Vol.2011-ARC-194 No.2
2011/3/10

走行時パワーゲーティングにおけるスラック解析を用いた動的命令スケジューリングの検討

山藤友紀[†1]　近藤正章[†1]
平澤将一[†1]　本多弘樹[†1]

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．リーク電力の削減に効果的であるパワーゲーティング手法では，電源の供給・遮断のモード切り替え時にはある程度エネルギー面でのオーバヘッドが生じる．そのため，PG による走行時リーク電流をより削減するためモード切り替えの回数を抑制しつつ，電源遮断の期間を長期化することが重要となる．本稿では，命令が持つスラックを利用することで，走行時パワーゲーティングを行う際のモード切り替え回数を抑え，電源遮断の期間を長期化する手法を検討する．本稿ではまず，通常の走行時パワーゲーティング手法に対し，スラックを利用することで，モード切り替え回数を抑制しつつ，どの程度電源遮断の期間が延ばせるかを，演算器利用ログを解析することで調査する．そして，スラックを利用した命令スケジューリングによる走行時パワーゲーティング手法を提案する．本手法を初期評価したところ，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，効率的にリーク消費エネルギーを削減可能であることがわかった．

A Study on Instruction Scheduling using Slack for Run-time Power-gating

Tomoki Yamafuji,[†1] Masaaki Kondo,[†1]
Shoichi Hirasawa[†1] and Hiroki Honda[†1]

As semiconductor technology scales down, leakage-power becomes dominant in the total power consumption of LSI chips. The objective of this paper is to reduce runtime leakage-power of microprocessors' functional units by a power-gating technique. Because of the overhead energy for mode transitions in power-gating, it is important to reduce the number of mode transitions and maximize a sleep period. In this paper, we use the slack for instruction scheduling to maximize a sleep period. we first analyze a usage log of functional units to examine how instruction scheduling utilizing slacks affects the sleep periods. Then, we propose a instruction scheduling method using slack for runtime power-gating. By the preliminary evaluation, it is revealed that the the proposed method is effective for reducing leakage energy of the functional units.

1. はじめに

近年，半導体プロセスの微細化によって CMOS LSI チップ上でトランジスタのスイッチングに関係なく漏れでるリーク電流の増加が問題とされている．今後，半導体プロセスの微細化が進むことでリーク電流によって生じるリーク消費エネルギーが LSI チップ全体で消費されるエネルギーの半分を占めると予測されているためである．

従来より，モバイル用途のプロセッサにおいても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．しかし，将来的なリーク消費エネルギーの増大を考慮すると，待機時やシステムアイドル時のみならず，プロセッサのアプリケーション実行時（走行時）に性能低下を引き起こすことなくアプリケーション実行中のリーク消費エネルギーも無視することはできない．したがって，走行時にリーク電流を削減する技術の開発が急務である．

これまでにもキャッシュにおける走行時リーク電流を削減するための手法は提案されている[5]．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，トランジスタ数に比例してリーク消費エネルギーが増大するため，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エネルギーは大きいとされている．したがって，キャッシュだけでなく，演算器を含めたリーク消費エネルギー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の一つとして，電源電圧の供給を停止するパワーゲーティング（PG）手法が存在する．PG による電源遮断中はリーク電流削減が可能となるが動作や情報の保持が不可能になる．したがって，PG 手法で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（アクティブモード）と回路ブロック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ためのモード（スリープモード）を各ユニットの処理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．

PG 手法で走行時リーク電力を削減する際，モード切り替え時にダイナミック電力が消費される．そのため，スリープモードに移行してから削減できたリーク消費エネルギーとダイ

†1 電気通信大学大学院情報システム学研究科
Graduate School of Infomation Systems, The University of Electro-Communications

ナミック消費エネルギーのオーバヘッドが釣り合うまでの時間，すなわち損益分岐点となる Break Even Time（BET）を考慮しなければならない．スリープモードに移行した際には BET 以上スリープしなれば逆にエネルギーが増大してしまう．そのため，PG 手法でより走行時リーク電力を削減するためには，モード切り替えを抑制しつつ，性能低下が起こらない範囲で 1 回のスリープモードを長期化する必要がある．

