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

2008－ARC－177 (17)
2008－HPC－114 (17)
2008／3／5

走行時パワーゲーティングのための命令実行制御手法の検討

高木紀子† 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．本論文では特に演算器等のロジック部の走行時リーク電力をパワーゲーティング手法を用いて削減することを目的とした命令実行制御方式について検討する．本方式は処理を時間的・空間的に機能ブロックに閉じ込め，処理を行う際はなるべくまとめて実行し，ストール時にはなるべく長くストールするように最適化するものである．そのための時間的最適化手法としてキャッシュミス発生時の新しい命令実行制御方式と，また空間的最適化手法として IPC を観測しながら同時発行命令数を変更する方式を提案する．ALU に対して評価を行った結果，本手法は性能低下を抑えつつ，効果的に走行時リーク電力を削減できることがあることが分かった．

Instruction Execution Control for Run-Time Power-Gating

Noriko Takagi,† Kondo Masaaki† 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 execution control strategy in which a set of processing is put into temporally and spatially packed a region to maximize leakage-energy saving by a power-gating technique. We propose an execution control method when 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 effectively reduces runtime leakage-energy with little performance degradation.

1. はじめに

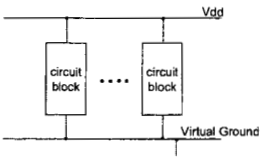
LSI を設計する上で消費電力・消費エネルギーの削減は重要な課題であるが，特に半導体プロセスの微細化により今後増大すると予測されているリーク電流による電力を削減するための技術開発は急務の課題である．従来より，特にモバイル用途のプロセッサにおいて，待機時やシステムアイドル時のリーク電流を削減するための手法が多く提案されており，それらを用いることで大幅にリーク消費電力を削減することができる．しかし，将来的なリーク電流増大を考えると，待機時やシステムアイドル時だけでなく，アプリケーション実行中のリーク消費電力も無視することはできない．したがって，性能低下なく走行時リーク電力を削減するための手法が必要となる．

これまでにも，多くのトランジスタで構成されリーク消費電力が問題となるキャッシュでは，走行時のリーク電力を削減する手法が多く提案されている[1)~3)]．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．

効率的な走行時リーク電力削減のためには，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，効率的な走行時リーク電力削減のためには，モード切り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．そこで本稿では，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の細粒

† 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



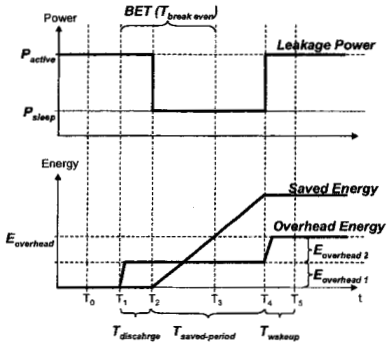


図 2 パワーゲーティングにおけるエネルギー削減

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実行制御手法は，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よう制御するものである．本稿では，そのためのアーキテクチャ上の工夫として，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，リーク電力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) 手法とは，回路ブロックと電源線，グラウンド線の間に，スリープトランジスタと呼ばれるスリープ信号で制御される閾値電圧の高いトランジスタを挿入し，回路非動作時にはこのスリープトランジスタをオフにすることで電源供給を遮断しリーク電力を削減する手法である．回路ブロックの PMOS 側・電源線間にスリープトランジスタを挿入するヘッダー型，回路ブロックの NMOS 側・グラウンド線間にスリープトランジスタを挿入するフッター型，両方に挿入するデュアル型が存在する．図 1 はフッター型の PG 構成回路を示している．図のスリープ信号がアサートされるとグラウンド線と仮想的なグラウンド線 (Virtual Ground) の間のスリープトランジスタがオフになり電源供給が遮断され，リーク電力が削減される．再度回路ブロックが使用される際にはスリープトランジスタがオンになり電源が供給される．

