動的最適化機構の電力最適化への適用

請園智玲† 田中清史†

近年，キャッシュメモリで消費されるリーク電力が無視できない程に大きくなってきている．これまでの研究で，キャッシュメモリを部分的に不活性化させ，リーク電力削減を実現するキャッシュメモリ制御機構と不活性化すべきキャッシュ領域を検出する手法が提案されてきた．本稿では，不活性化すべきキャッシュ領域の検出に動的最適化システム HDOS を用いる．HDOS は実発行アドレスを基にメモリアクセスパターンを解析し，時間的局所性の乏しい参照を行うメモリアクセス命令を特定することが可能である．我々はこの解析結果を基に，キャッシュメモリの電力制御を行う手法を提案する．本稿の評価では 19 の SPEC CPU 2000 のアプリケーションを用い，最大で 81%，全アプリケーション平均で 18%の L2 キャッシュのリーク電力削減効果が得られることを示した．

Use of Dynamic Optimization System for Energy Reduction.

TOMOAKI UKEZONO† and KIYOFUMI TANAKA†

Recently, leakage power in cache memories is growing. In past studies, techniques that reduce leakage energy in cache memories by partial inactivation, or techniques that find cache areas to be inactivated were proposed. In this paper, the dynamic optimization system, HDOS is used to detect cache areas to be inactivated. The HDOS can analyze memory access patterns by tracing effective addresses and find instructions that reference data with low-temporal locality. We propose a technique that controls energy according to the analysis. The evaluation results for 19 programs in SPEC CPU 2000 show that the proposed technique can reduce leakage energy in L2 cache memory by up to 81%, or by 18% on average.

1. はじめに

近年のマイクロプロセッサにおいて，メモリウォール問題がパフォーマンス向上の妨げとなっている．これを緩和するために，微細加工技術により得られた高集積化の恩恵をキャッシュメモリの増加に費やす設計傾向がある．このため，現在のマイクロプロセッサは，キャッシュメモリがダイエリアの大部分を占めている．一方で，微細化によるリーク電力消費の増大が無視できなくなってきている．キャッシュメモリはロジック部よりトランジスタ密度が高く，リーク電力はトランジスタの総ゲート幅に比例することから，ダイエリアの多くの部分を占めるキャッシュメモリで消費されるリーク電力の割合は大きい．

キャッシュメモリで消費されるリーク電力を削減するために，キャッシュの電源制御を行う手法が提案されてきた．そのような手法として gated-Vdd[1] と cache decay[2] が挙げられる．gated-Vdd はキャッシュメモリを構成する SRAM セルと GND の間に高い閾値電圧を持つトランジスタを儲けることで電力供給を断つ手段を提供している．電力供給が断たれた SRAM セルはリークが発生しない代わりに，それまで保持していた値が破棄される．論文中では，命令キャッシュに着目しており，動的にキャッシュを 2 分割し，それぞれを gated-Vdd により電力供給状態と非供給状態にすることによって，リーク電流削減制御を行う適用例を示している．この方式では 2 分割という粗粒度の制御を行ったが，cache decay は gated-Vdd の制御を細粒度のキャッシュブロック単位で行う．cache decay は当該キャッシュブロックが dead time であるか否かを判断し dead time である場合に電力供給を断つ．キャッシュブロックが deadtime 中であるかの判断を行うためには各キャッシュブロックにカウンタ回路を持たなければならない．

cache decay のように，ハードウェアによる細粒度の電源制御には制御のためのハードウェア資源の追加が必要となる．このことから，追加ハードウェアを必要としない電源制御手法 Software Self-Invalidation[3] が提案された．Software Self-Invalidation は従来研究の Self-Invalidation[4][5] をキャッシュのリーク電力削減のために応用した提案である．元来，Self-Invalidation はマルチプロセッサ環境でキャッシュコヒーレント維持のための手法である．Self-Invalidation は他のプロセッサからコヒーレント維持のための無効化要求が来

