Vol.2012-ARC-198 No.2
2012/1/19

アイドル時のキャッシュ電源遮断による性能ペナルティとその削減手法

有間 英志[†1]　薦田 登志矢[†1]
三輪 忍[†1]　中村 宏[†1]

プロセッサがアイドル時に消費するリーク電力が全消費電力に占める割合は，トランジスタの微細化が進むにつれて年々上昇を続け問題となっている．このようなリーク電力を削減する目的で，プロセッサアイドル時にコアへの電源供給を遮断するパワーゲーティング技術がモバイル向け・デスクトップ向けのプロセッサで広く用いられている．現行のシステムでは，CPU のアイドル時間が一定の閾値を越えた場合にコアへの電源遮断を制御しているが，このとき問題になるのがスリープモード中にキャッシュに存在するデータが揮発することによって生じるスリープ復帰後のキャッシュミスの増大である．本研究ではコアがスリープモードに入った場合に生じるキャッシュフラッシュの影響によるキャッシュミス増大を回避するための，キャッシュプリフェッチ手法を提案しその効果について予備的な評価を行う．

1. はじめに

1.1 背景と目的

プロセッサがアイドル時に消費するスタティックな電力は，トランジスタの微細化が進むにつれて指数関数的に増加し続けており，その傾向は今後も続くものと予想されている[6]．また，一般的な PC ユーザの使用状況では，アイドル時間がほとんどを占めているため，プロセッサが無駄なスタティック電力を消費する場面は多い[9]．

リーク電力を削減するための回路技術として，パワーゲーティング (Power Gating :以下，PG) と呼ばれる回路技術が知られている．PG 技術では，処理を行っていない回路への電源電圧供給を遮断することで当該回路に流れるリーク電流を削減し，消費リーク電力を大きく削減することができる．このような PG 技術は，バッテリーによるエネルギー制約のあるモバイル用途のプロセッサをはじめ，デスクトップ向けのマルチコアプロセッサにも搭載されている[7]．

現行のプロセッサでは ACPI[1] 等の電力制御規格に従って，プロセッサコアのモードを処理の負荷に応じて動的に制御することでアイドル時に消費されるリーク電力に対処している．ACPI に準拠するするプロセッサでは，複数のスリープモードが用意されており，アイドル時間が長くなるにつれて，より深いスリープモードへと移行していく電力制御が行われる．このとき，より深いスリープモードに入ることにより多くのリーク電力を削減することが可能になる一方，スリープからの復帰にかかる時間が長くなるというトレードオフ関係が存在する．具体的にはコア内部の演算部，クロック生成器，キャッシュといったプロセッサの構成要素を復帰する際のペナルティが小さい物から順に電源を遮断していくことになる．

スリープモードのプロセッサは，入力デバイス等からの割り込みによって電源を復帰させ，アクティブなモードに移行する．そうして，要求された処理を実行することになる．この電源を復帰してから処理を実行可能になるまでの時間，すなわちウェイクアップ時間が存在するため，電源遮断によってアプリケーションの性能は低下してしまう．

そのため，現行のシステムでは十分に長いアイドルがプロセッサに生じたときのみ深いスリープモードに入るようなモード制御が行われている．逆にスリープからの復帰に要するウェイクアップ時間を削減することができれば，性能低下を引き起こすことなく従来よりも短いアイドル期間において深いスリープモードへと移行することが可能となる．このような目的のもと，パワーゲーティングによるスリープモードからの復帰時間を削減する研究が多くなされてきた[4]．

一方，キャッシュの電源を遮断した場合は，キャッシュに格納されていたデータがすべて揮発してしまう．失われたデータを電源復帰後に参照すると，電源を遮断しない場合には起こらなかった，キャッシュ・ミスが発生してしまう．すなわち，キャッシュの電源遮断においては，通常のウェイクアップ時間に加えて，上述のキャッシュ・ミスによる性能ペナルティが存在する．このペナルティを削減できれば，電源遮断にともなう性能低下を抑えることができる．結果として，キャッシュの電源を遮断する機会が広がり，さらなる電力削減へ

