

キャッシュ電源遮断時の性能ペナルティ削減のための 損失データプリフェッチ

有間 英志^{1,a)} 薦田 登志矢¹ 中田 尚¹ 三輪 忍¹ 中村 宏¹

受付日 2012年12月21日, 採録日 2013年5月22日

概要: 現在のコンピュータシステムでは, プロセッサのアイドル時の電力を削減するために, OS がアイドル状態を検出するとコアやキャッシュの電源遮断が行われる. ただし, キャッシュの電源を遮断した場合には, 格納されていたデータが失われるため, 電源復帰後にそのような損失データを参照した場合, 追加のキャッシュミスが発生して性能が低下するという問題がある. そのため, 現行のシステムではキャッシュの電源遮断を行う機会は限られており, 特にラスト・レベル・キャッシュの電源遮断はほとんど行われていないのが現状である. 本稿では, このキャッシュの電源遮断にともなう性能低下を防ぐため, 電源遮断によって損失した各々のデータについて, そのデータがアクセスされる前にキャッシュに復帰させるプリフェッチ方式を提案する. 提案方式をシミュレータ上に実装し, 実験を行った結果, 電源遮断後も再利用されるような損失データのうち, 多くのデータについてプリフェッチが成功した. これにより電源遮断にともなう性能低下を大幅に抑えることができ, 電源遮断の機会を広げることができることが分かった.

キーワード: キャッシュ, 省電力化, プリフェッチ

Lost Data Prefetching to Reduce Performance Degradation Caused by Powering off Caches

EISHI ARIMA^{1,a)} TOSHIYA KOMODA¹ TAKASHI NAKADA¹ SHINOBU MIWA¹
HIROSHI NAKAMURA¹

Received: December 21, 2012, Accepted: May 22, 2013

Abstract: In current computer systems, to reduce power consumption of a processor in idle state, cores and caches are powered off when OS detects the idle state. However, powering off caches causes performance degradation, because it invokes data loss and additional cache misses. For this reason, caches, especially last level cache, are infrequently powered off in modern systems. To cope with this problem we propose a novel prefetch scheme: restoring such lost data before they are re-referenced. The experimental results show lots of lost data can be restored before re-references. Hence, it is cleared that this method suppresses performance degradation and increases opportunity of powering off caches.

Keywords: cache, low power, prefetch

1. はじめに

近年のコンピュータシステムの設計においては, システムの消費電力をいかに抑えるかが重要な課題となっている. 消費電力の削減は特に携帯情報端末において重要であ

り, 電力を削減することで, たとえばバッテリー駆動時間の延長や放熱部品の小型化などの恩恵を受けることができる. また, 近年では携帯情報端末だけでなく, デスクトップやサーバなどのミドル/ハイエンド PC においても省電力化の重要性は増している.

一般的な PC ユーザの使用状況下では, プロセッサはほとんどの時間でアイドル状態となっていることが知られている. 文献 [1] によれば, システムの電源が投入されてい

¹ 東京大学
The University of Tokyo, Bunkyo, Tokyo 113-8656, Japan
^{a)} arima@hal.ipc.i.u-tokyo.ac.jp

る時間のうちの75%以上がアイドル状態にある。したがって、システムの消費電力を削減するうえで、アイドル状態の電力を削減することの持つ意味は大きい。

現行のプロセッサでは、ACPI[2]などの電力制御規格に従ってプロセッサコアやキャッシュの電源を遮断することにより、アイドル時の消費電力を削減している。ACPIに準拠するプロセッサでは、電源遮断を行うコンポーネントに応じて、複数のスリープモードが用意されている。具体的にはキャッシュ以外のコア、クロック生成器、キャッシュなどのコンポーネントが制御の対象となる。OSがアイドル状態を検出すると、そのときの状況に応じてどのコンポーネントの電源遮断を行うかを決定する。そして、制御の対象に選ばれた各ハードウェアでは、アイドル状態の経過時間に応じて、自身の電源を遮断する。

省電力化のためには、アイドル状態においてなるべく多くのコンポーネントの電源を遮断した方がよい。また、その中でも特に消費電力の大きいコンポーネントの電源を遮断することが重要となる。文献[3]によれば、プロセッサの消費電力のうち、30%以上がキャッシュのスタティック電力によって消費されている。そのため、アイドル状態においてキャッシュの電源遮断を行うことが重要である。

しかし、キャッシュの電源を遮断した場合には性能が低下するという問題がある。これはキャッシュに格納されていたデータが電源遮断によって損失し、スリープ状態からの復帰後に、損失したデータにアクセスすることで、追加のキャッシュミスが発生するためである。このようなキャッシュミスは電源遮断を行わなければ発生せず、電源遮断による性能ペナルティと位置づけることができる。後述するように、この性能ペナルティによってプロセッサ全体の性能は大きく低下するため、現在のプロセッサではキャッシュの電源を遮断する機会が限られており、電力を十分に削減できていない。

このような性能ペナルティに対処するため、STT-MRAM (Spin Transfer Torque Magnetoresistive RAM) といったデータの保持に電源供給が不要な不揮発性メモリを、キャッシュに用いる研究がさかんに行われている[4], [5]。しかし、このような不揮発性メモリは通常キャッシュに用いられるSRAMに比べてアクセス性能が低いため、プロセッサの性能が低下するという問題がある。特に高負荷時の性能が低下してしまうことが問題となる。また、STT-MRAMは製造技術が十分に確立されていないこともあり、キャッシュをSRAMで構成するのが現在の商用プロセッサではまだ主流である。

そこで本研究では、再び利用される可能性が高いデータを、スリープ状態から復帰した後にバースト的にプリフェッチすることで、上述の性能ペナルティを削減する手法を提案する。提案手法では、キャッシュの電源を遮断する際にデータアレイの電源のみを遮断し、タグアレイに記

録された内容は残しておく。そうしてアクティブ状態に復帰した際に、残しておいたタグの情報を参照し、今後アクセスされるであろうデータを順次キャッシュへと書き戻す。そのデータへのアクセスが発生するよりも前に書き戻しが完了すれば、そのアクセスは損失したはずのデータにヒットすることになり、電源遮断による性能低下を抑えることができる。

本稿の構成は以下のとおりである。まず次章ではキャッシュの電源遮断による性能ペナルティについて分析する。続く3章ではその性能ペナルティを削減する手法として、今回提案するプリフェッチ方式を詳しく説明する。4章では提案方式の評価を行い、5章で関連研究について述べ、6章ではまとめと今後の課題について述べる。

2. 電源遮断による性能ペナルティ

2.1 キャッシュの電源遮断

近年のデスクトップPCやモバイル機器において、複数のコアを搭載するCPUを利用することが主流となっている。しかし、実用的な場面ではすべてのコアが稼働していることはまれである。システム稼働中のほとんどの期間は、計算負荷の小さい少数のプロセスが断続的に実行される、いわゆる低負荷な状況であることが知られている[1]。

低負荷な状況では、OSによって処理が割り当てられないコアが発生し、これらのコアはアイドル状態に移行する。性能だけを考えた場合、アイドル状態にあるコアの制御は特に問題にならない。しかし、消費電力を考えた場合事情は異なる。アイドル状態といえどもコアは電力を消費するためである。アイドル状態にあるコアにおける消費電力を削減するため、当該のアイドル状態に応じた電源遮断モードへの遷移が行われる。

L1キャッシュ(Level 1 Cache. 以下**L1C**とする)のように、各コア専用のコンポーネントは、当該コアがアイドルになれば電源遮断可能である。L1キャッシュの電源遮断は、他のコアとは独立に制御することが可能である。一方、ラスト・レベル・キャッシュ(Last Level Cache. 以下**LLC**とする)のようにすべてのコアで共有されるコンポーネントは、すべてのコアがアイドル状態になった場合にのみ、電源遮断が可能になる。このため、LLCの電源遮断が行われる直前では、実行可能なプロセスはシステム全体で1つだけであるのが通常である。また、LLCが電源遮断モードから復帰するのは、割込み処理からの復帰などにより新たに実行可能プロセスが発生したタイミングである。このとき、LLCの復帰直後に実行可能なプロセスは、LLCの復帰の引き金を引いたプロセスだけであると考えられる。理論的には、割込みが同じタイミングで重なることによって、LLCの復帰直後に複数のプロセスが実行可能である状況はありうるが、このようなケースはまれである。

したがって、キャッシュの電源遮断の問題を考える場合、

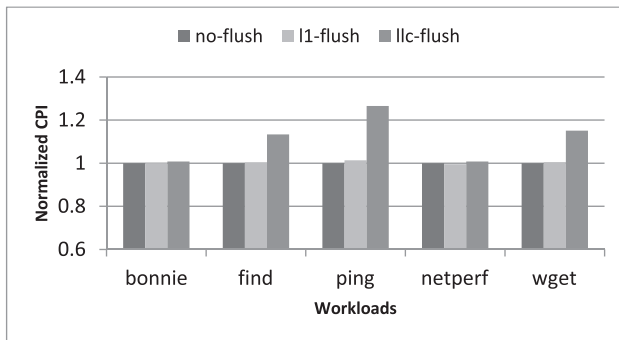


図 1 キャッシュフラッシュによる性能低下. 実験環境は後述する config A と同様

Fig. 1 Performance degradation caused by cache flush. Configuration is same as config A, which is referred later.