† 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

る前に，予測によってあらかじめキャッシュブロックを無効化する．キャッシュブロックの無効化は当該ブロックの保持するデータを破棄する行為であり，無効化された時点で当該ブロックの電力供給を遮断しても構わないことから，電源制御手法に応用することができる．Software Self-Invalidation では，プログラム上，最後のデータアクセスを行う命令をプログラマが認識し，最後のデータアクセスを行う命令を last touch 命令と呼ばれる特殊なロード／ストア命令に置き換えることによって invalidation かつ電源 OFF が実現される．これにより，cache decay の問題であったカウンタ等のハードウェアの追加の必要がなくなる．

Software Self-Invalidation は last touch 命令の置換候補を見つけるためにメモリアクセストレースを使用する．事前に対象プログラムを実行して発生した全てのメモリアクセスのトレースを収集し，それを解析することによって該当メモリブロックを最後に参照する命令を知ることができる．しかしながら，対象プログラムの実行時間が長い場合，収集するトレースは膨大なものとなり，収集行為が非現実的となる．また，解析とそれに基づくプログラムの修正を人手で行うことはソフトウェア開発効率上，極めて難しい．

この先行研究に対して，本稿は，動的最適化システム HDOS を電力最適化のプロセス (トレース収集，解析，バイナリ修正) に適用することによって，Software Self-Invalidation の欠点を克服する．HDOS は我々の以前の研究で提案したシステムである[6)7)]．HDOS は実行時のハードウェア情報を用いてネイティブバイナリをオンタイムに最適化することができる．これまでの研究で我々は HDOS をプリフェッチ最適化に適用した場合，ハードウェアプリフェッチの履歴テーブルに要するハードウェア量を削除してハードウェアプリフェッチと同等の効果が得られることを示した[7)]．このハードウェア削減効果はハードウェアプリフェッチに限らず，CPU 内の他のハードウェアにも同様に適用できる．本稿では，キャッシュメモリの電力最適化を HDOS の最適化適用例の 1 つとして紹介する．HDOS の概要は次節で述べる．

2. HDOS

本節では，本稿の提案に使用する動的最適化システム Hybrid and Dynamic Optimization System (HDOS) の概要について述べる．

HDOS はシステムユーザにトラップ発生条件を定義可能にするハードウェア User Definable Trap (UDT) と OS のトラップハンドラとして提供される最適化ルーチンの 2 つのコンポーネントで構成される．HDOS において UDT には 2 つの役割がある．1 つ目はトラップを発生させ，プロセッサの実行を最適化対象バイナリから最適化ルーチンに移す機能である．従来の CPU デザインでは，CPU が提供するトラップの機能は例外や外部割込み，システムコールなど，CPU の設計者によって定められた原因の場合のみ，トラップハンドラへ制御の遷移を許す．これに対し，UDT はシステムユーザによって指定されたトラップ条件が満たされた場合に，システムユーザによって指定されたトラップハンドラを起動する．2 つ目は起動した最適化ルーチンにトラップ発生時のハードウェアの実行時情報を知らせる機能である．例えば，本稿で提案するキャッシュの電力最適化を行う場合，最適化ルーチンがプログラム実行中に発行するアドレスを知る必要がある．UDT は最適化ルーチンからアクセスできるレジスタにこのような実行時情報をセットすることができる．