図 2 は縦軸に電力，横軸に時間をとり，リーク電力，

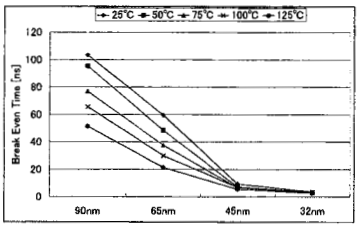


図 3 各プロセスにおける BET

オーバーヘッドエネルギー，削減されるエネルギーの時間経過を簡単に示したものである．

回路動作時には常に P_{active} のリーク電力が消費されている．時刻 $t = T_0$ で実行可能な処理がなくなり回路ブロックはアイドル状態となり，時刻 $t = T_1$ でスリープ信号をアサートすることで回路ブロックをスリープ状態にさせるとする．この切り替え時のスリープトランジスタでのダイナミック電力がエネルギーオーバーヘッド $E_{overhead1}$ となる．

時刻 T_1 の時点ではまだ Virtual Ground に電荷が流れていくためすぐにリーク電流が削減出来るわけではなく，一定期間 ($T_{discharge}$) が経過した時刻 $t = T_2$ よりリーク電流が削減可能となり，削減時のリーク電力は P_{sleep} となる．時刻 $t = T_4$ において再び実行可能な処理が発生すると，スリープトランジスタがオンになり電源の供給が再開される．この際スリープトランジスタの駆動，Virtual Ground に溜まった電荷の放電などのためにエネルギー的オーバーヘッド $E_{overhead2}$，時間的オーバーヘッド T_{wakeup} が生じ，回路ブロックでの処理が再開できるのは T_{wakeup} が経過した時刻 $t = T_5$ である．

この例で，リーク電力の削減に効果がある時間は，エネルギーオーバーヘッド ($E_{overhead}$) とスリープモードで削減されたエネルギー ($Saved\ Energy$) が等しくなる T_3 以降である．この，スリープ状態を開始してから，削減されたエネルギーとエネルギーオーバーヘッドが釣り合うまでの時間 (即ち T_1 から T_3) を Break Even Time (BET) と呼ぶ．

$$E_{overhead} = E_{overhead\,1} + E_{overhead\,2} \quad (1)$$

$$T_{break\ even} = T_3 - T_1 \quad (2)$$

$$E_{overhead} = (P_{active} - P_{sleep}) \times T_{break\ even} \quad (3)$$

スリープ状態の時間が BET より短い場合には逆に消費エネルギーが増加してしまうため，リークエネルギーを削減するためには，BET よりも長い期間アイドル状態になる場合にのみ PG を適用する必要がある．

2.2 BET の評価

前節で述べたように，PG 手法において BET の値は重要である．BET の値は半導体プロセス，温度，PG の対象回路の構成に依存するため，およその値と傾向を知るために予備評価を行った．PG 対象回路は 16 段

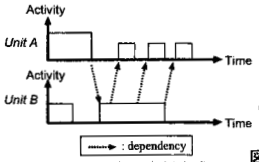


図 4 従来の実行方式

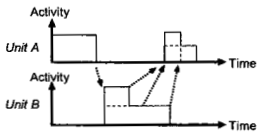


図 5 アイドル時間最大化のための実行方式

インバータとし，スリープトランジスタ として 8 個の閾値の高い NMOS トランジスタとその駆動のためのバッファを 1 個，図 1 と同じフッター型として挿入した．この構成で PG を行った場合の各半導体プロセスと BET の関係を表わしたものが図 3 である．各線は 25 ℃，50 ℃，75 ℃，100 ℃，125 ℃の各プロセス世代における BET を示している．評価には hspice を用い，各プロセスのパラメータには，Predictive Technology Model (PTM)[10] を用いた．電源電圧や，閾値電圧の値は ITPS のパラメータを使用した．