†1 東京大学
The University of Tokyo

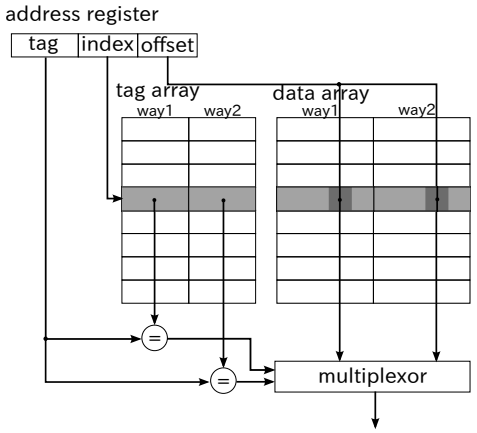

図 1 2way セットアソシアティブ方式の構成

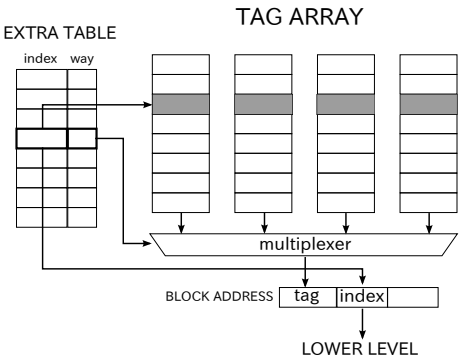

図 2 提案手法の概要

と繋がることが期待できる．

本研究では，キャッシュのスリープに伴うキャッシュ内データの揮発に伴う性能ペナルティを削減するためのキャッシュのスリープおよびスリープからの復帰手法を提案する．具体的には，キャッシュの電源を落とす際にタグアレイのみデータを残しておけるようにしておき，電源が復帰すると，利用される可能性の高いキャッシュブロックから順に，タグ部を参照してプロセッサの命令実行と並行してデータを復帰させることで，電源復帰後のキャッシュミスを減らすことを目指す．

2. 性能ペナルティ削減手法の検討

2.1 キャッシュの構成

プロセッサとメインメモリとの間の大幅な性能ギャップを解消するため，現行のプロセッサは小容量で高速アクセスが可能なキャッシュメモリを搭載している．メモリアクセスの空間的局所性から，キャッシュでは複数のデータをひとまとまりにしたブロックとよばれる単位でデータを管理している．ブロックのキャッシュへの配置方式として，今日よく使われているのはセットアソシアティブと呼ばれる方式である．図 1 は，2way セットアソシアティブ方式のキャッシュ構成を模式的に示したものである．セットアソシアティブ方式では，データのアドレスはタグ部，インデクス部，オフセット部に分割される．コアからキャッシュへのアクセスがあった場合，当該データアドレスのインデクス部を用いて当該データが存在しうるラインを決定する．タグ部は，キャッシュ内に存在するデータがアクセス要求があったアドレスのものであるかを判定するために用いられる．

セットアソシアティブ方式では，各ラインごとに複数のアドレスのデータを保持できるため，新しいデータによって古いデータを書き換える場合どのデータを上書きするかを決定するリプレースメントの問題が生じる．アプリケーションのデータアクセスには，時間的局所性が存在するため，通常 LRU ベースのリプレースメント方式が用いられる．LRU 方式のリプレースメント制御では，同一ライン内において一番長い間アクセスされなかったデータを新しいデータで置き換える．

2.2 キャッシュフラッシュを伴うスリープからの復帰

本研究では，CPU がキャッシュフラッシュを伴う深いスリープモードから復帰した直後に生じるキャッシュミスによるアプリケーションの性能低下を防ぐためのアーキテクチャ技術を検討する．キャッシュフラッシュを伴うスリープモードから復帰した直後には，スリープ直前にキャッシュに存在したデータが全て消失しているため，スリープをしなければ生じなかったキャッシュミスが頻発し，アプリケーションの性能低下が生じることになる．