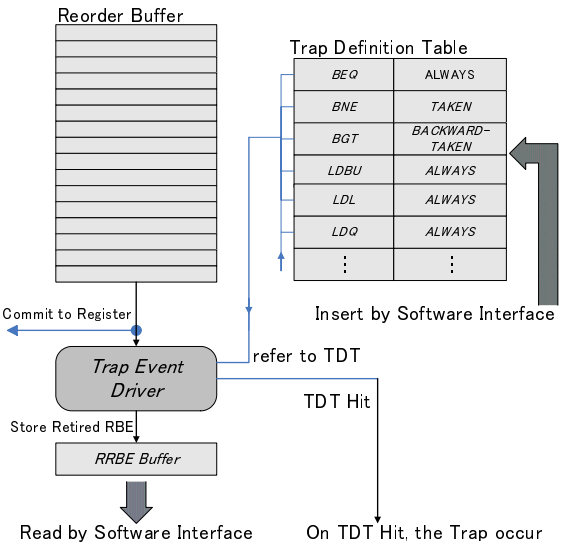

図 1 User Definable Trap (UDT) ハードウェア．

UDT のハードウェアブロックを図 1 に示す．

図 1 の Trap Definition Table は，命令セットに用意されたソフトウェアインタフェースを用いて，システムユーザによりトラップ発生条件をセットされる．トラップ発生条件には 2 つの対になる情報が記述される．1 つは命令の指定．もう 1 つは指定された命令の動作である．例えば，3 つ目のトラップ発生条件は BGT という分岐命令が Taken で動作し，飛び先が自己の PC より下位番地に飛ぶ場合にトラップを発生させる定義である．UDT は Reorder Buffer のコミットエントリの解析と，Trap Definition Table の参照を行うことで，システムユーザの指定したタイミングで

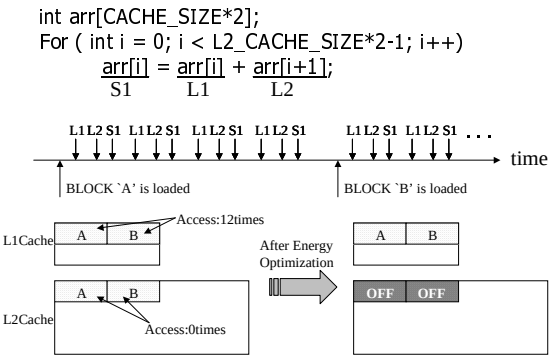

図 2 本研究が対象とする配列アクセス時の電力最適化の適用.

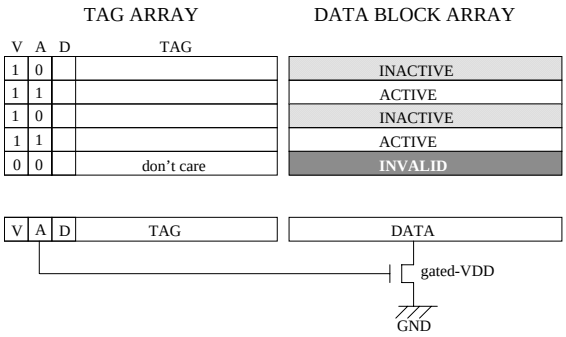

図 3 電源制御機構をもつキャッシュブロック図.

トラップを発生させることができる.

図 1 の RRBE(Retired Reorder Buffer Entry) Buffer にはトラップ発生の原因となった Reorder Buffer のコミットエントリが格納され，命令セットに用意されたソフトウェアインタフェースによって最適化ルーチンから参照される．最適化ルーチンはこの RRBE Buffer を解析することによって実行時情報を得ることが出来る.

HDOS を伴う実行 (1 回目の実行) は最適化対象プログラムに OS が割り込んで最適化処理を実行するため，オーバヘッドが存在するが，最適化後の実行 (2 回目以降の実行) では UDT を OFF にした状態もしくは UDT を持たない CPU 上で実行可能であるため，このオーバヘッドは存在しない．また，本研究の最適化アルゴリズムは以前の研究で示したデータプリフェッチ最適化のアルゴリズムと酷似するため，評価で示すアプリケーションにおけるオーバヘッドは同じ傾向を示すと考察する[7].

3. キャッシュ電力制御方式

本研究の目的は L2 キャッシュのリーク電力削減である．図 2 に本研究が対象とする配列アクセス時の電力最適化の適用事例を示す.