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．これはプロセスが進むとリーク電力の割合が増加するため，モード切替時のオーバーヘッドエネルギーとスリープモードで削減された電力が等しくなるまでの時間が短くなるためである．同様に回路の温度が高くなるとリーク電力が増加することから，高温時のほうが BET が短い．この結果から分かるように，45nm や 32nm などの将来的なプロセスでは BET が 5〜10ns，3〜4ns となるため，キャッシュミスや分岐予測ミスなどにより生じる短いアイドル期間でも走行時 PG を適用できる可能性がある．

2.3 走行時リーク電力削減に向けた課題

将来的に BET が短くなるとしても，BET 以下のアイドル期間ではやはりリーク電力が削減できず，また短いアイドル期間が小刻みに存在するような場合ではオーバーヘッドエネルギーが増大し効率的な PG は行えない．そこで，性能を落とさずに効率的に走行時リーク電力を削減するためには，処理を時間的・空間的に機能ブロックに閉じ込めるように最適化し，全実行時間を増大させることなく一回のアイドル時間を最大化するが重要である．次章で，上記を実行する命令実行制御方式を提案する．

3. 走行時パワーゲーティングのための命令実行制御手法

3.1 概　　要

図 4 はアウトオブオーダー実行を行うプロセッサにおける各ユニットの稼働状況を模式的に表わしたものである．矢印はユニット間の依存関係を表わす．従来のアウトオブオーダー実行では，実行可能な処理から実行することで高速化が達成されるが，実行フェーズとストールフェーズが短い時間間隔で繰り返し起こるため，効率的な PG を行うことは出来ない．

本稿で提案する命令実行制御方式は，処理を時間的・空間的に閉じ込めるように最適化を行うことで，実行とストールのフェーズをまとめることを目指す．図 5 は，図 4 の処理を時間的に最適化した場合の様子を示している．この例では，依存が解決し実行可能な命令でもあえて実行せず，後でまとめて実行することで，一回のアイドル時間を長くできている．これにより，PG によりリーク電力を削減可能なサイクルを増加させることが可能であり，また短いストール期間をまとめることで PG モードに移行する回数が少なくなり，PG のオーバーヘッドエネルギーを減少させることが可能である．

以降で，処理を時間的に最適化する手法，空間的に最適化する手法，時間的な手法と空間的な手法を統合した手法を提案する．

3.2 時間的な最適化

アウトオブオーダー実行方式ではキャッシュミス時にも，ミスに依存せず実行可能な後続命令を即座に実行できるノンブロッキングキャッシュが用いられる．ノンブロッキングキャッシュにおいて，複数のキャッシュミスが発生する状況を考える．一つ目のキャッシュミスが解決されるとそのデータに依存した命令が実行可能となる．しかし，次のキャッシュミスの解決に要する時間は長いため，その解決前に実行可能な命令がなくなり演算器は再度ストールしてしまう状況が発生する．この場合，実行フェーズとストールフェーズが繰り返され，効率的な走行時 PG は行えない．

そこで，プログラム実行中にキャッシュミスが積み重なった場合に命令発行を停止し，演算器を PG モードにし，全てのキャッシュミスが解決した時点で演算器を復帰させることで，処理とストールのフェーズをまとめるスケジューリングを提案する．$Miss$ 個のキャッシュミスが重なる時に PG する場合，スリープ期間は，$Miss \times T_{penalty}$ サイクル程度となる．$T_{penalty}$ はキャッシュミスペナルティーのサイクル数である．なお，PG モードから復帰した際に出来るだけ多くの命令を実行可能なように，ロード・ストアユニットには PG を行わない．

この命令スケジューリングを L1 データキャッシュ (以降 D1 キャッシュ) と L2 キャッシュに適用する．D1 キャッシュでキャッシュミスが $Miss$ 回積み重なると演算器を PG モードに移行するが，実行可能な命令が存在する可能性がある．そのために $Count$ サイクル演算器を動かした後に PG を行うことにする．この時スリープ期間は $Miss \times T_{D1-penalty} - Count$ サイクル程度になる．PG 期間が BET 以上になるためには，$Miss$ の値は以下の式を満たさなければならない．

$$Miss \times T_{D1-penalty} - Count \geq T_{break\ even}$$
$$\therefore Miss \geq \frac{T_{break\ even} + Count}{T_{D1-penalty}} \quad (4)$$