このようなアプリケーションの性能低下を防ぐためには，スリープモードからの復帰と同時にスリープ直前にキャッシュに存在したデータを全てキャッシュに復帰させることができればよいと考えられる．このようなキャッシュのデータ復帰を実現するためには，スリープ直前にキャッシュ内に存在したデータのアドレスを一度メモリ上に退避させておいた上で，スリープからの復帰直後にこれらのデータをキャッシュに書き戻すことが必要である．しかし，このような素朴な方法ではスリープ復帰時にキャッシュとメモリの間のデータ転送遅延が発生し，スリープからのウェイクアップレイテンシが増大してしまう上，キャッシュデータの退避によるスリープモードに入るまでの時間の増大が生じる．また，復帰させたデータの中には再利用されることのないデータも含まれているため，このようなデータに対しては余分なメモリアクセスが発生することになりこれは消費電力の増大も引き起こすことになる．

理想的にはウェイクアップ後に再利用されるデータだけを選択的にキャッシュへ復帰させることができれば，余分なメモリアクセスを抑えつつアプリケーションの性能低下を防ぐことが可能である．このとき，アプリケーション実行の再開をできる限り早くするため，キャッシュへのデータ復帰はアプリケーション実行と並行して行われるのが望ましい．

2.3 提 案 手 法

前節で説明したように，キャッシュフラッシュを伴うスリープからの復帰時において再利

用されるデータだけを選択的にかつアプリケーション実行と並行して行う手法が必要である．

ここでは，スリープモードにおいてキャッシュのタグアレイ部だけはスリープさせずタグ内のデータを復帰時まで残しておき，この情報を用いて選択的にデータを復帰させる手法を検討する．タグのデータとラインからスリープ直前に存在していたデータのアドレスが分かるため，タグのデータがあればスリープ直前のキャッシュを復元することができる．提案手法ではスリープからの復帰時，アプリケーションの実行開始と並行してスリープ中に保存しておいたタグの情報を用いたハードウェアプリフェッチを行うことで，スリープ時におけるアプリケーションの実行開始を遅らせることなく，キャッシュデータの復元を試みる．このとき，復帰後早い期間に再利用される可能性の高いデータから順にプリフェッチを行うことでアプリケーションがキャッシュミスを頻発することを防ぐ．

スリープからの復帰後におけるプリフェッチ順序を決定するために，提案手法では利用される可能性が高いキャッシュラインを順に記録しておく追加のテーブルを用意する．このテーブルにはインデックスとウェイの値が記録される．図 2 に提案手法で用いる，追加のテーブルの模式図を示す．具体的な復帰の流れとしては，まずこの追加のテーブルにインデックスとウェイの値を取りにいき，該当するインデックスのタグを選択する．得られたタグはウェイ数分存在するのでどのタグを使用するかを，追加のテーブルから得られたウェイの値を用いて選択回路を通すことで選択する．これにより，該当するブロックのアドレスが得られるので，このアドレスを用いて下のメモリ階層にブロックを取りにいく．

アプリケーション実行時におけるキャッシュのプリフェッチは様々な手法が考案されているが[8]，本研究のようにキャッシュのデータが全てが失われる状況は想定されていない．

2.4 提案手法の電力オーバーヘッド

タグアレイと検討手法で追加したテーブルは，スリープ時にもデータを保持しつづける必要がある．これらのハードウェアにはスリープ時にも電力を供給し続ける必要があり，その分スリープモードにおけるリーク電力削減効果が減少することになるがその影響は無視できる程度のものである．ここでは，提案手法でのスリープモードのリーク電力増大分を評価するため，スリープモードにおいて保持されるデータのビット数がキャッシュ全体のビット数に対してどの程度の割合になるかを評価する．

