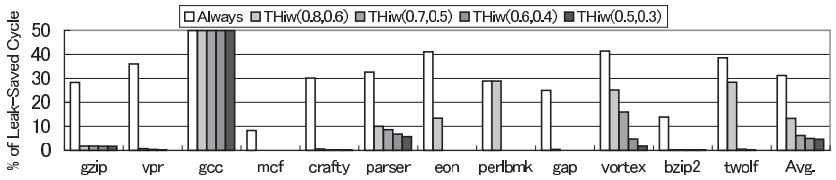

図 12　同時発行命令数を変更した場合の Int-domain のリーク電力削減可能なサイクルの割合
Fig. 12　Percentage of leakage saving cycles varying threshold for issue width scaling in Int-domain.

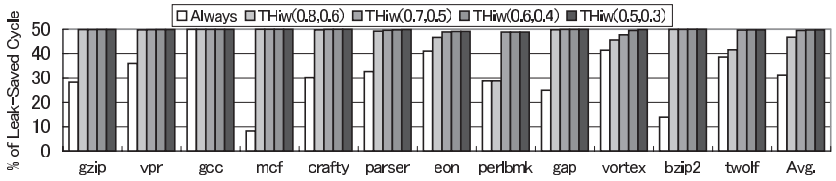

図 13　同時発行命令数を変更した場合の FP-domain のリーク電力削減可能なサイクルの割合
Fig. 13　Percentage of leakage saving cycles varying threshold for issue width scaling in FP-domain.

合，対象ユニットが次に使用されるまではスリープモードに移行した状態となるため，今回の FP-domain のようにほとんど使われないユニットに対しては効果的に働く．そのため，FP-domain に対しては，PBcache と DSSG を併用するなどしてスリープ期間を最大化する必要があると考えられる．

5.2　同時発行命令数変更の評価結果

図 12 および **図 13** に，3.4 節で述べた実行制御手法を用いた場合の Int-domain および FP-domain におけるリーク電力を削減できるサイクル数の割合を示す．本評価では 2 つの整数演算器と 2 つの浮動小数点演算器を持つプロセッサを仮定しており，各図はそれぞれの 2 つの演算器を合計した場合のリーク削減サイクル数である．図中，*Always* はつねに 1 つの演算器を PG した場合であり，$THiw(x, y)$ は提案手法において閾値 $THiw_{high}$ を x，$THiw_{low}$ を y とした場合を示している．また**図 14** は，それぞれの場合について IPC の低下率を示したものである．

まず，図 12 の Int-domain の結果を議論する．図より，同時発行命令数を変更することで長い期間演算器を PG することができるため，リーク電力削減サイクルの割合が高くなることが分かる．Always の場合はプログラム実行を通してつねに半分の演算器が PG モードであるが，実行時間が増大してしまうため，リークエネルギーが 50%にはならないことが

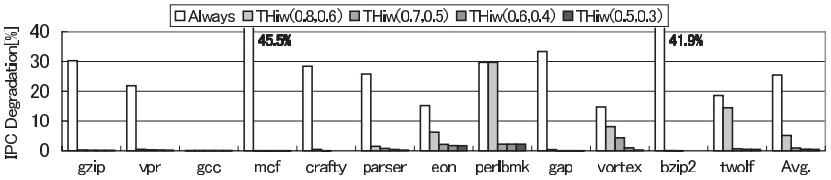

図 14　同時発行命令数を変更した場合の性能
Fig. 14　Performance degradation varying threshold for issue width scaling.

多い．提案手法の場合はリーク電力削減サイクルは閾値の値によって変化する．IPC が高いプログラムでは 2 つの演算器を使うことが多く，PG の機会がほとんどない場合が多い．一方，IPC が閾値近辺のプログラムでは閾値の値に応じてリーク削減サイクルが変化している．gcc は非常に IPC が低く，どの閾値を用いてもつねに 1 つの演算器を PG することができ，高いリーク削減サイクルを達成している．

次に，図 12 の FP-domain の結果を議論する．図より，提案手法では閾値の値にあまり影響せずに，ほぼリークエネルギーを半分程度に削減できていることが分かる．これは，浮動小数点演算命令が少ないため，つねに 1 つの演算器をスリープモードにさせておいても，性能にあまり影響しないためである．Always の場合は，Int-domain での性能低下が大きく実行時間が延びてしまう結果，提案手法ほどのリークエネルギー削減を達成できていない．

図 14 の性能を見ると，Always や比較的高い閾値を用いた提案手法の場合には性能が大きく低下するものが見受けられる．上述のように，性能低下は実行時間増大を招きリーク削減率を悪化させるばかりか，PG を行っていない他の機構のリークエネルギーが増加する恐れがあるため，なるべく避ける必要がある．したがって，性能とリーク削減効果を考えると，本評価で仮定したプロセッサ構成では，閾値が THiw(0.6,0.4) の場合にほぼ性能低下せずに，かつ良いリーク削減効果が得られているため，THiw(0.6,0.4) 程度が妥当なパラメータであると考えられる．