走行時 PG において，モード切り替えを抑制しつつ，1 回のスリープモードを長期化するための手法として PipelinBlocking（PB）手法[9] が存在する．PB 手法は，積み重なるキャッシュミス数が閾値よりも高ければ，全てのキャッシュミスが解消されるまで実行可能な処理があったとしても PG 対象ユニットに対してスリープモードを保つ．そして，全キャッシュミスが解消された際にアクティブモードへ復帰する．これにより複数のストールを 1 つにまとめ，モード切り替えの抑制と 1 回のスリープモードを長期化するスケジューリングを実現している．

ただし従来の研究では，走行時パワーゲーティングにおけるモード切り替えの抑制と 1 回のスリープモードの長期化がどの程度行えるか解析されていない．そこで本研究では命令のスラック[3] を用いて PG によるモード切り替えの抑制とスリープモードの長期化がどの程度行えるかを演算器利用ログから解析した．ここで，スラックとはある命令の実行開始サイクルを遅らせてもプログラム実行の総サイクル数を増加させることのないサイクル数の最大値である．また本稿では，スラックを利用した命令スケジューリングを実現するアーキテクチャ手法を提案し，リーク消費エネルギーについて評価を行う．

2. パワーゲーティング手法

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技術である．この電源供給の制御のために，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，またはその両方に挿入する．**図 1** に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 手法の概要を示す．

図 1 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．ここで，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．

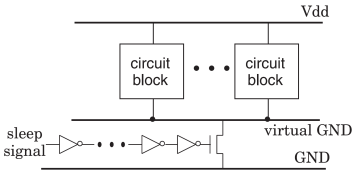

図 1 パワーゲーティングの概要

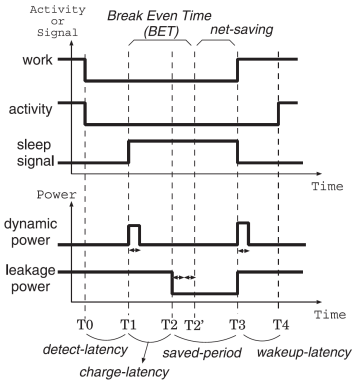

図 2 モード切り替え時のオーバヘッド

2.2 Break Even Time

図 2 は PG 手法におけるモード切り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エネルギーを示したものである．

図 2 における時刻 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 に溜ま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* 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，リーク電流が削減できる期

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エネルギーが増大してしまうため，BET を考慮したモード切り替えの最適化が重要となる．

2.3 走行時パワーゲーティング手法の課題

PG 手法において，よりリーク消費エネルギーを削減するためには，PG のモード切り替えの抑制と 1 回のアイドル期間を長期化を実現するアーキテクチャの設計を行う必要がある．

PG 手法では，短い期間のアイドルではスリープモードに移行できない，あるいは移行しても消費エネルギーのオーバヘッドが大きく効率的でない．そのため，アイドルで分断された複数の命令を連続して処理することで，アイドルの発生を減少させ複数のアイドルを 1 回のアイドルにまとめる．これにより，モード切り替え回数と charge-latency の抑制，かつ総 saved-priod の増加から PG によるリーク電力削減量の増加につながる．次章では，この課題を解決するためのスラックを用いた動的命令スケジューリングを提案する．

3. 演算器利用ログの解析による命令スケジューリングの効果

3.1 スラックを用いた動的命令スケジューリング

スラック[3] とは，プログラム実行の総サイクル数を増加させることなく，当該命令の実行レイテンシを増加させることのできるサイクル数である．本稿では，文献 3) でのスラックの定義とは多少異なり命令の処理を行う演算器の使用状況も考慮して命令の実行開始サイクルを遅らせることのできるサイクル数と定義する．

スラックは命令の依存関係と関わりがある．**図 3**，**図 4** は命令の依存関係とスラックの関連性，およびスラックを用いて命令の実行開始サイクルを遅らせる様子を表している．

図 3，図 4 における ALU-0，ALU-1，MULT は各演算器を表している．横軸はサイクル，inst は各演算器における命令の実行を表している．点線の矢印は命令間に存在する依存関係を表している．ここで図 3 を例として示すと inst1 の実行レイテンシが終了した後，inst1 と依存関係をもつ inst3/inst4 は実行を開始することが可能となる．この場合，inst1 は inst1 に依存している命令 inst3/inst4 の開始予定サイクルまでに実行を終了すれば inst3/inst4 の実行開始サイクルに遅れがでることはない．