N ビットのアドレス空間を考える．キャッシュのセット数を n_{set} とし，ブロックサイズを n_{block} バイトとする．ここで，タグのビット数は $N - \log_2(n_{set}) - \log_2(n_{block}) = N - \log_2(n_{set}n_{block})$ となる．よってタグアレイ全体のビット数はウェイ数を n_{way} として $n_{way}n_{set}(N - \log_2(n_{set}n_{block}))$ となる．また追加のテーブルの 1 エントリー当たりのビット数は $\log_2(n_{way}n_{set})$ であり，エントリー数を $n_{entry}(\leq n_{way}n_{set})$ とすると，追加のテーブル全体のビット数は $n_{entry}\log_2(n_{way}n_{set})(\leq n_{way}n_{set}\log_2(n_{way}n_{set}))$ となる．また，データアレイの全ビット数は $8n_{block}n_{way}n_{set}$ となるので，キャッシュの消費電力は SRAM セルのビット数に比例すると考えると，提案手法を用いない場合の削減電力に対する，提案手法を用いた場合の削減電力の比は次の様になる．

$$
\begin{aligned}
&\frac{8n_{block}n_{way}n_{set} - n_{way}n_{set}(N - \log_2(n_{set}n_{block})) - n_{entry}\log_2(n_{way}n_{set})}{8n_{block}n_{way}n_{set}} \\
\geq\ &\frac{8n_{block}n_{way}n_{set} - n_{way}n_{set}(N - \log_2(n_{set}n_{block})) - n_{way}n_{set}\log_2(n_{way}n_{set})}{8n_{block}n_{way}n_{set}} \\
=\ &\frac{8n_{block} - (N - \log_2(n_{set}n_{block})) - \log_2(n_{way}n_{set})}{8n_{block}} \\
=\ &1 - \frac{N - \log_2(n_{set}n_{block}) + \log_2(n_{way}n_{set})}{8n_{block}} \qquad (1)
\end{aligned}
$$

64KB，64B ブロックサイズ，ウェイ数 4 の L1 キャッシュと，2MB，64B ブロックサイズ，ウェイ数 8 の L2 キャッシュについてこの値を求めるとそれぞれ 0.945，0.943 となる．提案手法におけるスリープモードのリーク電力削減量は，通常のスリープモードにおけるリーク電力削減量の 94 % 程度であり，タグ部および追加テーブルのデータを保持することによる電力オーバーヘッドは無視できる程度のものであることが分かる．また，タグアレイと追加のテーブルのデータ保持に不揮発性の素子を用いることでこの電力オーバーヘッドを削減することも可能である．