L2 キャッシュに対してはキャッシュミスが起こり次第 PG を開始するようにスケジューリングを行う．L2 キャッシュミスはメモリにアクセスするためミスペナ

ルティーが 100〜数 100 サイクルと長く，一つ積まれた時点で PG を行ってもリーク電力を削減可能なためである．

3.3 空間的な最適化

命令レベルの並列性 (ILP) の低いプログラムでは，演算器の使用率は低い．そこで ILP が低い場合には，複数の命令を同時発行可能であっても，同時発行命令数を減らし処理を限られた演算器に集中させ，残りの演算器を PG モードにすることでリーク電力を削減することを提案する．

同時発行命令数の変更は以下のアルゴリズムに沿って行う．ある制御周期 $ITVL$ ごとに，その期間中に演算器に発行された命令数 N_{issued_inst} をカウントして，その周期中の IPC を計算する (式 5)．

$$IPC = \frac{N_{issued\_inst}}{ITVL} \tag{5}$$

この IPC の値を，IPC に対する閾値 TH_{high}，TH_{low} と比較し，TH_{high} 以上なら次の期間の同時発行命令数を一つ増やし，IPC が TH_{low} 以下ならば命令発行幅を一つ減らす．なお命令発行幅を 0 としてしまうと，以降の処理が停止するため，同時発行命令数の下限は 1 としておく必要がある．

3.4 時間的な最適化手法と空間的な最適化手法の統合

時間的な最適化手法と空間的な最適化手法を組み合わせ，同時発行命令数変更の制御周期 $ITVL$ 中にも 3.2 節のスケジューリングを行う．

時間的最適化手法で PG すると，その間の発行命令数は 0 であるため，観測される IPC はプログラムの ILP を正しく反映しない．そこで時間的最適化によるスリープサイクル $T_{sleep\ cache}$ を $ITVL$ から引き，以下の式のように $IPC_{effective}$ を求める．この値と閾値を比較することで同時発行命令数の変更を行うものとする．なお $T_{sleep\ cache}$ が $ITVL$ と等しい場合には $IPC_{effective}$ は 0 とする．

$$IPC_{effective} = \frac{N_{issued\_inst}}{ITVL - T_{sleep\ cache}} \tag{6}$$

4. 評価環境

4.1 評価環境

本論文で提案する命令実行制御の効果を調べるため，手法を ALU 部に適用し SimpleScalar Tool Set[6] を用いてサイクルレベルのシミュレーションにより評価を行う．評価プログラムには，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

本手法で用いる各パラメータは以下の通りである．

- $Miss$: 1〜4

表 1 評価における仮定

Fetch, Decode, Commit	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	2048KB, 128B line, 8way 10 cycle latency
Memory latency	100 cycle
Bus width	8B
Bus clock	1/4 of processor core

- $Count$: 5
- $ITVL$: 1024 サイクル

TH_{high} および TH_{low} の値はいくつかの値に変更して評価を行う．プロセッサは表 1 の構成のものを仮定する．評価の指標の一つとして性能低下率を用るが，これは提案手法非適用の場合に対する IPC の低下率とする．プロセッサは表 1 の構成のものを仮定する．

5. 評価結果

5.1 時間的な最適化手法の評価結果

図 6 に $Miss$ を変化させた場合の性能低下率を示す．$Miss$ 1 では最高で約 7% の性能低下となったが，$Miss$ の値を大きくするにつれて性能低下が小さくなっているのが分かる．$Miss$ 3 以上では `twolf` を除き性能低下が 1% 未満におさまっている．これは，PG により実行開始が遅くなる命令数が減少するからである．