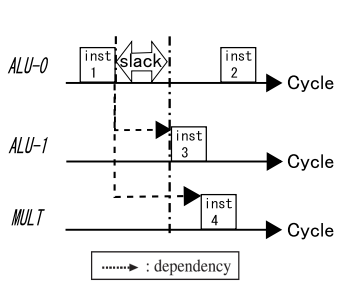

図 3 命令間の依存関係によって生じるスラック

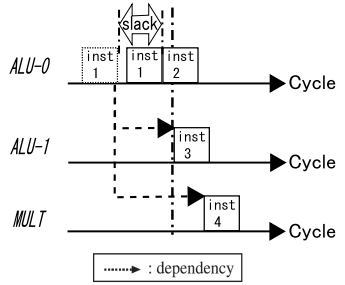

図 4 スラックを用いて命令の実行開始サイクルを遅らせる条件

そのため，inst1 の実行をスラック分遅らせることで，inst1 の前のアイドル期間を延ばすことができる．しかし，inst1 以降に再び ALU-0 でアイドルが発生してしまう場合，すなわち図 3 のように inst2 の実行開始サイクルが inst3 の実行開始サイクルよりも遅く，inst1 と inst2 のアイドルにアイドルが発生してしまう場合には，モード切り替えの回数は減らすことができない．よって，モード切り替えに伴うエネルギーオーバヘッドは削減できないことになる．したがって，ログ解析においては，図 4 のようにスラック分命令の発行を遅らせることで，同一ユニット上の命令を連続して処理可能となる場合にのみ，スラック分命令の実行を遅らせることにする．

3.2 演算器利用ログを用いた解析

スラックを用いた動的命令スケジューリングによるリーク消費エネルギーの削減効果を解析するため，SimpleScalar Tool Set[2] と，SPEC CPU2000 ベンチマークを用いて演算器利用ログ情報を取得し，命令のスラックや各演算器のアイドルを解析する．

演算器利用ログの解析を行うことで，各動的命令が保持するスラックを求めることができる．また，求めたスラックを用いて命令の実行開始サイクルを増加させることで複数の命令の実行間に存在するアイドル期間の変化や BET を考慮した場合のリーク電流の削減サイクル数について調査することができる．

表 1 は，解析に用いる演算器利用ログ情報を取得するためのプロセッサの仮定である．また本稿では，分岐予測ミスによって生じるストールは考慮しない．そのため，SimpleScalar でのシミュレーションでは分岐予測ミスは生じない設定を用いる．次節以降に解析によって得た本手法を行うことによる演算器のアイドル期間の変化とリーク消費エネルギーの理論値を示す．

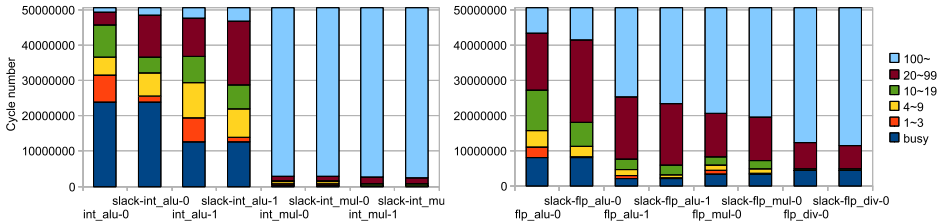

図 5 スラックによるアイドルサイクルの変化（左グラフ：整数ユニット，右グラフ：浮動小数点ユニット）

3.3 演算器毎のアイドル分布

図 5 は eon ベンチマーク実行時におけるスラック解析による各演算器のアイドル，および演算器が使用されているサイクル数を示している．左のグラフは整数ユニット，右のグラフは浮動小数点ユニットの結果である．図の縦軸はサイクル数，横軸は各演算器名を表し，*slack* と追加記述されているのはスラックを動的命令スケジューリングに用いた場合の結果を表している．例えば，int_alu-0 は 0 番発行ポートにおける整数 ALU 演算器を指す．busy は演算器が使用されているサイクル，1~3 といった数字はアイドルサイクルの期間を表している．

BET を 10 サイクルと仮定すると，図 5 より通常の命令スケジューリングと比べスラックを用いると BET 以下のアイドルサイクル数を減少させ BET 以上のアイドルサイクル数を増加させていることが分かる．そのため，PG 手法でスラックを用いたスケジューリングを行うことにより，総 saved-period の増加，および PG におけるモード切り替えのオーバヘッドの低減からリーク電流の削減効果が高まると考えられる．