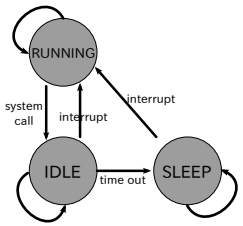

図 3 評価で想定したプロセッサの状態遷移

3. 予 備 実 験

3.1 性能ペナルティの評価方法

今回の予備実験では，本手法の有効性を確認するため，アイドル時間がある閾値を越えた

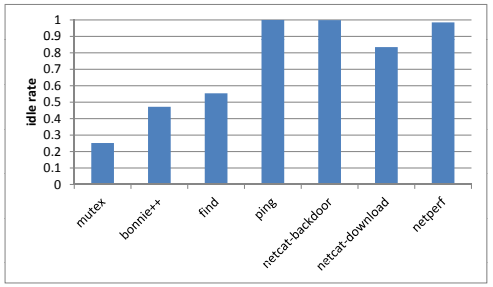

図 4 全サイクル数に占めるアイドル時間の割合

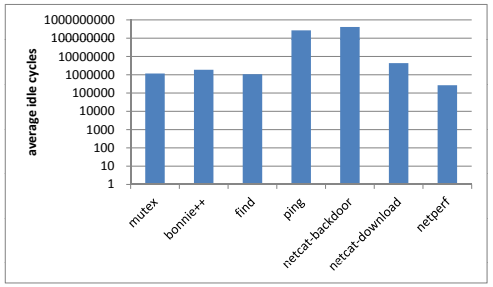

図 5 アイドル状態でのサイクル数の平均値

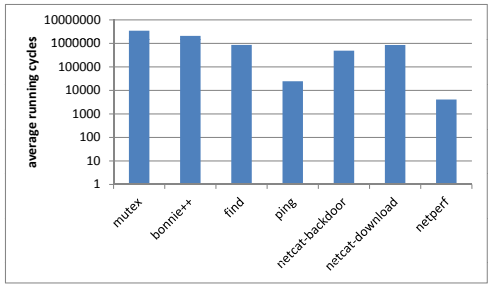

図 6 実行状態でのサイクル数の平均値

場合に，キャッシュの電源を遮断した場合の性能低下を評価する．プロセッサの状態遷移は図 3 の様なものを仮定している．RUNNING 状態はプロセッサが何らかのプロセスを実行している状態を表し，IDLE 状態はプロセッサがアイドル状態であることを表し，SLEEP 状態はキャッシュの電源を遮断した状態を表す．RUNNING 状態で実行可能なプロセスが無くなると，プロセッサはアイドル状態に入る．割り込みが入ると再び RUNNING 状態に戻り，アイドル時間が閾値を越えると SLEEP 状態に移行する．SLEEP 状態で割り込みが入ると RUNNING 状態へと移行する．

キャッシュの構成は 2 レベルとし，L1 キャッシュのみスリープさせる場合と，L1 キャッシュ，L2 キャッシュともにスリープさせる場合の両方について評価する．また，実験では性能低下を評価するためキャッシュラインの再利用回数を測定した．ここでの再利用とは，各々のキャッシュラインについてスリープからの復帰後の最初の操作が、リプレイスメントか読み書きのヒットによるアクセスかで場合分けし，アクセスである場合を再利用と定義する．再利用されるラインについてはスリープによってデータが揮発することに起因するキャッシュミスが生じるが，再利用されないラインについては，たとえデータを保持していたとしても，キャッシュミスによって置き換えられるので，データが揮発することによる性能ペナルティは発生しない．

L1 ミスペナルティを P_{l1miss}，L1 キャッシュの再利用数を $N_{l1reuse}$ とおくと，L1 キャッシュのみをスリープさせる場合の性能ペナルティ P_{sleep1} は以下の様に表すことができる．

IL1 キャッシュ	32KB，4way-set associative，64B block size，1 cycle latency
DL1 キャッシュ	64KB, 4way-set associative，64B block size，1 cycle latency
L2 キャッシュ	1MB shared，8way-set associative，64B block size，10 cycles latency
メモリ	100 cycles latency

図 10 シミュレーションに用いたプロセッサの構成

$$P_{sleep1} = N_{l1reuse} P_{l1miss} \quad (2)$$

さらに L2 ミスペナルティを P_{l2miss}，L2 キャッシュの再利用数を $N_{l2reuse}$ とおくと，L1 キャッシュ，L2 キャッシュともにスリープさせる場合の性能ペナルティ P_{sleep2} は以下の様に表すことができる．

$$P_{sleep2} = N_{l1reuse} P_{l1miss} + N_{l2reuse} P_{l2miss} \quad (3)$$

これらのペナルティが実行サイクル数にどの程度影響するのかを評価する．

3.2 評価環境

フルシステムシステムシミュレータ GEM5[3] 上で，ベンチマークプログラムを実行して再利用率を測定した．アプリケーションが I/O 待ち状態に遷移することで，長いアイドル