図 7 に，時間的な最適化を行った場合の，BET が 5, 10, 15, 20 サイクルにおけるリークエネルギー削減率を示す．このリーク削減率は，性能低下による実行時間の増加分の影響も含んだものである．提案手法において $Miss$ を 1〜4 としたグラフに加えて，文献[11] で提案されている dynamic sleep signal generator (DSSG) を用いた場合の結果も示している．DSSG とは動的に決められた閾値よりも長くアイドル期間が続いた場合に PG モードに移行する手法である．DSSG は実行すべき処理がある場合はスリープ状態から復帰し実行するため性能低下は起こらないものとしている．

図 7 において，BET が 15, 20 サイクルの場合に，$Miss$ の値が小さいとリークエネルギー削減率がマイナスになることがある．これは PG を行ったがスリープ期間が BET に届かず，オーバーヘッドエネルギーが削減分を上回ってしまったためと，性能低下による実

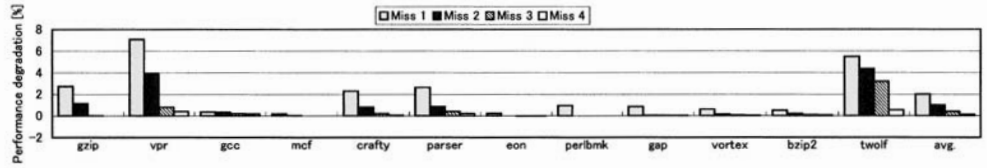


図 6　時間的最適化手法を用いた場合の性能低下率

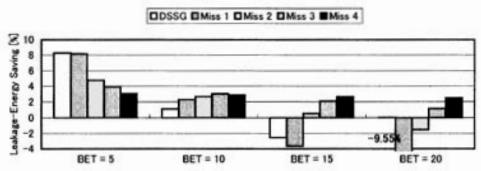


図 7　時間的最適化手法を用いた場合のリークエネルギー削減率

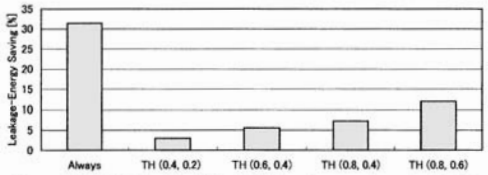


図 8　空間的最適化手法を用いた場合のリークエネルギー削減率

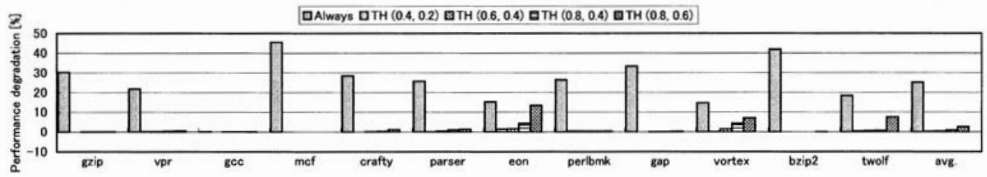


図 9　空間的最適化手法を用いた場合の性能低下率

行時間の増加分だけリークエネルギーが増加したためである．DSSG では BET が 15，20 サイクルではリークエネルギーが削減できないが，提案手法では BET に対応して異なる $Miss$ の値を用いることでリークエネルギーを削減できることが分かる．

5.2　空間的な最適化手法の評価結果

本評価は ALU が二つのプロセッサを仮定しており，図 8 に TH_{high}，TH_{low} の値を変えた場合のリークエネルギー削減率，図 9 に性能低下率を示す．$Always$ は，常に一つの ALU を PG した場合，図中の TH $(x,\ y)$ は閾値を $(TH_{high},\ TH_{low}) = (x,\ y)$ とした場合の結果である．

図 9 より $Always$ では `gcc` を除く全てのプログラムで大きな性能低下が見られ，TH (0.8, 0.4)，TH (0.8, 0.6) の場合にも大きく性能が低下するプログラムが存在する．性能が低下すると動作モードの期間が増加し，ALU 自体のリークエネルギーを増加させるだけでなく，他の機構のリークエネルギーも増加させてしまうことから，性能低下はなるべく小さく抑える必要がある．しかし，図 8 から分かるように，閾値の値によりリークエネルギー削減率が異なる．$Always$ ではリークエネルギーは 30% 以上削減可能だが，TH (0.4, 0.2) では 3% ほどしか削減できない．性能低下率とリークエネルギー削減率の両方から考えて，本評価では TH (0.6, 0.4) が妥当な値であると考えられる．