単一コア上で単一のアプリケーションが実行されている状況のみを考えれば十分である. 以降, 本稿では単一コア上で単一アプリケーションが動作する場面を想定して議論を行うが, これらの議論は複数のコアを持つ CPU に対してもそのまま適用することが可能である.

2.2 性能ペナルティとその影響

本稿の冒頭で述べたように, OS がアイドル状態を検出すると, キャッシュなどの個々のコンポーネントは低電力モードへと移行しようとする. キャッシュの場合は低電力モードとして電源が遮断された状態が用意されているが, キャッシュの電源を遮断するとキャッシュ上のデータがすべて失われてしまい, 性能が大きく低下するという問題がある.

予備評価として, OS がアイドル状態を検出したタイミングでキャッシュをフラッシュした場合に, プロセッサ性能がどの程度低下するかを評価した. プロセッサ・シミュレータ Gem5 上で Linux を稼働させ, その上でユーザが何らかのタスクを実行している状況を想定し, 評価を行った. キャッシュは L1C/LLC の 2 階層からなるシステムを想定する. その他, 評価環境の詳しい説明は 4.1 節で行う.

図 1 がその結果である. グラフの横軸がタスクを表し, 縦軸はフラッシュを行わない場合に対する CPI の増加率を表している. 評価は, L1C のみをフラッシュした場合, および, L1C/LLC とともにフラッシュした場合の両方のケースについて行った.

結果より, まずキャッシュ・フラッシュによって性能が低下していることから, find や wget などのタスクをユーザが実行中であってもフラッシュの機会があった, すなわち, OS がシステムがアイドル状態になったと判断する機会がそれなりにあったことが分かる. これは, こうしたコマンドの実行中にディスク・アクセスや通信が発生し, その応答を待っている間に他に行うべき処理がない場合に, OS はシステムがアイドル状態になったと判断しているも

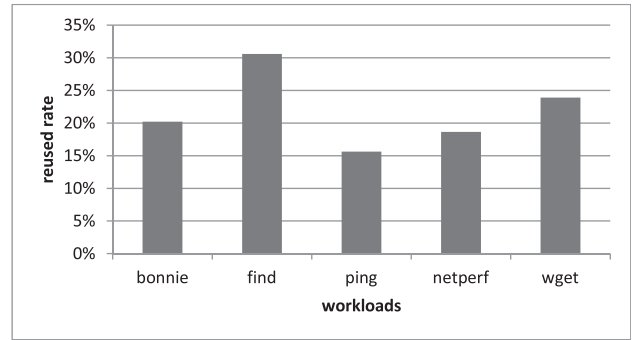


図 2 全キャッシュラインに占める再利用データの割合. 実験環境は後述の config A と同様

Fig. 2 Ratio of reused lines to all cache lines. Configuration is same as config A, which is referred later.

のと考えられる.

次に, フラッシュを行う場合どうして比較すると, L1C をフラッシュすることによる性能低下はそれほど大きくないが, LLC までフラッシュすると性能が大幅に低下することが分かる. タスクによっては実行時間が 2 割近く増加している. 今回の評価では, ユーザが何らかのタスクを実行中に発生する I/O 待ちに起因するアイドル状態においてキャッシュの電源を遮断することを想定したが, このような場合にタスクの実行時間が 1.2 倍になるというのは, ユーザにとっては受け入れがたいものがある.

そのため現在のコンピュータ・システムでは, OS がアイドル状態と判断した場合でもキャッシュ, 特に LLC の電源遮断はほとんど行われていない. LLC の電源遮断が行われるのは, システム全体がスリープ状態に移行する場合に限られている^{*1}.

2.3 アイドル状態から復帰後に再利用されるデータ

前節で述べたように, OS がアイドル状態と判断するたびに LLC の電源を遮断していたのでは性能低下が大きすぎる. これは, アイドル状態から復帰した後に電源遮断によって失われたデータをアクセスした場合に, 電源を遮断しない場合には発生しなかったキャッシュ・ミス (電源遮断による性能ペナルティ) が発生するためである.

図 2 は, アイドル状態となる前に LLCに残っていたデータのうち, アイドル状態から復帰した後に参照されたデータ (再利用データ) の割合を表している. すなわち, 電源遮断時の性能低下を引き起こすデータの割合である. 評価条件は前節と同様とした.

グラフより, 電源遮断による性能低下を引き起こすのは LLC 上の一部のデータであることが分かる. 最も多いタスクでも全体の 3 割程度にすぎない. すなわち, これらの

^{*1} Intel の省電力技術である C-State Technology では, LLC の電源が遮断された状態として C7 state が定義されているが, デフォルトではこの状態は使用されない. この状態を使用するためには, ユーザが BIOS の設定を変更する必要がある.

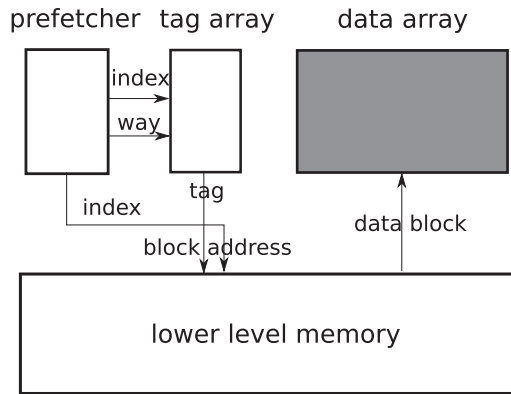


図 3 提案プリフェッチの概要

Fig. 3 Overview of proposal prefetch.

データへのアクセスさえキャッシュにヒットすれば、LLCの電源を遮断してもプロセッサの性能は低下しない。

そこで我々は、このようなデータを予測し、それへのアクセスが実際に発生する前にキャッシュへと書き戻すことで、電源遮断による性能ペナルティを抑制する手法を提案する。次章からはその方法を詳しく述べる。

3. 性能ペナルティ削減手法

3.1 タグを用いたデータアレイの復帰

本稿では、STT-MRAMのようにスリープ中もキャッシュデータを保持するのではなく、スリープ状態から復帰した後に、今後参照されるであろうデータを選択的にキャッシュに書き戻す(プリフェッチする)ことで、スリープにともなうキャッシュミス増加を抑制する手法を考える。この手法は、STT-MRAMのような不揮発性メモリを使用したキャッシュミス発生抑制手法に対して、SRAMなど、高速な揮発性メモリデバイスを用いて実装することができる点が利点となる。

復帰後に参照されるデータを書き戻すためには、スリープする前にキャッシュに保持していたデータのメモリアドレスを、スリープ中も保存しておく必要がある。提案手法では、既存キャッシュのタグアレイとデータアレイのパワードメインを分離し、面積的に大部分を占めるデータアレイのみスリープさせ、タグアレイはスリープさせずタグ内の情報を保持しておくことでこの問題に対処する。

図 3 にタグアレイに残された情報を用いたプリフェッチの概要を示す。ここでは、キャッシュの電源復帰直後でデータアレイは揮発し、タグアレイのみデータが残っている状態を考える。新しいハードウェアとして損失データ・プリフェッチャを用意する。損失データ・プリフェッチャは、直前のキャッシュアクセスのデータアドレスを入力とし、近い将来に参照されるであろうラインのアドレスを出力する。プリフェッチャが予測したアドレスを用いてタグアレイを参照し、ヒットした場合にはプリフェッチが行われる。これにより、キャッシュの電源遮断によって失われ

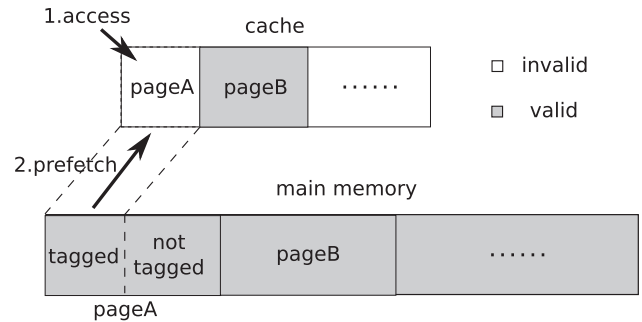


図 4 ページ単位のプリフェッチの概要

Fig. 4 Overview of page level prefetch.