5.3　全手法を統合した場合のリーク削減効果

キャッシュミス時のスケジューリングの手法と同時発行命令数変更手法を同時に適用した場合における，PG を行わなかった場合に対するリーク電力を削減できるサイクル数の割合を，Int-domain，Frontend-domain，FP-domain それぞれについて**図 15**，**図 16**，**図 17** に示す．図中の *All* が提案手法を統合して用いた場合を表しており，比較のために DSSG 手法のリークエネルギー削減率も示している．また，**図 18** は PG を行わなかった場合に対

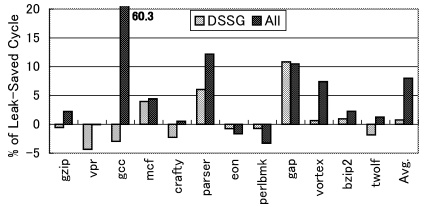

図 15　全手法を用いた場合の Int-domain のリーク削減率
Fig. 15　Leakage saving cycles with all the power-gating techniques in Int-domain.

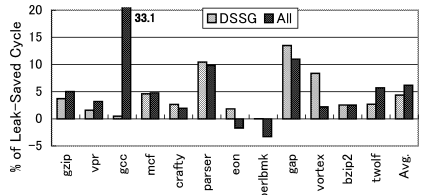

図 16　全手法を用いた場合の Frontend-domain のリーク削減率
Fig. 16　Leakage saving cycles with all the power-gating techniques in Frontend-domain.

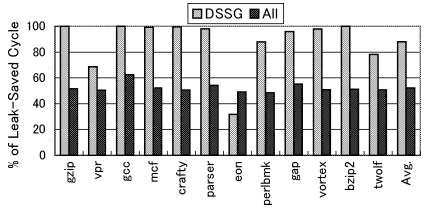

図 17　全手法を用いた場合の FP-domain のリーク削減率
Fig. 17　Leakage saving cycles with all the power-gating techniques in FP-domain.

する性能（IPC）低下率を示している[*1]．なお，リークエネルギー削減率には性能低下による実行時間の増加分の影響も含まれている．

これらの図より，提案手法を用いることでいくつかのプログラムを除いて全ドメインでリークエネルギーを削減できることが分かる．特に L1 データキャッシュミス回数の多いプログラムで効果が大きい．ただし，一部のプログラムでは提案手法によりリークエネルギーが増加している．これは，図 18 からも分かるように，提案手法を用いた場合の性能低下率が大きなプログラムであり，実行時間が延びた分のリークエネルギー増加が，PG によるリーク削減よりも大きくなってしまったためである．ただし，性能低下は平均すると 1.3%程度であり，それほど大きくない．このように性能低下を抑えられたのは，キャッシュミス時のスケジューリング手法に関しては，ミスが連続して発生し積み重なるような場合には，1 つのキャッシュミス解決によって実行可能となる命令はそれほど多くなく，すぐに実行可能な命令がなくなりストールが発生してしまうため，その間の命令が命令実行上のクリティカルパスでないためと考えられる．また，同時発行命令数の変更手法に関しては，適切な閾値を選択したことが理由である．したがって，提案手法では実行可能な処理を後回しにしたり，あるいは演算器の半分を使用しないサイクルがあるにもかかわらず，性能に及ぼす影響は小さいことが分かる．

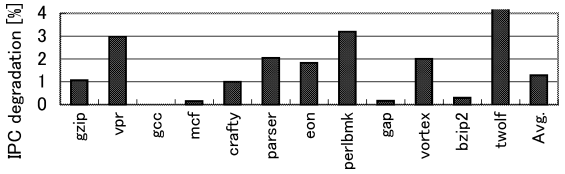

図 18　提案手法における性能低下率
Fig. 18　Performance degradation of pipeline blocking with all the power-gating techniques.

今回比較対象にした DSSG 手法では，Int-domain では多くの場合でリークエネルギーが増大している．平均すると，Int-domain では DSSG に比べ PB では 7.3%程度多くリーク電力を削減できている．この結果より，PB はアイドルサイクルが続いた場合にスリープモードに移行するという従来手法に比べて，特に Int-domain においては PG に適した手法であると考えられる．

一方，FP-domain では DSSG の方がリークエネルギーを削減できている．これは，5.1 節でも述べたように，ほとんど使われないユニットに対しては DSSG が効果的であるためである．提案手法と DSSG を併用するなど FP-domain に対して効率的にリークエネルギーを削減する手法については，今後の課題である．