図中で示すプログラムでは，キャッシュブロック A はキャッシュブロック B がロードされるまでアクセスが頻発するが，その後はアクセスされることは無い．この場合，L1 キャッシュでは，ブロック A，ブロック B 共に 12 回アクセスされる (図の例では L1，L2 ともにブロックサイズが 16 バイト，一回のアクセスが 4 バイトであるとする)．この時，L2 キャッシュに着目すると，キャッシュミス時に L2 キャッシュにロードしてから一度もアクセスされておらず，この先アクセスされる予定も無い．このようなメモリ参照では，逐次アクセスや近傍要素同士の演算が原因となり，短期的なデータ再利用性のある L1 キャッシュブロックのヒットが頻発する一方で，長期的なデータ再利用性を見込んだ L2 キャッシュのヒットが見込めない．本研究ではこのようなアクセスを対象に電力最適化を行う．図 2 で示す L2 ブロック A と B はロードされた時点から時間的局所性を見込めないため，L2 キャッシュに存在する意義がないことから，電力供給を断ちリーク電力の低減を行っても性能に影響を与えない．本稿では，キャッシュロード時に L1 キャッシュにデータを読み込み，L2 キャッシュにデータを読み込まずに対象ブロックの電力供給を断つ (ただしタグ情報は生成，保持される) ロード／ストア命令を提案する.

提案するキャッシュメモリの電力最適化には，HDOS の他に以下 3 つの要素技術が必要となる.

(1) 電源制御が可能なキャッシュメモリ
(2) ソフトウェアインターフェース
(3) 最適化アルゴリズム

以降はこの 3 つの要素技術の詳細を個々に述べる.

3.1 電源制御可能キャッシュメモリ

本研究では短期的に時間的局所性の高いキャッシュブロックの参照を L1 キャッシュに任せ，長期的に見た場合に，時間的局所性の低い L2 キャッシュブロックの電力供給を止める操作を行う．本研究で対象となる電源制御機能を備えたキャッシュ機構を図 3 に示す.

L2 キャッシュの電源制御を行うために，ブロックの電源 ON ／ OFF を示す 1 ビット (図 3 の A:Active ビット) を制御情報に追加する．A ビットは gated-Vdd の ON/OFF の制御に直結され (図 3 の下部)，A ビットが 1 ならばキャッシュブロックに電力供給がなされている状態で，逆に 0 なら電源供給が止められている状態となる.

対象とする L2 キャッシュは L1 キャッシュのデータの保持状態に関係なく，L2 ブロックの電力供給を断つ．各ブロックの電力供給制御は gated-Vdd を想定するため，ブロックの保持する内容は破棄される．しか

しながら，インクルージョンプロパティを守らない場合，マルチプロセッサ環境で，キャッシュコヒーレント問題の解決が難しくなる．そこで，対象キャッシュはブロックのデータ領域の電力供給が断たれた場合でも，タグは放棄せずにブロックの有効ビットを立て続け，インクルージョンプロパティを保全する．(図 3 の 1 番目と 3 番目のブロック)

対象とする L2 キャッシュは，データを保持せず，タグを保持する特性から，タグヒットをしたが，電力供給は断たれてデータはすでに破棄されている状態 (ミスシャットダウン) が存在する．ミスシャットダウンは有効ビット (V ビット) と A ビットの参照により検出可能である．ミスシャットダウンの検出時，電力供給は再開 (A ビットをアサート) され，再度主記憶からメモリブロックは当該ブロックエントリに格納される．

また，図 3 の 5 番目のブロックのように，invalid であるブロックはタグヒットせず，値を保持する必要が無いため，A ビットは 0 にすべきである．本稿における評価では，V ビットを 0 にする場合は同時に A ビットを 0 にするキャッシュ制御を想定して行った．

マルチプロセッサ環境☆を想定しない場合は，A ビットは削除可能である．その場合，V ビットを gated-Vdd の制御に用いることにより，前述のキャッシュブロック制御と同様の効果が得られる．