表 1 演算器利用ログを取得するためのプロセッサの仮定

Fetch & Decode & Commit width	4
Branch prediction	perfect
BTB	1,024 sets, 4way
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	32KB, 32B line, 2way, 10 cycle latency
Memory latency	100 cycle
Bus width	8 B
Bus clock	1/4 of processor core

3.4 演算器毎のリーク消費エネルギーの理論値

図 6 は演算器利用ログの解析で得た演算器毎の整数・浮動小数点ベンチマークの平均リーク消費エネルギーの理論値である．図 6 の左のグラフは整数ベンチマーク，右のグラフは浮動小数点ベンチマークの平均リーク消費エネルギーの理論値を示した図である．

図における縦軸は，PG を行わないサイクル数/総実行サイクル数で求めたリーク消費エネルギーの理論値であり BET によるオーバヘッド分のサイクルも考慮している．縦軸が高いほどリーク消費エネルギーが高い．横軸は演算器名を示し，SLACK はスラックを用いた命令スケジューリング，IDLE は通常の命令スケジューリングの結果を示している．ここで，本手法におけるリーク消費エネルギーの最大削減量を示すために BET を 10 サイクルと仮定し，BET 以上のアイドル時は必ず PG を行うと仮定した．

図 6 の左のグラフを見ると整数 0 番 ALU，および整数 1 番 ALU において SLACK のリーク消費エネルギーは IDLE と比べ 9%程度低いことがわかる．また，図 6 の右のグラフより整数 0 番 ALU，整数 1 番 ALU，および浮動小数点乗算器において SLACK のリーク消費エネルギーは IDLE と比べ 4%，浮動小数点 0 番 ALU において 5%程度低いことがわかる．そのため，スラックを用いた命令スケジューリングを行うことで走行時 PG におけるリーク消費エネルギー削減可能であると考えられる．

ただし，図 6 は演算器利用ログを解析したことによって全てのアイドル，および各命令のスラックが既知，かつ BET 以上のアイドル時のみ必ず PG を行えると仮定したことによって得た結果である．しかし，ハードウェア動作時では全てのアイドル，および各命令のスラックは未知である．そのため，ハードウェア上でスラックを利用した命令スケジューリングを実現しても必ずしも図 6 の様な良い結果が得られるとは限らない．次章では，スラックの概念をプロファイル情報を用いてハードウェア動作時でも利用可能となる手法について述べる．

4. スラックを用いた動的命令スケジューリング手法

4.1 PG 制御手法

本稿では，**図 7** に示すようなスーパースカラプロセッサを仮定してスラックを用いた動的命令スケジューリング手法を提案する．なお，いつスリープモードとアクティブモードを切り替えるかの PG 制御手法は本手法とは独立であり，本研究では，Dynamic sleep signal generator（DSSG）手法[1] を採用する．DSSG 手法は，動的に変化させる閾値よりもアイ

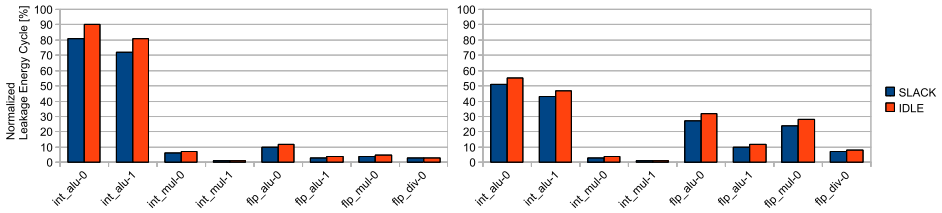

図 6 演算器におけるリーク消費エネルギーの理論値
（左グラフ：整数ベンチマーク平均，右グラフ：浮動小数点ベンチマーク平均）

ドルサイクルが長ければ PG を行う手法である．閾値は，PG 対象ユニットで生じるアイドルが BET よりも短い場合が一定回数発生した場合には高くし，BET よりも長い場合が一定回数発生した場合には低くする．これにより，DSSG 手法は PG 対象ユニットにおける BET より長いアイドルサイクルの発生を予測し PG 制御が行えることが期待される．

ここで，前章の演算器利用ログの解析ではより粒度の小さい演算器単位を考慮して命令スケジューリングを行っていた．そのため，前章の解析によるスラックを用いた命令スケジューリングで仮定する PG 対象範囲は各演算器単位であった．