たデータを、近い将来にアクセスされる可能性の高いものから順にキャッシュに書き戻すことができる。

上述のように、提案手法ではキャッシュの電源遮断時にもタグアレイを通電しておかなければならないが、それによる消費電力の増加は微々たるものといえる。タグの bit 数は 32 bit メモリ空間では 20 bit 程度であり、ラインサイズを 64 B と仮定すると、タグの容量はキャッシュ全体の 4%程度にすぎない。一般に、リーク電力は回路規模に比例するため、キャッシュ全体のリーク電力に占めるタグアレイのそれは 4%程度である。

タグアレイのリーク電力を減らすため、タグアレイのみを STT-MRAM のような不揮発性メモリで構成し、データアレイを通常の SRAM によって構成したうえで提案手法を用いることも考えられる。タグアレイを STT-MRAM にすることによってタグのアクセス・レイテンシは増加するが、LLC においてはデータアレイのアクセス・レイテンシやバスのレイテンシが支配的であることから、プロセッサ性能に与える影響は少ないと考えられる。

3.2 プリフェッチするデータの選択方式

データアクセスには、一般に、空間的局所性があることが知られている。すなわち、あるデータがアクセスされると、その近傍のアドレスのデータもアクセスされるという性質である。この局所性を考慮し、あるラインがアクセスされると、その付近のラインは短時間の間にアクセスされる可能性が高いと考え、プリフェッチを行う。

プリフェッチは、ある程度の空間的なまとまりをもって行う。本稿では、このまとまりをページ単位とする(図 4)。すなわち、あるデータへのアクセスが発生すると、そのデータが属するページ内の他のデータへのアクセスが近い将来に発生すると見なす。このようにして今後アクセスされる可能性の高いラインのアドレスを求め、タグアレイを参照し、ヒットした場合にはそのラインをプリフェッチする。

電源遮断前のキャッシュには同じページに属する複数のラインが格納されており、性能ペナルティを抑えるためには、それらのラインをどのような順序で復帰するかが重要となる。次章で述べる評価では、簡単のため、ページの先

頭のラインから順に復帰処理を行うものとしたが、この点にはまだ工夫の余地がある。空間的局所性を考慮すれば、たとえば、電源復帰後にキャッシュ・アクセスが発生した際、そのときアクセスされたラインの後続のアドレスのラインから順にプリフェッチを行うのが自然であろう。ページ内のラインの復帰順序を変えた場合の提案方式の効果については、今後評価する予定である。

上述のように、プリフェッチを行うラインは、タグアレイにそのアドレスが残っているもの、すなわち、電源遮断前にキャッシュに残っていたラインに限るものとする。これは、電源遮断前にキャッシュに存在しなかったデータは、そもそも損失したデータではないため、復帰させなくても性能低下には寄与しないためである。

電源復帰後に発生するデマンドアクセスにより、タグアレイの状態は次々と更新される。それにともない、電源遮断前にキャッシュされていたデータに関するタグの情報は、時間とともに徐々に減っていく。提案方式では、電源遮断前にキャッシュされていたデータのうち、その時点でタグアレイに残っているラインのみがプリフェッチの対象となる。

通常のプリフェッチとは異なり、提案するプリフェッチャは、デマンドアクセスによってキャッシュされたデータを追い出すことはない点に注意されたい。提案方式は、電源遮断前にキャッシュに格納されていたデータを、それへのアクセスが発生する前に元に戻すことが目的である。すなわち、電源遮断によって invalid な状態になったラインを（それへのアクセスの発生前に）valid に戻すことが目的である。そのため、通常のプリフェッチとは異なり、提案するプリフェッチによってキャッシュが汚れることはない。

提案プリフェッチャは通常のプリフェッチャと併用することもできる。すなわち、キャッシュの電源遮断から復帰してしばらくの間は提案方式によるプリフェッチを行い、電源遮断前のデータが十分書き戻された後は通常のプリフェッチャによるプリフェッチを行う。上述のように提案方式はキャッシュを汚すことなく電源遮断前に存在したデータをバンド幅が許す限り書き戻そうとするため、復帰直後においては通常のプリフェッチャよりも高い効果を発揮すると考えられる。この点に関しては今後詳しく評価する。

3.3 プリフェッチャの実装

図 5 に実装した損失データ・プリフェッチャの構成を示す。プリフェッチ・キュー、および、Requested Line Address Register（以下 **RLAR** とする）が追加される。ただし、プリフェッチ・キューは通常のプリフェッチャのそれと共用できるため、実質的な追加ハードウェアは RLAR のみである。

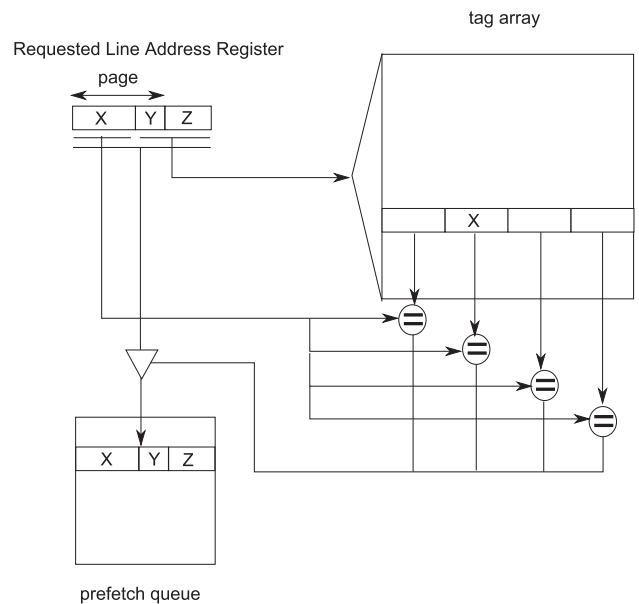


図 5 提案プリフェッチャの実装方式

Fig. 5 Implementation methodology of proposal prefetcher.

キャッシュへのデータアクセスが発生すると、当該デマンドアクセスによるキャッシュの参照が行われる。それと同時に、参照したデータの属するページのアドレスを求め、それを RLAR に登録する。たとえば、ページサイズが 4KB であれば、アクセスされたデータのアドレスのオフセットを含む、下位 12 ビットを切り捨てたものを登録する。図では、ラインアドレスのうち、X の部分と Y の部分がページアドレスを表すものとしている。

上述のデマンドアクセスによるタグの参照が終わり、タグアレイのポートが空いており、かつ、プリフェッチ・キューに空きがある場合には、続いてプリフェッチのためのタグアレイの参照が行われる。RLAR に格納されているアドレスを用いてタグアレイを参照する。図では、X がタグを、Y と Z によってインデックスを表している。

上記の参照がヒットし、かつ、valid ビットが 0 だった場合は、そのラインはプリフェッチの対象となる。そこで、RLAR に格納されているアドレスをプリフェッチ・キューに格納する。ミスした場合、あるいは、ヒットした場合でも valid ビットが 1 だった場合は何もしない。

上記の処理が完了すると、RLAR の値をインクリメントする。そして、タグアレイのポート、および、プリフェッチ・キューに空きがある限り、上記の処理を繰り返す。

RLAR はページに含まれるアドレスの範囲内でインクリメントされる。RLAR の値がページの末尾に到達すると、プリフェッチ・キューに空きがある場合でも、タグアレイの参照は行われない。

プリフェッチリクエストの発行は、通常のプリフェッチと同様の方式で行われる。すなわち、下位のメモリにアクセスするためのバスがデマンドアクセスによって使用されておらず、空いていた場合にはリクエストが発行される。