5.4　BET を変化させた場合の評価結果

PB によるリーク電力の削減効果は BET に大きく依存する．そこで，本節では BET を変化させて PB および DSSG のリーク電力削減効果について評価する．**図 19**，**図 20**，**図 21**

[*1] DSSG 手法は実行すべきも処理がある場合は実行を行うため，性能は PG を行わない通常のプロセッサと同じである．

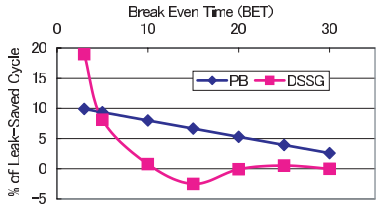

図 19　BET を変化させた場合のリーク電力を削減可能なサイクル（Int-domain）
Fig. 19　Leakage saving cycles for different BET in Int-domain.

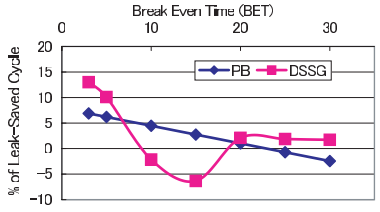

図 20　BET を変化させた場合のリーク電力を削減可能なサイクル（Frontend-domain）
Fig. 20　Leakage saving cycles for different BET in Frontend-domain.

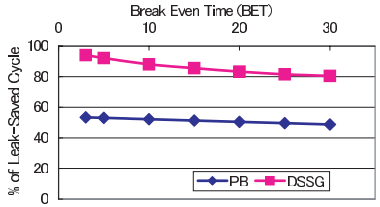

図 21　BET を変化させた場合のリーク電力を削減可能なサイクル（FP-domain）
Fig. 21　Leakage saving cycles for different BET in FP-domain.

は種々の BET の場合における，Int-domain，Frontend-domain，FP-domain の総実行サイクルに対する PG によりリーク電力を削減できるサイクル数の割合を示している．なお，PB はキャッシュミス時のスケジューリング手法と同時発行命令数変更手法の両方を用いた場合の結果である．

図より，Int-domain，Frontend-domain においては，BET が 3 サイクル程度と短い場合には PG，DSSG ともに大きくリーク電力を削減できていることが分かる．また，この場合には DSSG の方が削減効果が大きい．これは，DSSG は BET が短いときには積極的に PG を行うためである．一方，BET が Int-domain で 5 サイクル以上，Frontend-domain で 10 サイクル以上になると DSSG よりも PB の方がリーク電力削減効果が高い．FP-domain では DSSG が非常に効果的であるため，BET が変わってもつねに DSSG が良い結果となっている．

BET が長くなると，PB においても徐々にリーク電力が削減できるサイクルが減少してしまう．これは，PG をしてから，そのオーバヘッドがリーク電力削減と釣り合うまでの時間が長くなり，同じスリープ期間であっても十分にその削減効果が得られない，あるいはオーバヘッドによってリーク電力が増加してしまう場合が増えたためである．Int-domain，Frontend-domain に着目すると，PB では BET が 15 サイクル程度になった場合でも，リーク電力削減効果が得られている．一方で，DSSG では BET が 10 サイクルあたりで，リーク電力削減効果はほとんどないか，かえって増えてしまっている．このことからも，処理が時間的・空間的に集中するように実行を制御する PB が走行時 PG に有効であることが分かる．

なお，本論文では PB 手法において BET に合わせたパラメータの最適化は行っていない．たとえば，キャッシュミス時のスケジューリング手法では，BET に合わせて TH_{NQ} を調節することで，より高いリーク電力削減効果が得られると考えられる．このように，温度などの状況によって変化する BET に合わせた PB 手法の最適化については，今後の課題である．

6. 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[9),11)]．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[7),10),12),13)]．

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，さらにそのモデルに基づいたモード切替え戦略を提案している．文献 12) 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替えを行う手法を提案している．文献 10) 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替え戦略を提案している．文献 13) では，すでに存在するクロックゲーティング信号をモード切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．

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，1 回のアイドル時間を最大化することで効率的に PG を行う点については考えられていない．この点で，本論文で提案する手法の新規性は高い．

7. まとめと今後の課題

本論文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 Pipeline Blocking 手法を提案した．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 1 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ように命令実行を行い，1 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．

PB の具体的な手法として，本論文では L1 データキャッシュミス発生の際の命令実行スケジューリング，および同時発行命令数の細粒度制御手法を提案し評価した．評価結果より，従来手法に比べて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分かった．

今後は提案手法を拡張し，さらに効率的なパワーゲーティング手法を検討するとともに，他のスケジューリング手法を検討する予定である．また，スリープモード復帰時のレーテンシを隠蔽する手法や，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．また，本研究の一部は東京大学大規模集積システム設計教育研究センターを通し，シノプシス株式会社の協力により行われたものである．