しかし，この DSSG 制御を行うには一つの PG 対象ユニットに対して一つの DSSG 制御ユニットを要する．そのため，PG 制御を演算器毎で行うと多くの DSSG 制御ユニットを要するためハードウェアコストが増大してしまう．そこで本研究では，ハードウェアコスト増大による消費エネルギーの増大を避けるため PG 制御範囲を図 7 に示すように発行ポート単位とすることで DSSG 制御ユニットによるハードウェアコストを抑えつつ，粒度の小さい PG 制御を実現する．

本研究で対象としている PG 対象範囲は，図 7 のように整数 0 番発行ポートドメイン（Int issue port-0 domain），整数 1 番発行ポートドメイン（Int issue port-1 domain），浮動小数点 0 番発行ポートドメイン（Flp issue port-0 domain），浮動小数点 1 番発行ポートドメイン（Flp issue port-1 domain），である．

なお，本手法では L2 キャッシュミスが生じた際は即座にスリープモードに移行させる．

4.2 スラック表

本手法は，コンパイル時などにプロファイル実行することでスラックを解析し，解析情報を命令スケジューリングに用いる．本節では，プロファイリングから得たスラック情報をハードウェア上の動的命令スケジューリングに活用するための方法について述べる．

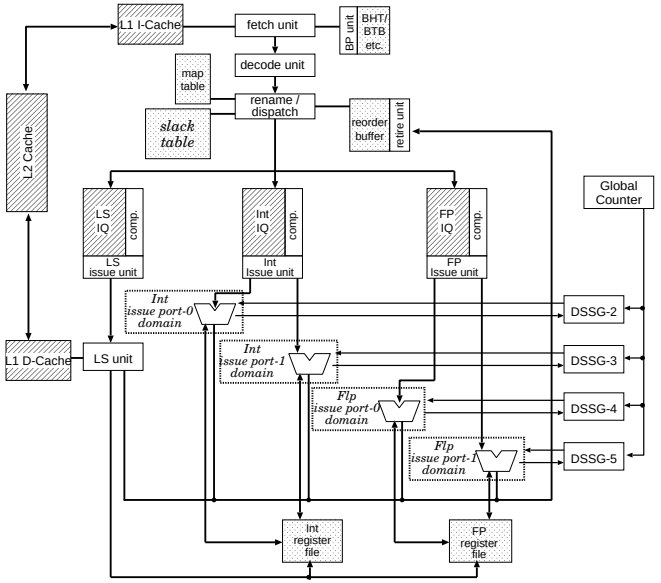

図 7 スラックを用いた動的命令スケジューリングを行うプロセッサの構成

従来，スラックを動的に予測するためのスラック予測器[10] が提案されている．しかし，このスラック予測器は複雑なハードウェアを要する．そのため，スラック予測器を実装するとハードウェアコストと共に消費エネルギーも増大してしまう可能性がある．そこで，本研究における動的命令スケジューリングに用いるスラックは，複雑なハードウェアを必要としないプロファイリングから得たスラック情報を保存するスラック表を用い，ハードウェアコストの増大を避ける．

提案するスラック表は，解析によって得た静的命令に対するスラックを，プログラムカウンタ（PC）をインデックスとして登録することで，各命令のスラック情報を簡単に与えることができる．そのため，コンパイル時のプロファイル実行等により予め演算器使用情報を取得し，解析を行うことで静的命令毎のスラックを得る必要がある．プログラムのロード時，あるいは命令キャッシュミス時に PC をインデックスとしてスラック表にスラックを登録することで，本手法であるスラックを用いた命令スケジューリングが実現可能となる．

図 8 は，図 7 のプロセッサ構成図におけるスラック表と，ディスパッチステージの概要，

および命令キュー内を示した図である．スラック表は，プログラムカウンタ（PC）をインデックスとして PC 毎のスラックが予め登録されている．このスラック表へアクセスするタイミングは命令がディスパッチステージに到達した時である．ディスパッチステージから各命令キューへ命令が発行される前にスラック表へアクセスすることでスラックを取得し，その情報を命令に付与する．命令に付与されたスラック情報は，各命令キューに格納される．

命令キューに送られた命令は，通常のアウトオブオーダ実行のスーパースカラプロセッサではソースオペランドが全て揃うことで ready フラグを 1 にして命令に発行許可を与える．本手法では，命令キュー内でスラック分命令の発行を遅らせる命令スケジューリングを実現する．そこで，ソースオペランドが全て揃っても ready フラグを 1 にしない．ソースオペランドが揃うと，スラック表から付与されたスラックが 0 でない限り毎サイクルスラックをデクリメントする．スラックが 0 となることで始めて ready フラグを 1 とする．これにより，命令キュー内でスラック分命令の発行を遅らせるスケジューリングを可能とする．