その際、プリフェッチ・キューの先頭からアドレスを1つ取り出し、下位のメモリに対してそのデータを要求する。

プリフェッチのためにタグアレイを参照している最中に、デマンドアクセスが発生したとしよう。その場合は、上述の参照を中止し、デマンドアクセスによるキャッシュの参照を優先する。そして、当該アクセスにより要求されたデータのページアドレスがRLARのページアドレスと異なる場合は、RLARを更新する。以降は、このページアドレスをもとにタグアレイの参照を繰り返す。これにより、最近アクセスされたデータの属するページ内のデータが優先的にプリフェッチされる。なお、デマンドアクセスが要求したデータのページアドレスとRLARのページアドレスが一致していた場合は、RLARは更新されない。

すべてのデータの書き戻しが終わったページについては、上述のタグアレイの参照を再度繰り返すのはエネルギー的に無駄である。そこで、図には明記していないが、すべてのデータの書き戻しが終わったページのアドレスを記録しておくテーブルを設けておく。デマンドアクセスによってRLARの値を更新する際、このテーブルを参照し、それにヒットした場合にはRLARを更新しないものとする。

上述のように、電源復帰後のキャッシュは時間の経過とともにデマンドアクセスによって格納されたデータが増えていき、提案方式によるプリフェッチが行われにくくなる。それにもかかわらず、上述したタグアレイの参照を繰り返すと無駄なエネルギーを消費する。また、時間の経過とともに実行状況が電源遮断前とは変わり、遮断前のデータを参照する機会は減っていくと予想される。すなわち、電源復帰後ある程度経過した後に損失データ・プリフェッチャによって要求されるデータは、それをキャッシュに書き戻したとしても参照されない可能性が高い。下位のメモリを参照するためのエネルギーや書き戻しのためのエネルギーを無駄に消費するだけである。

そこで、次章で述べる評価では、プリフェッチを行うページ数に上限を設けた。プリフェッチ済みのページ数がこの上限に達したら、たとえまだ書き戻しが終わっていないデータがキャッシュに残っていたとしてもプリフェッチを停止する。

4. 実験

提案方式をシミュレータに実装し評価を行った。以下ではまず実験の内容を詳しく説明し、次いでその結果を述べる。

4.1 実験内容

本節では、まず、今回の実験で想定したキャッシュの電源遮断に至るシナリオを説明する。次いで、具体的な実験環境やシミュレーションに用いたパラメータについて述べる。最後に今回エネルギーの評価も行ったので、その際に

用いたエネルギーモデルを説明する。

4.1.1 想定する状況

本稿の冒頭で述べたように、提案方式は携帯情報端末やデスクトップPCなどの個人用情報機器において使用することを想定している。個人用情報機器ではI/O待ちなどで頻繁にアイドル状態となるため、キャッシュの電源遮断を行う機会も多い。今回はノートPCクラスのプロセッサ、および、デスクトップPCクラスのプロセッサの両方において提案方式の効果を評価した。これらのプロセッサのLLCに提案方式を適用する。

最近の個人用情報機器に搭載されるプロセッサはマルチコア化されており、LLCをコア間で共有する構成となっているものが多い。本稿ではLLCの電源遮断を行う状況を想定しているが、マルチコアプロセッサでLLCの電源遮断を行う場合は全コアがアイドル状態となっている必要がある。そのため、以降の評価では、マルチコアの1コアでタスクを実行し、その他のすべてのコアはすでにアイドル状態になっている状況を想定する。このような状況ではタスクを実行中のコアがI/O待ちなどでアイドル状態になると、LLCの電源が遮断できるようになる。そして、I/Oからのリクエストが返ってくると、割込みが発生してプロセッサはアクティブ状態へと復帰する。

タスクとして、I/O待ちをとまなう何らかの処理をユーザが実行している状況を想定する。2章で述べたように、I/O待ちの間に他に実行するプロセスがなければ、OSはCPUがアイドル状態になったと判断する可能性が高い。

以降の評価では、シミュレータで実際にOSを稼働させ、OSがCPUがアイドル状態になったと判断したときにはつねにLLCの電源遮断を行うものとした。前述のようにLLCの電源遮断は大きな性能ペナルティをとまなうため、このような積極的な電源遮断は実際には行われていないが、我々が目指すのは性能ペナルティの削減によって電源遮断の機会を増やすことである。後述するように、提案方式を用いた場合は、このような積極的な電源遮断を行った場合でもあまり性能は低下しない。

4.1.2 実験環境

フルシステムシステムシミュレータgem5[6]上でタスクを与え前述の評価を行った。シミュレーションに用いたパラメータを表1に示す。前述のようにシミュレーションはノートPCクラスのローエンドなもの(config A)とデスクトップPCクラスの中ドルレンジのもの(config B)について行った。それぞれのパラメータは、前者についてはIntel社のAtomプロセッサを、後者についてはCore i7プロセッサの値を参考にした。

OSはLinuxカーネル2.6.27とし、キャッシュはL1C、LLCの2階層とする。OSがCPUがアイドル状態になったと判断しhalt命令を発行すると、すべてのキャッシュの電源が遮断されたと見なし、L1CとLLCをフラッシュす

る。損失データ・プリフェッチャは、特に電源遮断による性能ペナルティが大きい LLC に対してのみ（メモリから LLC へのプリフェッチのみ）行った。ただし、今回の実験では、プリフェッチ動作におけるタグの参照とプリフェッチキューへの登録の、時間的なオーバーヘッドがないものとした。

LLC のレイテンシは config A は 10 サイクル、config B は 18 サイクルとした。ポートはリード/ライト共有のものが 1 つとする。

表 1 の Table Entry はどのページに対してプリフェッチを行ったかを保存しておくテーブルのエントリ数（プリフェッチの上限ページ数）、Queue Entry はプリフェッチ・キューのエントリ数である。それぞれ 512 エントリ、256 エントリとする。なお、プリフェッチの上限ページ数については、後で 32~512 まで変えて評価する。

メモリ・バンド幅は config A は 6.4 GB/s、config B は 12.8 GB/s とする。ただし、バンド幅は提案方式の効果に強く影響を及ぼすと考えられることから、後の評価では 1.6~25.6 GB/s まで変化させてその影響を検証する。

表のコア数が 1 になっていることに注意されたい。前述のように、最近のプロセッサは CMP が主流であり、本実

験では CMP の 1 コアを除いた他のすべてのコアはすでにアイドル状態になった状況を想定している。稼働しているコアが 1 つだけであることから、シミュレーション時間を短縮するため、シミュレーションはシングルコアで行った。ただし、ハードウェアとしては CMP を想定していることから、LLC はコア間で共有されているものとして容量を大きく（config A は 512 KB、config B は 4 MB）してある。ユーザが実行するタスクとして以下のものを与えた。

bonnie ディスク性能を計測するベンチマークである bonnie++ を実行する。多数のディスク・アクセスが短期間に発生し、CPU が頻繁にアイドル状態となる。

find find コマンドを実行する。bonnie 同様、頻繁なディスク・アクセスが発生し、アイドル状態になりやすいと考えられる。ホームディレクトリ以下のファイルを探すように引数として与えた。

ping ping コマンドを実行する。イーサネットに接続された 2 台のシステムが存在する環境を Gem5 上に再現し、そのうちの 1 台からもう 1 台の方へ向けて ping を実行した。自身がパケットを送信した後、相手からパケットが返ってくるまでの間にアイドル状態になると考えられる。パケットの送信間隔は 1 秒とした。

netperf ネットワーク診断ツールである netperf を実行する。頻繁にネットワークからの応答待ちの状態となり、アイドル状態になりやすい。

wget wget コマンドを実行する。ping の場合と同様 2 台のシステムが起動しており、wget コマンドによって、もう 1 台の http サーバからデータを取得する。

4.1.3 エネルギーの評価方法

提案プリフェッチを適用する場合では、電源遮断のみを行う場合と比較して、アクティブな時間が少なくなるので、その分スタティックなエネルギーも削減できると考えられる。その一方で、再利用されないデータを復帰させてしまうため、メインメモリへのアクセスが増えるという問題がある。また、メインメモリのアクセスエネルギーが大きいと、そもそも電源遮断を行わない方がエネルギー的に得する場合も考えられる。このように、いずれの手法がエネルギー的に優れているのかについて一概にはいえない。そこで、電源遮断や提案プリフェッチャを適用することによる、エネルギー的な得失についても定量的な評価を与える。ただし、ここでのエネルギー評価は LLC のみを対象とした。また、今回の評価では、電源遮断時のタグのスタティックエネルギー、損失データ・プリフェッチャのタグ検索時のエネルギーは考慮していない。

LLC のダイナミックエネルギーを $E_{dynamic}$ 、LLC のスタティックエネルギーを E_{static} 、電源遮断やプリフェッチにともなう追加のメインメモリアクセスによるエネルギーオーバーヘッドを $E_{overhead}$ とすると、エネルギー評価式は以下ようになる。

表 1 シミュレーションに用いたパラメータ
Table 1 Paramaters utilized in simulation.