参 考 文 献

1) Predictive Technology Model. http://www.eas.asu.edu/˜ptm/
2) International Technology Roadmap for Semiconductors, ITRS 2005 Edition.
3) Austin, T., Larson, E. and Ernst, D.: SimpleScalar: An Infrastructure for Computer System Modeling, *IEEE Computer*, Vol.35, No.2, pp.59–67 (2002).
4) Bahar, R.I and Manne, S.: Power and Energy Reduction via Pipeline Balancing, *Proc. 28th Intl. Symp. on Computer Architecture*, pp.218–229 (2001).
5) Butts, J.A. and Sohi, G.S.: A Static Power Model for Architects, *Proc. 33rd Intl. Symp. on Microarchitecture*, pp.191–201 (2000).
6) Collins, J.D., Tullsen, D.M., Wang, H., Lee, Y.-F., Lavery, D., Shen, J.P. and Hughes, C.: Speculative Precomputation: Long-Range Prefetching of Delinquent Loads, *Proc. 28th Intl. Symp. on Computer Architecture*, pp.14–25 (2001).
7) Dropsho, S., Kursun, V., Albonesi, D.H., Dwarkadas, S. and Friedman, E.G.: Managing Static Leakage Energy in Microprocessor Functional Units, *Proc. 35th Intl. Symp. on Microarchitecture*, pp.321–332 (2002).
8) Farkas, K.I. and Jouppi, N.P.: Complexity/Performance Tradeoffs with Non-Blocking Loads, *Proc. 21st Intl. Symp. on Computer Architecture*, pp.211–222 (1994).
9) Flautner, K., Kim, N.S., Martin, S., Blaauw, D. and Mudge, T.: Drowsy Caches: Simple Techniques for Reducing Leakage Power, *Proc. 29th Intl. Symp. on Computer Architecture*, pp.148–157 (2002).
10) Hu, Z., Buyuktosunoglu, A., Srinivasan, V., Zyuban, V., Jacobson, H. and Bose, P.: Microarchitectural Techniques for Power Gating of Execution Units, *Proc. 2004 Intl. Symp. on Low Power Electronics and Design*, pp.32–37 (2004).
11) Kaxiras, S., Hu, Z. and Martonosi, M.: Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power, *Proc. 28th Intl. Symp. on Computer Architecture*, pp.240–251 (2001).
12) Rele, S., Pande, S., Onder, S. and Gupta, R.: Optimizing Static Power Dissipation by Functional Units in Superscalar Processors, *Proc. 11th Intl. Conf. on Compiler Construction*, pp.85–100 (2002).
13) Usami, K. and Ohkubo, N.: A Design Approach for Fine-grained Run-Time Power Gating using Locally Extracted Sleep Signals, *Proc. 24th Intl. Conf. on Computer Design*, pp.155–161 (2006).
14) Youssef, A., Anis, M. and Elmasry, M.: Dynamic Standby Prediction for Leakage Tolerant Microprocessor Functional Units, *Proc. 39th Intl. Symp. on Microarchitecture*, pp.371–381 (2006).

(平成 21 年 1 月 27 日受付)
(平成 21 年 5 月 17 日採録)

近藤　正章（正会員）

1998 年筑波大学第三学群情報学類卒業．2000 年同大学大学院工学研究科博士前期課程修了．2003 年東京大学大学院工学系研究科先端学際工学専攻修了．博士（工学）．独立行政法人科学技術振興機構戦略的創造研究推進事業 CREST 研究員，2004 年東京大学先端科学技術研究センター特任助手，2007 年同特任准教授を経て，現在電気通信大学大学院情報システム学研究科准教授．計算機アーキテクチャ，ハイパフォーマンスコンピューティング，ディペンダブルコンピューティングの研究に従事．電子情報通信学会，IEEE，ACM 各会員．

高木　紀子（学生会員）

2008 年東京大学工学部計数工学科卒業．現在，同大学大学院情報理工学系研究科修士課程在学中．

中村　　宏（正会員）

1985 年東京大学工学部電子工学科卒業．1990 年同大学大学院工学系研究科電気工学専攻博士課程修了．工学博士．同年筑波大学電子・情報工学系助手．同講師，同助教授，1996 年東京大学先端科学技術研究センター助教授，2008 年より東京大学大学院情報理工学系研究科准教授．この間，1996〜1997 年カリフォルニア大学アーバイン校客員助教授．高性能・低消費電力プロセッサのアーキテクチャ，ハイパフォーマンスコンピューティング，ディペンダブルコンピューティング，ディジタルシステムの設計支援の研究に従事．情報処理学会より論文賞（平成 5 年度），山下記念研究賞（平成 6 年度），坂井記念特別賞（平成 13 年度）各受賞．IEICE，IEEE，ACM 各会員．