3.2 ソフトウェアインタフェース

A ビットを持つ L2 キャッシュを想定した場合に，A ビットをソフトウェアから操作する手段が必要となる．本研究では，A ビットの操作を行うロード／ストア命令を導入した．提案するロード／ストア命令は命令セットに備わるロード／ストア命令とは別に用意される．提案するロード／ストア命令を本稿では，Inactive-Block(IB) ロード／ストア命令と呼ぶ．IB 命令は命令実行パイプライン上で元来命令セットに備わるメモリアクセス命令と同じ動作をする．唯一違う点は，IB 命令が原因となって L2 キャッシュミスが発生した場合に，置換対象のブロックの A ビットを 0 にする機能を有していることである．本節冒頭で示した 2 レベルキャッシュの例では，IB 命令が原因で L2 キャッシュミスが起きた場合，まず，L2 キャッシュのタグが更新される．この時，A ビットは 0 にセットされるので，置換対象ブロックの電力供給は断たれる．その後，主記憶から読み出されたメモリブロックが到着すると，対象ブロックは L2 キャッシュには格納されずに (電力供給が断たれているため)L1 キャッシュにのみ格納される．ソフトウェアはこの IB 命令と通常のメモリアクセス命令を使い分けることで，L2 キャッシュにロードするかしないかを制御することができる．

3.3 最適化アルゴリズム

HDOS において最適化ルーチンはトラップハンドラとして実装されることは 2 節で示した．以降は，IB 命令を扱う最適化ルーチン上のアルゴリズムを述べる．IB 命令はロード／ストア機能を備えるので，最適化ルーチンの行う操作は最適化対象バイナリ上のロード／ストア命令を IB 命令に置換することのみである．置換対象メモリアクセス命令を見つけるために，最適化ルーチンは最適化対象バイナリ上のどのメモリアクセス命令が長期的に見た場合に低い時間的局所性を示すかを解析しなければならない．

最適化アルゴリズムはまずプログラムのループ構造に着目する．これは，ループ実行中は規則性のあるメモリアクセスを発行することが多く，ループ実行のごく一部のメモリアクセストレースを収集することのみで，対象メモリアクセス命令のアクセスパターンを高確率で予測できるからである．低い時間的局所性を示すアクセスパターンの 1 つとして，本節冒頭で述べた配列アクセスが挙げられる．このアクセスパターンを見つけるためのアルゴリズムを述べる．

アルゴリズムには以下の 3 段階のステップがある．

(1) ループ検出
(2) メモリアクセス履歴テーブルの作成
(3) 履歴テーブルの検査とバイナリコードへの修正

UDT によって駆動される最適化ルーチンは現在どのステップの処理中であるかで振る舞いを変える．プログラム実行の大半はループ実行で占められる傾向があることから，HDOS は全ての処理の最初にループ検出を行う (ステップ 1)．ステップ 2 はステップ 1 で検出したループを何度か実行し，発生する全てのメモリアクセスを集計して履歴テーブルを作成する．ステップ 3 は履歴テーブルを解析して，IB 命令に置換する命令の候補を選別し，バイナリコードに修正を加える．ステップ 3 が終了するとステップ 1 に遷移し，プログラム実行で検出できる全てのループに対して同様の最適化を施す．以降は各ステップの詳細を述べる．

3.3.1 ステップ 1:ループ検出

ステップ 1 では，まず，HDOS は UDT のトラップを発生条件設定を後方分岐の Taken 実行にセットする．この UDT 設定では，UDT は後方分岐が実行される毎に最適化ルーチンを呼び出すことになる．

ステップ 1 の主な処理は後方分岐リストの作成であ

☆ 厳密には，コヒーレンスキャッシュの機能を利用した周辺 DMA 転送を含む．

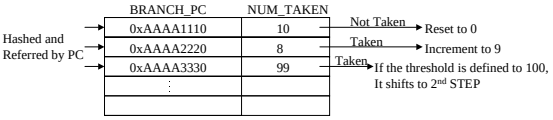

図 4 後方分岐リストの例.