Parameters		Remarks	
		config A	config B
OS:	kernel	linux 2.6.27	
	Page Size	8 KB	
CPU:	#Core	1 ※	
	ISA	alpha	
	Cpu Type	in-order	OoO
	Clock	1.6 GHz	3.0 GHz
	Fetch Width	2	4
	Issue Width	2	6
L1C:	Capacity	32 KB (inst.)	32 KB (inst.)
		32 KB (data)	32 KB (data)
	Latency	2 cycles	3 cycles
	#Way	4	4
	Line Size	64 B	64 B
LLC:	Capacity	512 KB	4 MB
	Latency	10 cycles	18 cycles
	#Way	8	16
	Line Size	64 B	64 B
	#Ports	1	1
Prefetcher:	#Table Entry	256	
	#Queue Entry	256	
Main Memory:	latency	160 cycles	300 cycles
	Bandwidth	6.4 GB/s	12.8 GB/s
Disk:	Latency	10 ms	
Ethernet:	Link Delay	100 μ s	
	Bus Bandwidth	100 Mb/s	

表 2 エネルギーパラメータ
Table 2 Energy parameters.

Parameters	Remarks	
	config A	config B
SRAM (LLC):		
Leakage Power: P_{sram}	0.373 W	2.86 W
Dynamic Energy: E_{sram}	0.153 nJ/line	0.843 nJ/line
DRAM (main memory):		
Dynamic Energy: E_{dram}	51 nJ/line	

$$E = E_{dynamic} + E_{static} + E_{overhead}$$

$$= n_{LLC}E_{sram} + T_{on}P_{sram} + n_{extra}E_{dram}$$

ただし, n_{LLC} は LLC へのアクセス回数, T_{on} は LLC への通電時間, n_{extra} は電源遮断を行わない場合に対するメインメモリアクセス回数の増分, E_{sram} は LLC のアクセスエネルギー, P_{sram} は LLC のスタティック電力, E_{dram} はメインメモリアクセスエネルギーを表す. n_{LLC} , T_{on} , n_{extra} をシミュレーションの結果から取得し, E_{sram} , P_{sram} , E_{dram} を表 2 のように与えて, エネルギーを算出した. ただし, 表 2 のパラメータは CACTI [7] を用いて算出した.

4.2 実験結果

本節ではまず, 損失データ・プリフェッチャによる性能ペナルティの削減効果と, エネルギー削減効果の評価結果を示す. 次いで, リソースによる制約が変化した場合の提案手法の効果を示す.

4.2.1 性能ペナルティとエネルギーの削減効果

まず, config A, config B の構成について, 損失データ・プリフェッチを行った場合の, 再利用データの復帰率をそれぞれ図 6, 図 7 に示す. ただし, ここでの復帰率とは, 全再利用ラインのうち, アクセスされる前に復帰できたラインの割合を意味している. グラフで横軸は各タスクを表し, 縦軸に復帰率を示している.

config A の場合では, 60~80% 近くの再利用データについて復帰が成功しており, config B の場合では, 50~70% の再利用データについて復帰が成功している. config B の方が復帰率が低いのは, アウト・オブ・オーダー実行であり, 動作周波数も高いため, デマンドアクセスの間隔が短く, より短い時間でデータ復帰を間に合わせる必要があるためである.

次に, config A, config B の構成について, 損失データ・プリフェッチャによる性能ペナルティ削減効果をそれぞれ図 8, 図 9 に示す. ここで, 凡例 no-flush, ll-flush, llc-flush, lost-data-prefetch は, それぞれ, キャッシュの電源遮断を行わない場合, L1 のみ電源遮断を行う場合, LLC まで電源遮断を行う場合, LLC まで電源遮断を行って電源復帰後に損失データ・プリフェッチを行う場合を表している. 縦軸は正規化 CPI を示しており, no-flush の場合の

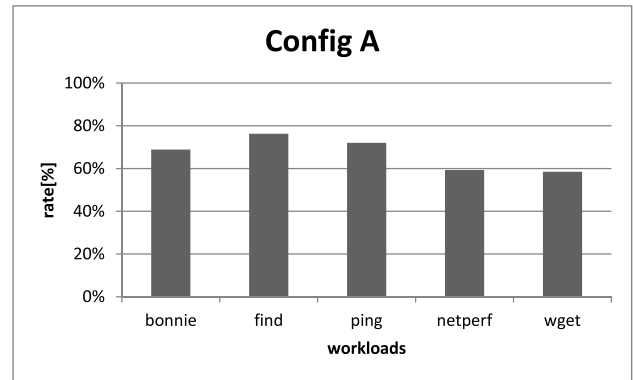


図 6 再利用データの復帰率 (config A)

Fig. 6 Ratio of restored data to reused data (config A).

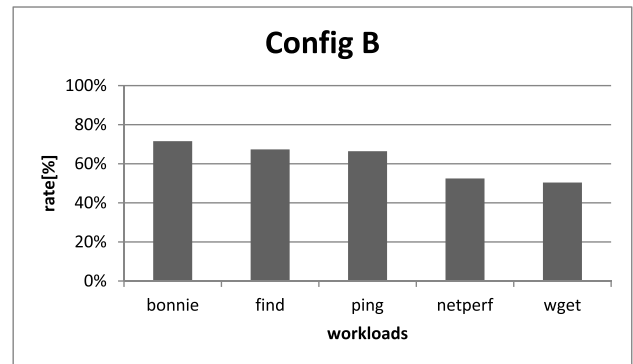


図 7 再利用データの復帰率 (config B)

Fig. 7 Ratio of restored data to reused data (config B).

CPI によって正規化されている. また, 横軸は各タスクを表している.

ll-flush では LLC のデータ損失がないため, 損失データ・プリフェッチがすべて間に合う場合と等価であり, 提案手法の性能限界を表している. 提案手法では, 図 8, 9 に示すように, いずれの構成においても, 損失データ・プリフェッチによって性能ペナルティを大幅に削減できているが, 削減率に関しては, 前述の復帰率の影響を受けていると考えられる. また, config B の方が電源遮断による性能低下が大きくなっている. これは, config B の方が動作周波数が高いため, メインメモリアクセスにかかるサイクル数も長く, メインメモリアクセスによる性能へのペナルティも大きくなるためであり, また, レイテンシが長いためにアウト・オブ・オーダー実行であっても, ペナルティをほとんど隠蔽できないためであると考えられる.

次に, config A, config B の各構成について, 全キャッシュラインに対する損失データプリフェッチによって復帰させたラインの割合を図 10, 図 11 の prefetched lines に示す. また reused lines は全キャッシュラインに占める再利用されたラインの割合を示している. reused line は必要なデータのみ復帰を行う, 理想的な損失データプリフェッチャでの復帰ライン数と考えられ, いずれの構成においても, それと同程度のライン数の復帰で済んでいることが分

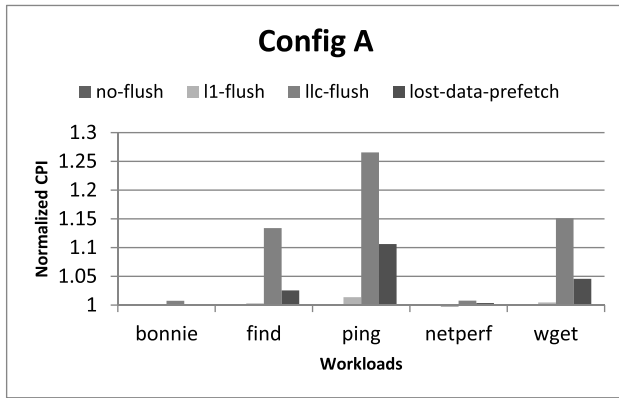


図 8 各方式の性能比較 (config A)

Fig. 8 Performance comparison between each method (config A).