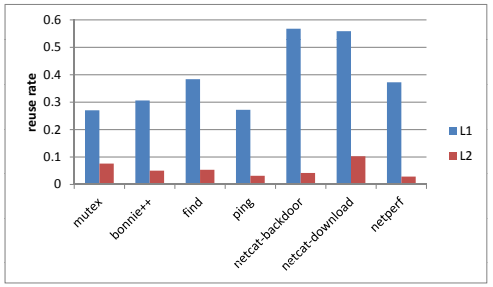

図 7 起床後に再利用されるラインの割合

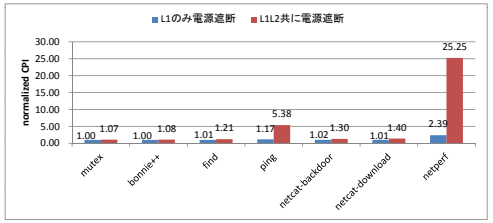

図 8 キャッシュ電源遮断による CPI の増加率

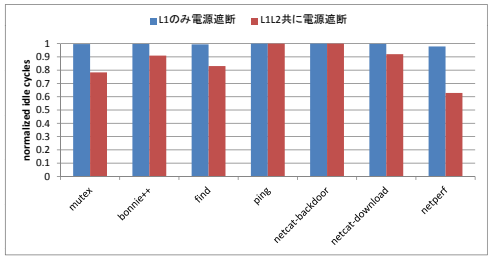

図 9 キャッシュ電源遮断によるアイドル時間の減少率

が生じるため，ベンチマークプログラムにはディスクやネットワーク等の I/O を使用するものを主に選定した．具体的には，ディスクアクセス性能を測定するためにディスクに頻繁にアクセスする bonnie++，ネットワーク性能を測定するベンチマークである netperf，linux コマンドの find，ping，netcat や，クリティカルセクションでの同期処理を取るためのプログラムである mutex をテストするベンチマーク等を選定した．find についてはルートディレクトリ以下にある全てのファイルとディレクトリをターミナルに表示させるようにし，netcat についてはネットワーク越しに他のマシンを動かすものと，巨大なファイルを転送するものを用意し，前者を netcat-backdoor，後者を netcat-download とした．

再利用数を測定するために GEM5 に変更を施した．具体的には，各キャッシュラインに，スリープからの起床後の最初のアクセスかどうかを判断できるようにフラグを追加しておき，システムがアイドルモードから復帰する際に，アイドル時間が閾値よりも長い場合に全てのキャッシュラインについてフラグを立てにいき，その後各キャッシュラインの操作時にフラグが立っていると最初の操作と見なしてフラグを下ろし，アクセスであれば再利用数をインクリメントするというものである．プロセッサの構成は，図 10 のように設定した．

3.3 実験結果

まず，図 4 はプログラム実行時間とアイドル時間を合わせた全サイクル数に占めるアイドル時間の割合を示す．これは，当該のプログラムにおいて CPU をスリープさせるチャンスすなわちリーク電力削減の機会がどれだけあるかを表している．多くのベンチマークアプリケーションで，大きなスリープのチャンスがあることが分かる．図 5 に CPU に生じる平均アイドルサイクル数を，図 6 に連続してアプリケーションが実行される場合の平均実行サイクル数を表している．縦軸は，実行サイクル数であり対枢軸となっている．

次に図 7 は，CPU のアイドル期間をまたいで再利用されるラインがキャッシュ全体のライン数に対してどの程度存在するかを表している．今回評価したベンチマークでは，L1 キャッシュで平均 39%，L2 キャッシュで平均 5.5%のデータがアイドル期間をまたいで再利用されていることが分かる．実験結果から，特に L1 キャッシュにおいてアイドル期間をまたいだデータの再利用が発生することが分かる．また，今回の実験結果では L2 キャッシュにおける再利用率が小さくなっているがこれは評価に用いたベンチマークのワーキングセットが小さく，そもそも L2 キャッシュの大部分を使用していないためであると考えられる．

一方，L2 キャッシュにおけるキャッシュミスの性能への影響は L1 キャッシュミスよりも大きいため，L1 だけを電源遮断した場合に比べて，L1，L2 キャッシュの双方を電源遮断した場合の性能低下率は大きくなる．図 8 は，前述のキャッシュの再利用率から計算したスリープによるキャッシュフラッシュによってアプリケーションの実行時間がどの程度長くなるかを，図 9 はプログラムの実行時間が伸びることによって，スリープのチャンスであるアイドル時間がどの程度減少するかを示したものである

実験結果から，今回用いたベンチマークアプリケーションにおいてはスリープに伴うキャッシュフラッシュによる性能低下の影響は無視できないほど大きなものであり，従ってスリー

プ復帰時におけるキャッシュデータの復元手法が必要であることが分かった．

4. 関 連 研 究

スタティック電力に対応するための主要な技術として，PG についてはこれまで様々な研究がなされてきた．Kawasaki ら[5] は，高速に電圧を落とすことができる回路技術を提案しており，それによってより細粒度での PG ができるようになることで，より多くの電力が削減できることを示している．Agarwal ら[2] は，複数のスリープモードを持つ PG を提案していて，徐々に深いスリープに移ることでより多くの電力が削減できるとことを示している．