ここで，命令が発行されるタイミングや各演算器の使用状況によってスラック長は動的に変動する．静的命令の発行タイミングによってスラック長が動的に変動するスラックを動的スラックと呼ぶ．

スラック表は静的命令毎にスラックを登録するが，ここではプロファイル情報から解析した動的スラックがある程度一定である場合，その平均を求めスラック表に登録する．スラック表に登録する動的スラックの平均を平均スラックと呼ぶ．ここで，動的スラックは一定でないため動的スラックよりも平均スラックが大きくなり，その分プログラムの実行完了サイクル数が増加することが懸念される．そこで各命令の動的スラックの標準偏差を求め，標準偏差が 10 以下の場合のみ平均スラックを登録する．また，標準偏差が 10 以上であれば平均スラックは 0 とみなしスラック表には登録しない．

5. 評　　価

5.1 評 価 環 境

本研究で提案するスラックを用いた動的命令スケジューリング方式の効果を調べるため，SimpleScalar Tool Set 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 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

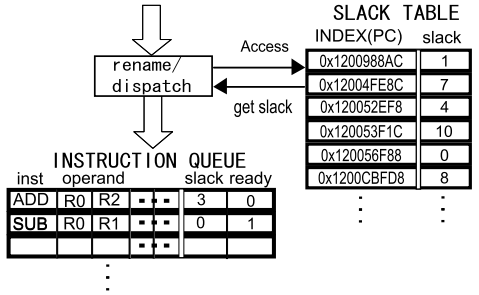

図 8　スラック表

実行後の 1 億命令を評価した．

5.2 評価の仮定

本節における評価では 3.2 節で述べた表 1 のプロセッサ構成と同じ仮定を用いる．また，本評価での BET は 10 サイクルと仮定する．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 回のアイドル期間の最大化に関する評価に注力するため，PG 制御における wakeup-latency については理想的な場合を仮定し，スリープモードからの復帰は 0 サイクルで行えるとした．

本手法で用いるスラックはコンパイル時のプロファイル実行で解析することを前提としている．しかし，スラックを利用した動的命令スケジューリングにおけるリーク消費エネルギーの最大削減効果を調査するために，今回は演算器利用ログの解析で得たスラックを利用して評価を行う．

5.3 性能の評価結果

図 9 に PG 手法を行わないプロセッサおける各ベンチマーク実行に費やされた総実行サイクル数に対する本手法および，DSSG，PipelineBlocking 手法の総実行サイクル数の比を示す．左のグラフは整数ベンチマーク，右のグラフは浮動小数点ベンチマーク実行時の結果である．グラフの横軸は実行ベンチマーク名，縦軸は PG 手法を行わないプロセッサに対する総実行サイクル数の比である．縦軸が高いほど性能が低いことを表す．また，グラフには本手法（SLACK）のほかに，DSSG 手法（DSSG）および，PipelineBlocking 手法（PB）の結果を示している．PB 手法は積み重なるキャッシュミス数を予測し，予測が閾値よりも大きければ全てのキャッシュミスが解消されるまで PG による電源供給を遮断する手法である．ここで，図 9 において，DSSG は全てのベンチマークの結果で総実行サイクル数が増

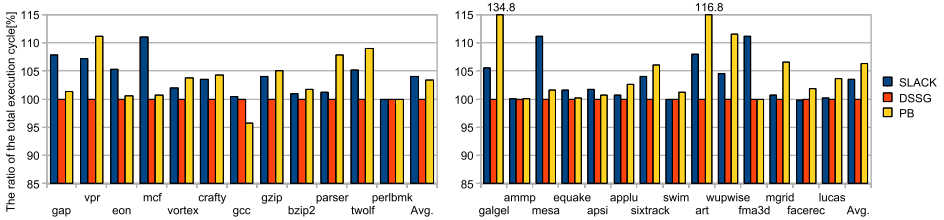

図 9 総実行サイクル数の増加率（左グラフ：整数ベンチマーク，右グラフ：浮動小数点ベンチマーク）

加していない．これは DSSG では処理すべき命令がある場合はすぐにアクティブモードにモードを切り替え命令を実行するためである．また，本研究では前述した PG 手法における wakeup-latency を 0 サイクルと仮定している．そのため，本評価環境において DSSG の総実行サイクル数は増加しない．