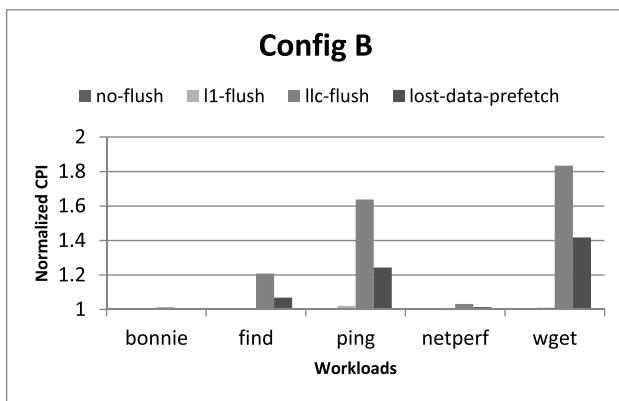


図 9 各方式の性能比較 (config B)

Fig. 9 Performance comparison between each method (config B).

かる。

次に, config A, config B の各構成における, 各方式によるエネルギー削減効果をそれぞれ, 図 12, 図 13 に示す。凡例 llc-static, llc-dynamic, overhead は, それぞれ, LLC のスタティックエネルギー, LLC のダイナミックエネルギー, エネルギーオーバーヘッド (4.1.3 項参照) を表している。横軸は, 各タスクごとに, 各方式を並べて示している。ただし, 方式 no-flush, llc-flush, lost-data-prefetch はそれぞれ, 電源遮断なし, LLC まで電源遮断, LLC まで電源遮断したうえで電源復帰後に損失データ・プリフェッチを行う場合を示している。縦軸は正規化エネルギーであり, それぞれのタスクにおいて, no-flush の場合のエネルギーで正規化されている。なお, これらの結果にはタグのスタティックエネルギーや, プリフェッチでタグ参照を行う際に消費されるダイナミックエネルギーは含まれていないことに注意されたい。

まず, 電源遮断によって大幅にエネルギーを削減できていることが分かる。これは, グラフから分かるように, LLC のエネルギーはスタティックエネルギーが支配的であり, 電源遮断によるスタティックエネルギーの削減効果が

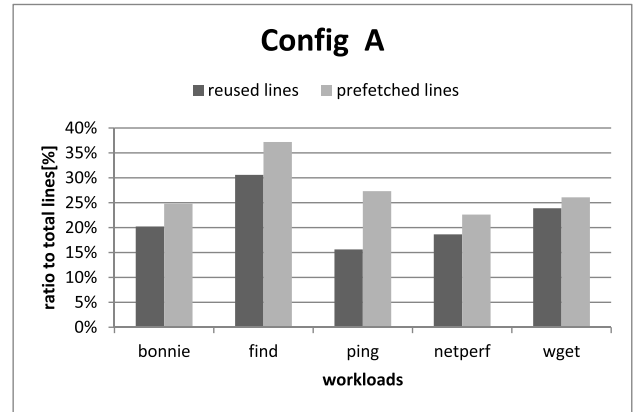


図 10 全キャッシュラインのうち, プリフェッチされたラインの割合 (config A)

Fig. 10 Ratio of prefetched lines to all cache lines (config A).

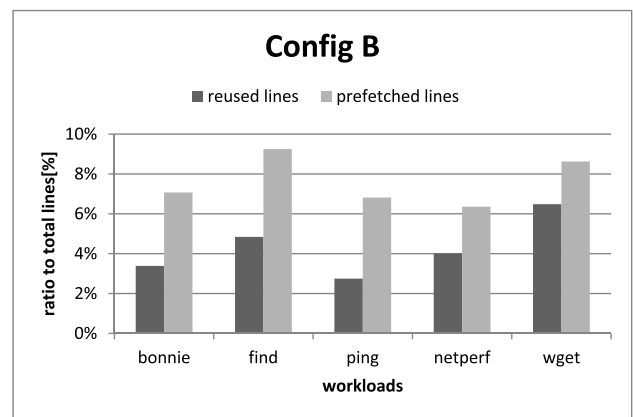


図 11 全キャッシュラインのうち, プリフェッチされたラインの割合 (config B)

Fig. 11 Ratio of prefetched lines to all cache lines (config B).

大きいためである。削減率はタスクのアイドル状態が全稼働時間に占める割合に依存するが, タスクによっては 9 割近くのエネルギーを削減できている。

また, いくつかのタスクでは, 損失データ・プリフェッチを行うことで, 電源遮断のみの場合と比較して, スタティックエネルギーを削減できていることが分かる。これは, 図 8, 9 で示したように, 損失データ・プリフェッチによって電源復帰後に LLC にヒットするようになった結果, アクティブ状態の時間が減少したことによる。

エネルギーオーバーヘッドに関しては, 損失データ・プリフェッチでは再利用されないデータの復帰が存在するため, 電源遮断のみの場合よりも大きくなる。ただし, 前述のように LLC のエネルギーはスタティックエネルギーが支配的であるため, タスクによっては無視できるほど小さくなる。エネルギー全体で比較すると, 損失データ・プリフェッチでは電源遮断のみの場合と比較して, 同程度のエネルギー削減効果が得られている。

次に, config A, config B の各構成における, 性能, エネルギーの散布図を図 14 と図 15 に示す。凡例は図 8, 9

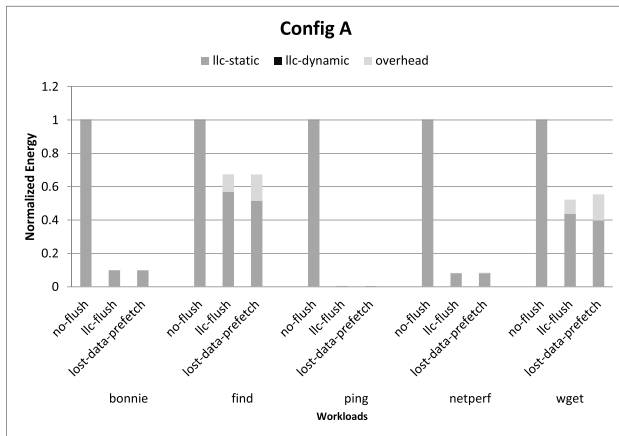


図 12 各方式のエネルギー比較 (config A)

Fig. 12 Energy comparison between each method (config A).

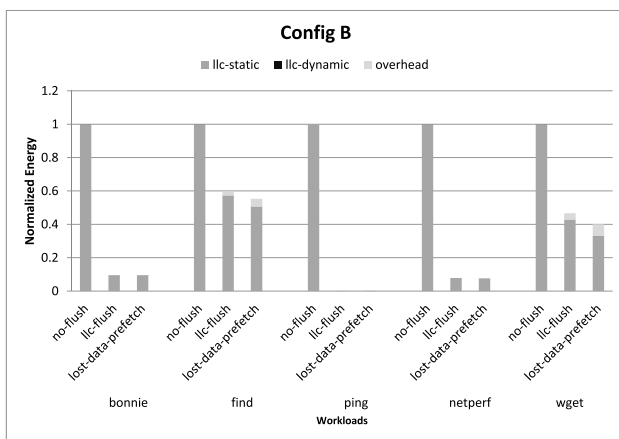


図 13 各方式のエネルギー比較 (config B)

Fig. 13 Energy comparison between each method (config B).

の場合と同様であり、横軸は正規化 CPI、縦軸は正規化エネルギーを示している、no-flush の場合で正規化されている。各々の点はタスクを表しており、同一タスクを破線でつないで示している。

前述のように、損失データ・プリフェッチによって、電源遮断のみの場合と比較して、性能は大幅に向上し、消費エネルギーはほとんど変化しないため、エネルギー対性能はこれによって向上する。たとえば、10%の性能低下を許容して、電源を遮断するという制御を行うと仮定すると、config A であれば、電源遮断を行う機会を損失データ・プリフェッチによって増大させることができ (find や wget も電源遮断ができるようになる)、よりエネルギーを削減することができる。

4.2.2 リソース制約と性能の関係

ここでは、リソース制約が変化した場合に、提案手法がどのような影響を受けるのかについて明らかにする。具体的には、提案手法に大きく影響を与えられ、かつ制約も強いと考えられる、メモリバンド幅や、電源復帰ごとのプリフェッチを行うページ数の上限と提案手法の関係性を明らかにする。

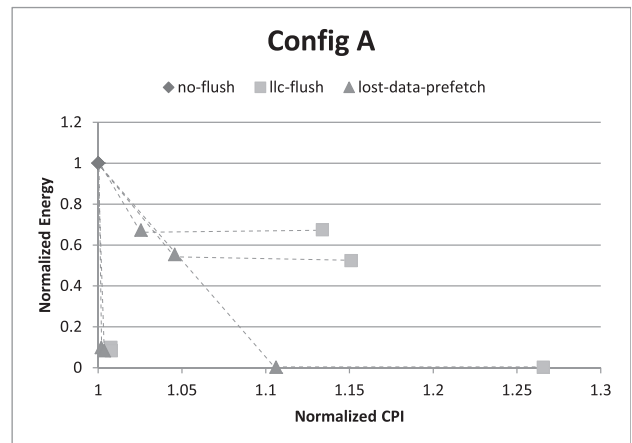


図 14 各方式の性能、エネルギー比較 (config A)

Fig. 14 Performance and energy comparison between each method (config A).