5.3　統合した手法の評価結果

閾値を $(TH_{high}, TH_{low}) = (0.6, 0.4)$ とした空間的最適化に $Miss$ を 1〜4 とした時間適最適化を行った場合の結果を示す．図 10 は性能低下率，図 11 は性能低下により引き起こされる実行時間の増加を考慮した ALU 部全体のリークエネルギー削減率を表わす．図 10 より，`vpr` や `twolf` の性能低下が大きいが，他のプログラムでは 3% 以下であることが分かる．$Miss$ の値が大きくなるにつれて性能低下は小さくなり，$Miss$ 4 ではすべてのベンチマークで 1% 以下である．

図 11 より，提案手法では $Miss$ の値を変えることで，どの BET の場合でもリークエネルギーを削減可能であり，$Miss$ の値を大きくするにつれてリークエネルギー削減率が大きくなっていることが分かる．これは，$Miss$ の値が小さい方が命令スケジューリングによりスリープする回数が多く，統合した手法においてスリープ期間を $ITVL$ から引いたことで $IPC_{effective}$ が上昇し TH_{high} を超えるようになったためである．そのため本評価では $Miss$ 4 の削減率が大きかったが，閾値の値を変えることにより，他の $Miss$ の値の場合も削減率が増加すると考えられる．

手法を統合した結果，すべての場合で時間的最適化のみの場合の削減率を上回り，$Miss$ 1, 2 の一部を除き，空間的最適化のみの場合の削減率を上回った．時間的最適化のみでは ILP が低い場合でも片方の ALU を PG できず，空間的最適化のみではキャッシュミスが頻発するような場合でも常に一つの ALU が動いてしまうため，両手法を統合して用いることで走行時 PG に適した命令実行制御ができると考えられる．

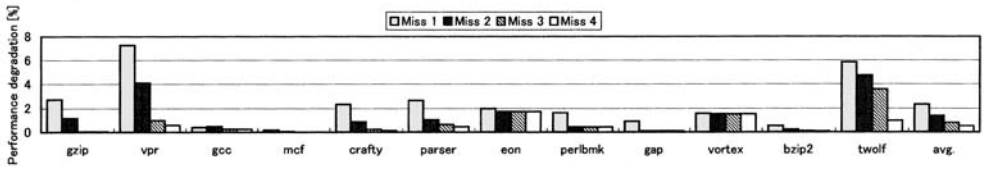


図 10 統合した手法を用いた場合の性能低下率

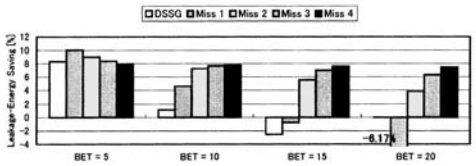


図 11 統合した手法を用いた場合のリークエネルギー削減率

6. 関連研究

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，PG 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 PG 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，従来手法よりも効果的にリーク消費エネルギーを削減でき，手法を適用したことによる性能低下は十分小さいことが分かった．

今後は，提案手法を拡張し，さらに効率的な PG 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．またプロセッサ全体を考慮した合計の消費エネルギー評価も今後の課題である．

謝辞 本研究を行うにあたり御助言を頂いた，東京大学の今井雅特任准教授，佐々木広氏，椎名公康氏に感謝を申し上げます．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」，および文部科学省科学研究費補助金 (基盤研究 (A) No.18200002) 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) 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.
10) http://www.eas.asu.edu/~ptm/
11) A. Youssef, et al., "Dynamic Standby Prediction for Leakage Tolerant Microprocessor Functional Units", In Proc. 39th MICRO, pp371.–381, Dec. 2006.