5. まとめと今後の課題

本稿では，アイドル時にキャッシュの電源を遮断した場合に生じる性能ペナルティを削減するため，キャッシュの電源を遮断する際にタグアレイのみデータを残しておけるようにしておき，電源が復帰すると，利用される可能性の高いキャッシュブロックから順に，タグ部を参照してプロセッサの命令実行と並行してデータを復帰させるという手法を検討した．また，手法の有効性を示すため，スリープ時にキャッシュの電源を遮断した場合の性能低下を定式化し，フルシステムシミュレータ GEM5 上で評価実験を行った．その結果，L1 キャッシュでは半分程度，L2 キャッシュでは 1 割程度のデータのみ復帰させればよいことが分かった．また，これらのデータを復帰させなかった場合の性能低下は最悪 25 倍にも達するため，復帰させる意義は大きい．

今後の課題としては，検討手法を GEM5 上に実装し，どの程度効果があるかを実験によって確かめるという事があげられる．ただし，本稿では，使用されるの可能性が高いキャッシュラインを順番付けするためのテーブルを提案したが，具体的にどのようにして順番付けを行うかについては，検討していない．例えば，MRU(Most Recently Used) に基づいた順番付けは，メモリアクセスの空間的局所性から有効であると考えられるが，エントリー数が多い場合には，ハードウェアによる実装が困難になるという問題がある．このように，どのようにして順番づけを行うかも課題の一つである．

また，本稿では性能低下については，評価を行ったが具体的な削減電力については評価を行っていない．キャッシュの電源遮断と復帰に要するエネルギーオーバーヘッドが具体的に求まると，そこからスリープ時間の損益分岐点が算出でき，それによってどの程度スリープすべきかが分かる．

提案手法を用いた場合に，性能低下が許容範囲に存在する中で削減電力を最小にする様な，アイドルからスリープへと移行するためのアイドル時間の閾値を求め，どの程度電力が削減できたかを評価するのが今後の目標である．

6. 謝　　辞

本研究の一部は，NEDO「ノーマリーオフコンピューティング基盤技術開発」事業による．

参 考 文 献

1) Advanced Configuration and Power Interface, http://www.acpi.info/.
2) Agarwal, K., Nowka, K., Deogun, H. and Sylvester, D.: Power Gating with Multiple Sleep Modes, *Proceedings of the 7th International Symposium on Quality Electronic Design*, ISQED '06, Washington, DC, USA, IEEE Computer Society, pp.633–637 (2006).
3) Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M.D. and Wood, D.A.: The gem5 simulator, *SIGARCH Comput. Archit. News*, Vol.39, pp.1–7 (2011).
4) Kanno, Y. and et.al: Hierarchical Power Distribution with 20 Power Domains in 90-nm Low-Power Multi-CPU Processor, *Proceedings of the 2006 IEEE International Solid-State Circuits Conference*, IEEE (2006).
5) Kawasaki, K.-i., Shiota, T., Nakayama, K. and Inoue, A.: A Sub-us wake-up time power gating technique with bypass power line for rush current support, *VLSI Circuits, 2008 IEEE Symposium on*, pp.146 –147 (2008).
6) Kim, N.S., Austin, T.M., Blaauw, D., Mudge, T.N., Flautner, K., Hu, J.S., Irwin, M.J., Kandemir, M.T. and Vijaykrishnan, N.: Leakage Current: Moore's Law Meets Static Power, *IEEE Computer*, Vol.36, No.12, pp.68–75 (2003).
7) Rajesh Kumar and Glenn Hinton: A Family of 45nm IA Processors, *Proc. IEEE International Solid-State Circuits Conference* , California,USA (2009).
8) Wang, Z., Burger, D., McKinley, K.S., Reinhardt, S.K. and Weems, C.C.: Guided region prefetching: a cooperative hardware/software approach, *SIGARCH Comput. Archit. News*, Vol.31, pp.388–398 (2003).
9) Zagacki, P. and Ponnala, V.: Power Improvements on 2008 Desktop Platforms, *intel technical journal* (2008).