図 9 の左のグラフより，整数ベンチマークの結果では，SLACK の総実行サイクル数は DSSG の総実行サイクル数に対して最大で 11%（mcf），平均で 4%総実行サイクル数が増加している．対して PB の総実行サイクル数は DSSG の総実行サイクル数に対して最大で 12%（vpr），平均で 4%総実行サイクル数が増加している．よって SLACK は，PB 手法と比べ同等の性能であることが言える．

同様に図 9 の右のグラフより，浮動小数点ベンチマークの結果では，SLACK の総実行サイクル数は DSSG の総実行サイクル数と比べても総合実行サイクル数の増加を平均で 3%程度に抑えられている．

5.4 リーク消費エネルギーの評価結果

図 10 に整数ベンチマーク実行時における整数 0，1 番発行ポートドメインのリーク消費エネルギー比を示し，**図 11** に浮動小数点ベンチマーク実行時における浮動小数点 0，1 番発行ポートドメインのリーク消費エネルギー比を示している．ここで，リーク消費エネルギー比は「PG しないサイクル数/総実行サイクル数」として求めている．また，BET や総実行サイクル数のオーバヘッドも考慮している．図 10 の左のグラフは整数 0 番発行ポートドメイン，図 10 の右のグラフは整数 1 番発行ポートドメインの結果である．同様に，図 11 の左のグラフは浮動小数点 0 番発行ポートドメイン，図 11 の右のグラフは浮動小数点 1 番発行ポートドメインの結果である．

図 10 の左のグラフ（整数 0 番発行ポートドメイン）においてベンチマークによっては，SLACK は DSSG よりもリーク消費エネルギーが高い場合がある（gap，crafty，gzip，parser，

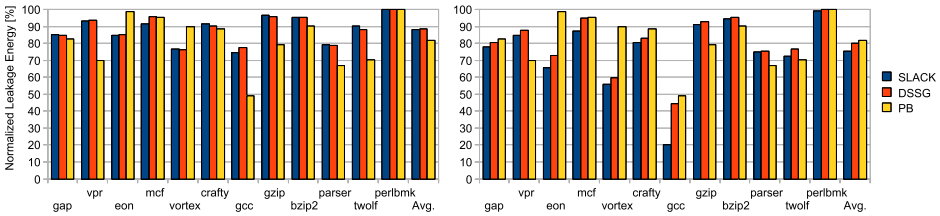

図 10 整数ベンチマークにおける整数ドメインのリーク消費エネルギー
（左グラフ：整数 0 番発行ポートドメイン，右グラフ：整数 1 番発行ポートドメイン）

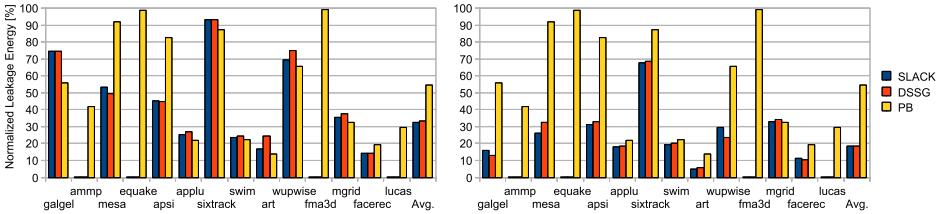

図 11 浮動小数点ベンチマークにおける浮動小数点ドメインのリーク消費エネルギー
（左グラフ：浮動小数点 0 番発行ポートドメイン，右グラフ：浮動小数点 1 番発行ポートドメイン）

twolf）これは命令発行の際，0 番発行ポートに優先的に命令が発行されることから 0 番発行ポートドメインで処理される命令の動的スラックが平均スラックよりも短い場合が多いためである．その結果，プログラムの実行に費やす総実行サイクル数が増加したことによるオーバヘッド分リーク消費エネルギーが増加している．ただし，全ベンチマークの平均（Avg.）では DSSG と同等のリーク消費エネルギーとなる．また，SLACK は PB よりもリーク消費エネルギーが高い．

ここで，図 10 の右のグラフ（整数 1 番発行ポートドメイン）においては全ベンチマークにおいて SLACK は DSSG よりもリーク消費エネルギーが低い．また，SLACK は PB と比べてもリーク消費エネルギーが低い．全ベンチマークの平均でも DSSG と比べて 5 ポイント，PB と比べて 4 ポイントリーク消費エネルギーを低い．これは，1 番発行ポートは 0 番発行ポートよりも発行優先度が低いことから平均スラックは動的スラックよりも長い場合が少なく本手法が効果的であったと言える．