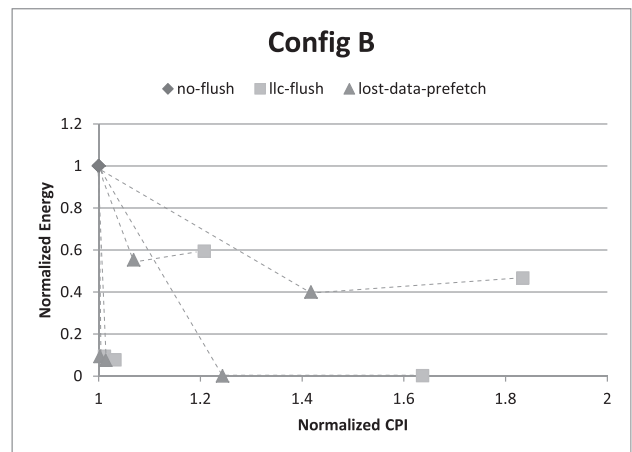


図 15 各方式の性能、エネルギー比較 (config B)

Fig. 15 Performance and energy comparison between each method (config B).

メモリバンド幅制約と提案手法との関係性を明らかにするため、config A, config B の各構成において、損失データ・プリフェッチを行った場合の性能がメモリバンド幅を変化させるとどう変化するかを実験によって確かめた。実験結果は図 16、および、図 17 のようになった。横軸はバンド幅、縦軸は各タスクの正規化 CPI の幾何平均を示しており、CPI は L1, LLC とともに電源遮断を行わない場合の性能で正規化されている。

グラフから、いずれの構成においても 6.4GB/s 以降は律速しており、メモリバンド幅が増大してもほとんど性能は向上していないことが分かる。1.6GB/s では他と比較して性能低下が大きくなっているが、これは 1 チャンネルの DDR-200 の値と等しく現行の PC では採用されていない。このように、本手法は現行のシステムでは、バンド幅制約による影響をあまり受けないことが分かる。

次に、電源復帰ごとのプリフェッチを行うページ数、すなわち、プリフェッチを行ったページを記録しておくテ

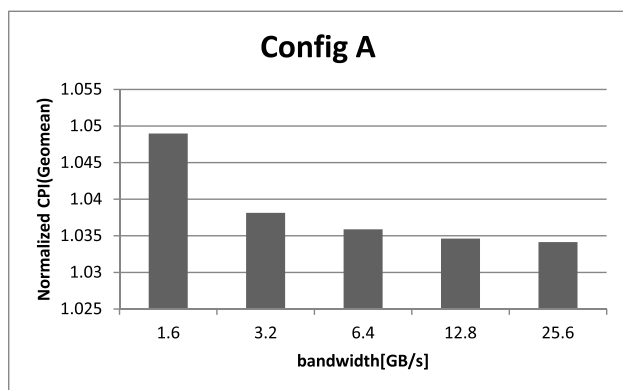


図 16 メモリバンド幅と性能の関係 (config A)

Fig. 16 Relationship between memory bandwidth and performance (config A).

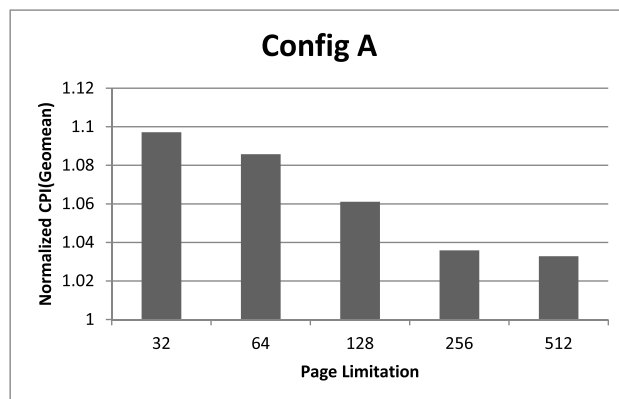


図 18 プリフェッチページエントリ数と性能の関係 (config A)

Fig. 18 Relationship between prefetch page entry size and performance (config A).

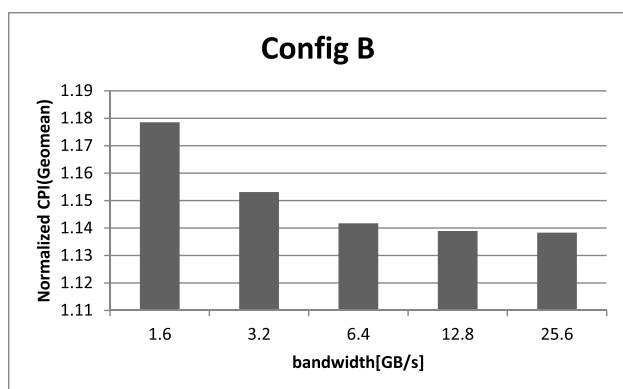


図 17 メモリバンド幅と性能の関係 (config B)

Fig. 17 Relationship between memory bandwidth and performance (config B).

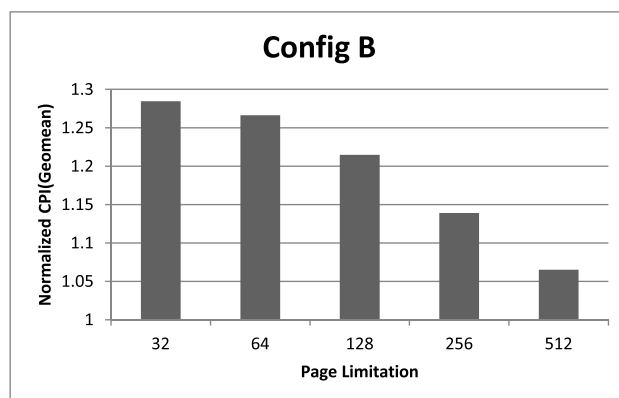


図 19 プリフェッチページエントリ数と性能の関係 (config B)

Fig. 19 Relationship between prefetch page entry size and performance (config B).

ブルのエントリ数を変化させた場合の、性能への影響について述べる。この評価結果は、config A, config Bそれぞれについて、図 18 と図 19 に示す。横軸はエントリ数を表しており、縦軸は図 16, 17 の場合と同様で、各タスクの正規化 CPI の幾何平均を示しており、CPI は L1, LLC ともに電源遮断を行わない場合の性能で正規化されている。

グラフから、config A の場合は 256 エントリ、config B の場合は 512 エントリ以降は律速していると考えられ、各々 256, 512 エントリあれば十分であることが分かる。構成によって律速するエントリ数が異なるのは、キャッシュサイズの影響を受けていると考えられる。

5. 関連研究

これまで、キャッシュのスタティック電力を削減する目的で、様々な研究が行われてきた。

たとえば、アクティブ時、すなわち、プログラムの実行中における電力削減手法に Cache Decay [8], [9] や Drowsy Cache [10], [11] がある。これらの手法では、プログラムの実行中に今後アクセスされないと予想されるキャッシュライン（デッドブロック）を見つけ、そのラインに対応す

るメモリ・セルの電源を遮断する、あるいは、データが保持できる限界までセルへの供給電圧を低下させる。このようにライン単位の電力制御を行うため、これらの方法は SRAM のメモリ・アーキテクチャに対する大幅な変更が必要である。我々の手法は、これらの手法とは異なり、アイドル状態におけるキャッシュのスタティック電力削減を目的としている。また、現行のプロセッサでも採用されているキャッシュ全体をまとめて電源遮断する状況を前提としており、ハードウェアの変更は少ない。

不揮発性メモリをキャッシュに用いる研究もさかに行われている。Wu ら [4] は、キャッシュの各階層に不揮発性メモリを用いる方式を提案しており、それによってキャッシュの大部分のスタティック電力を削減できることを示している。Smullen ら [5] は、データの保持期間を犠牲にすれば STT-MRAM の性能を向上させることができ、キャッシュに用いた場合のプロセッサの性能低下を抑えることができることを示している。しかし、それでも不揮発性メモリは現行のキャッシュに用いられる SRAM よりもアクセス性能が劣るため、CPU の性能低下を引き起こすという問題がある。特に高負荷時の性能低下が問題である。

プリフェッチに関する研究は長年行われてきたが、それらは基本的にはプログラムの実行中におけるキャッシュ・ミスを減らすことを目的としていた。そのため従来のプリフェッチでは、一般に、特定のアクセスパターンやミスパターンを検出し、そこから将来アクセスされるであろうデータを予測する [12], [13]。これらのプリフェッチ方式をキャッシュの電源復帰後にも用いることが考えられるが、電源復帰後につねに何らかのパターンが検出されるわけではない。また、従来のプリフェッチ方式はキャッシュを汚す可能性があるため通常はあまり積極的には行われないが、キャッシュの電源復帰直後のケースでは有効なデータはほとんどないため、本稿で提案したような積極的なプリフェッチを行う方式が有効である。

特殊な状況に着目したプリフェッチに関する研究もいくつか行われている。Cui ら [14] の研究では、コンテキストスイッチ後に頻発するキャッシュミスに対処するため、実行された各々のプロセスに対してアクセス履歴を持っており、コンテキストスイッチ時にアクセス履歴に従ってプリフェッチを行う方式を提案している。また Daly ら [15] の研究では、仮想マシンの切替え直後に頻発するキャッシュミスに対処するため、キャッシュから追い出されたデータの履歴を仮想マシンごとに記録しておき、仮想マシンの切替え時に、その履歴に従ってプリフェッチを行う方式を提案している。これらの方式では、履歴の中には再利用されないデータも含むため、プリフェッチによって無駄なメインメモリアクセスが増えるという問題がある。本手法では、タグアレイに残っている情報を履歴と考えることができ、その中から必要なデータを選択してプリフェッチしている。

6. おわりに

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、電源復帰後に電源遮断によって失われたデータをプリフェッチする方式を提案した。提案手法では、電源遮断の際にタグアレイのデータを残しておき、電源復帰後に、残したタグの情報をもとにアクセスされる可能性が高いデータから順にキャッシュへと書き戻す。

上記の提案手法をシミュレータ上に実装し、その効果を確認した。その結果、電源遮断後にも再利用される損失データのうち、多くのデータについてプリフェッチが成功し、キャッシュの電源遮断による性能低下が大幅に抑えられることが分かった。

今回の評価ではデータを書き戻す順序をキャッシュ・ミスしたページの先頭からとしたが、この順序についてはまだ最適化の余地があるので今後も検討を続けていく予定である。また、web ブラウジングやテキスト入力などの GUI の使用中など、実際のユーザの使用環境に即したタスクの

実行中に発生するアイドル状態を対象に評価を行い、本手法の有効性をさらに確認したいと考えている。

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] Zagacki, P. and Ponnala, V.: Power Improvements on 2008 Desktop Platforms, *Intel Technology Journal*, pp.219-227 (2008).
- [2] Corporation, C.C. and B, R.: Advanced Configuration and Power Interface Specification (2000), available from (<http://www.acpi.info/>).
- [3] Li, S., Chen, K., Ahn, J.H., Brockman, J. and Jouppi, N.: CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques, *International Conference on Computer-Aided Design (ICCAD)*, pp.694-701 (online), DOI: 10.1109/ICCAD.2011.6105405 (2011).
- [4] Wu, X., Li, J., Zhang, L., Speight, E., Rajamony, R. and Xie, Y.: Hybrid cache architecture with disparate memory technologies, *SIGARCH Comput. Archit. News*, Vol.37, No.3, pp.34-45 (online), DOI: 10.1145/1555815.1555761 (2009).
- [5] Smullen, IV, C.W., Mohan, V., Nigam, A., Gurumurthi, S. and Stan, M.R.: Relaxing non-volatility for fast and energy-efficient STT-RAM caches, *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp.50-61 (2011).
- [6] 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 (online), DOI: <http://doi.acm.org/10.1145/2024716.2024718> (2011).
- [7] Muralimanohar, N. and Balasubramonian, R.: CACTI 6.0: A Tool to Understand Large Caches, available from (<http://www.hpl.hp.com/research/cacti/>).
- [8] Kaxiras, S., Hu, Z. and Martonosi, M.: Cache decay: Exploiting generational behavior to reduce cache leakage power, *SIGARCH Comput. Archit. News*, Vol.29, No.2, pp.240-251 (online), DOI: 10.1145/384285.379268 (2001).
- [9] Sankaranarayanan, K. and Skadron, K.: Profile-based adaptation for cache decay, *ACM Trans. Archit. Code Optim.*, Vol.1, No.3, pp.305-322 (online), DOI: 10.1145/1022969.1022972 (2004).
- [10] Flautner, K., Kim, N.S., Martin, S., Blaauw, D. and Mudge, T.: Drowsy caches: Simple techniques for reducing leakage power, *Proc. 29th annual international symposium on Computer architecture, ISCA '02*, Washington, DC, USA, IEEE Computer Society, pp.148-157 (online), available from (<http://dl.acm.org/citation.cfm?id=545215.545232>) (2002).
- [11] Petit, S., Sahuquillo, J., Such, J.M. and Kaeli, D.: Exploiting temporal locality in drowsy cache policies, *Proc. 2nd conference on Computing frontiers, CF '05*, pp.371-377, ACM (2005) (online), DOI: 10.1145/1062261.1062321.
- [12] Nesbit, K.J. and Smith, J.E.: Data Cache Prefetching Using a Global History Buffer, *Proc. 10th International Symposium on High Performance Computer Architecture, HPCA '04*, pp.96-105, IEEE Computer Society

- (online), DOI: 10.1109/HPCA.2004.10030 (2004).
- [13] Chen, T.-F. and Baer, J.-L.: Effective hardware-based data prefetching for high-performance processors, *IEEE Trans. Computers*, Vol.44, No.5, pp.609-623 (online), DOI: 10.1109/12.381947 (1995).
 - [14] Cui, H. and Sair, S.: Extending data prefetching to cope with context switch misses, *IEEE International Conference on Computer Design (ICCD)*, pp.260-267 (online), DOI: 10.1109/ICCD.2009.5413144 (2009).
 - [15] Daly, D. and Cain, H.: Cache restoration for highly partitioned virtualized systems, *IEEE 18th International Symposium on High Performance Computer Architecture (HPCA)*, pp.1-10 (online), DOI: 10.1109/HPCA.2012.6169029 (2012).



有間 英志 (学生会員)

2011 年東京大学工学部計数工学科卒業。2013 年同大学大学院情報理工学系研究科修士課程修了。同年同大学院同研究科博士課程進学。コンピュータアーキテクチャの研究に従事。



薦田 登志矢 (学生会員)

2008 年東京大学工学部計数工学科卒業。2010 年同大学大学院情報理工学系研究科修士課程修了。2010 年より同大学院情報理工学系研究科博士後期課程に在学中。計算機アーキテクチャ、コンパイラの研究に従事。



中田 尚 (正会員)

2002 年豊橋技術科学大学工学部情報工学課程卒業。2004 年同大学大学院工学研究科情報工学専攻修士課程修了。2007 年同大学院工学研究科電子・情報工学専攻博士後期課程修了。博士(工学)。同年奈良先端科学技術大学院

大学情報科学研究科助教。2012 年東京大学大学院情報理工学系研究科特任助教。計算機アーキテクチャとシミュレーションに関する研究に従事。電子情報通信学会, IEEE, ACM 各会員。



三輪 忍 (正会員)

1977 年生。2000 年京都大学工学部情報学科卒業。2002 年同大学大学院情報学研究科通信情報システム専攻修士課程修了。2005 年同大学院情報学研究科通信情報システム専攻博士後期課程単位取得退学。京都大学大学院法学

研究科助手, 東京農工大学工学府特任助教を経て, 2011 年 4 月より東京大学大学院情報理工学系研究科助教。博士(情報学)。計算機アーキテクチャ, 並列処理, 組み込みシステム, 高性能計算の研究に従事。組み込みシステムシンポジウム 2010 優秀論文賞等を受賞。IEEE, 電子情報通信学会各会員。



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, 同助教授を経て, 1996 年東京大学先端科学技術研究センター助

教授。2010 年東京大学大学院情報理工学系研究科教授。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 VLSI システム, 省電力コンピューティング, ハイパフォーマンスコンピューティングの研究に従事。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。IEEE, ACM senior member。