図 11 の左のグラフ（浮動小数点 0 番発行ポートドメイン）のベンチマーク平均（Avg.）において，SLACK は DSSG と比べて 1 ポイントリーク消費エネルギーが低く，図 11 の右のグラフ（浮動小数点 1 番発行ポートドメイン）のベンチマーク平均（Avg.）において，

SLACK は DSSG と同等のリーク消費エネルギーであった．ただし，図 11 の両グラフ（浮動小数点 0，および 1 番発行ポートドメイン）のベンチマーク平均において，SLACK は PB よりもリーク消費エネルギーが低い．

6. 関 連 研 究

半導体プロセス微細化によるリーク消費エネルギー増大への対処を目的に，演算器部のリーク電流削減のための研究が多く開発されている[6)7)8)]．

文献 6) では，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 7) 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 8) 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．

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，PG のモード切り替え回数の抑制，および 1 回のアイドル期間の長期化がどの程度行えるかの解析や，スラックを利用することでどの程度よりリーク消費エネルギーを削減できるか評価されていない．この点で，本手法の新規性は高い．

7. まとめと今後の課題

本研究では PG 手法において，モード切り替え回数の抑制と 1 回のアイドルの長期化でよりリーク消費エネルギーを削減することを目的としたスラックを用いた動的命令スケジューリングを提案した．この手法は，命令が保持するスラック分命令の発行を遅らせることで命令を連続的に処理することで PG のモード切り替え回数を抑制し，1 回のアイドル期間を長期化することでよりリーク消費エネルギーを削減するものである．

はじめに演算器利用ログを解析することで全スラック，およびアイドルが既知である場合に BET 以上のアイドル時は PG を行うと仮定した際，通常の命令スケジューリングと比べ本手法を行うことでリーク消費エネルギーを低くすることが可能であることを理論値として示した．また，スラック，およびアイドルが未知であるハードウェア動作時にスラックを命令スケジューリングに利用できるスラック表の機構を提案した．その結果，発行優先度の低い演算器ユニットに対しては従来の PG を応用した手法よりもリーク消費エネルギーを削減できることが分かった．

今後は提案手法を拡張し，動作時にも動的にプロファイル情報を取得することで動的にスラックを求める手法や発行優先度の異なる発行ポート毎に最適なスラックを割り振る手法の提案を検討することが今後の課題である．

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」，および文部科学省科学研究費補助金 (若手研究 (B) No.21700055) の支援によって行われた．

参 考 文 献

1) Ahmed, Y., Mohab, A. and Mohamed, E.:Dynamic Standby Prediction for Leakage Tolerant Microprocessor Function Units, Proc. 39th Intl. Symp. on Microarchitecture, pp.371-384 (2006).
2) Austin, T., Larson, E. and Ernst, D.:SimpleScalar: An Infrastructure for ComputerSystem Modeling, IEEE Computer, Vol.35, No.2, pp.59-67 (2002).
3) Brian, F., Rastislav, B. and Mark D.H.:Slack: Maximizing Performance Under Technological Constraints, Proc. 29th Intl. Symp. on Computer Architecture, pp. 47-58 (2002).
4) Butts, J.A. and Sohi, G.S.:A Static Power Model for Architects, Proc. 33rd Intl.Symp. on Microarchitecture, pp.191-201 (2000).
5) Flautner, K., Kim, N.S., Martin, S., Blaauw, D. and Mudge, T.:Drowsy Caches:Simple Techniques for Reducing Leakage Power, Proc. 29th Lntl. Symp. on Computer Architecture, pp.148-157 (2002).
6) 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).
7) 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).
8) 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).
9) 近藤 正章, 高木 紀子, 中村 宏:PipelineBlocking:走行時パワーゲーティングのための命令実行制御手法, IPSJ Transactions on Advanced Computing Systems, Vol.2, No.3, pp83-95 (2009).
10) 福山 智久，福田 匡則，三輪 忍，小西 将人，五島 正裕，中島 康彦, 森 眞一郎，富田 眞治:スラック予測器を用いた省電力アーキテクチャ向け命令スケジューリング, Symp. on advanced computing systems and infrastructures, pp.123-132 (2005).