る．後方に分岐する命令はループバック分岐命令になる可能性があるため，後方分岐リストを生成することでループバック分岐命令を特定することができる．

作成する後方分岐リストを図 4 に示す．テーブルは単純な構成で，後方分岐が連続して Taken で実行された回数 (図中の NUM_TAKEN) を記録している．

最適化ルーチンは呼び出された後，専用命令を用いて RRBE Buffer を読み出し，そこから得られた後方分岐命令の PC 値を用い，既存の後方分岐リストを検索する．テーブルは PC 値をキーにしたハッシュ構造となっており，高速に検索できる．図 4 の 1 番目のエントリは Not Taken で最適化ルーチンが呼び出された場合の例である．後方分岐リストは連続した後方分岐回数を記録するテーブルであるので，もし Not Taken でリストにヒットした場合は，NUM_TAKEN を 0 にリセットし，トラップから復帰する．図 4 の 2 番目のエントリは Taken で最適化ルーチンが呼び出された場合の例である．この場合は該当するエントリの NUM_TAKEN をインクリメントしてトラップから復帰する．図 4 の 3 番目のエントリは最適化ルーチンのステップを次に進める原因となるエントリの例である．最適化ルーチンは後方分岐命令が一定閾値以上連続で Taken 実行された場合に，ステップ 2 に遷移する．例えば，閾値を 100 に設定した場合，NUM_TAKEN が 99 から 100 にインクリメントされた場合に，次回以降のトラップ起動はステップ 2 の処理となる．また，一度 NUM_TAKEN が 100 を超えたエントリは 2 度とステップ 2 への遷移の原因とはならない．

3.3.2 ステップ 2: メモリアクセス履歴テーブルの作成

ループバック分岐が検出された後，UDT 設定は以前の設定に加えて，ロード/ストア命令が実行された後にトラップを発生する設定が加えられる．最適化ルーチンは該当ループバック分岐命令が指定された回数分 taken 実行されるまで，このループで発生するメモリアクセスの解析を行う．本稿では，この作業を観測フェーズと呼ぶ．図 5 は観測フェーズで作成されるメモリアクセス履歴テーブルの例を示す．

図の履歴テーブルは 4 つの項目を持つ．図中の項目を左から順に示す．MEM_INST_PC はメモリアクセス命令の PC，LAST_ADDR は該当メモリアクセス

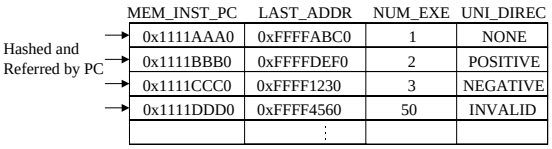

図 5 メモリ参照履歴テーブルの例.

命令が最後に発行したメモリアドレス，NUM_EXE はメモリアクセス命令の実行回数，UNI_DIREC はメモリアクセスパターンを示すフラグである．履歴テーブルは後方分岐リスト同様，PC 値をキーにしたハッシュ構造となっており，高速に検索できる．図 5 の 1 番目のエントリは作成された直後の状態である．初めて観測されるメモリアクセス命令は履歴テーブルにエントリが作成され，LAST_ADDR にその時アクセスしたアドレス，NUM_EXE に 1，UNI_DIREC にはまだアクセスパターンが判明していない NONE がセットされる．図の 2 番目のエントリは 2 回目のメモリアクセス命令が実行されたときのエントリの更新結果である．エントリの更新は以下の手順で行われる．

(1) 今回アクセスしたアドレスから該当エントリの LAST_ADDR を減算し結果が正か負かを判断する．
(2) 該当エントリの UNI_DIREC が NONE なら UNI_DIREC に正 (POSITIVE) か負 (NEGATIVE) のフラグをセットする．もし正でも負でも無いゼロの場合 (前回と同じアドレスにアクセスした) は無効 (INVALID) フラグをセットする．
(3) 該当エントリの UNI_DIREC に正か負のフラグがセットされているなら，減算結果と比較する．異なるフラグであるなら，無効 (INVALID) フラグをセットする．
(4) 今回アクセスしたアドレスで LAST_ADDR を更新する．
(5) NUM_EXE をインクリメントする．

3 番目と 4 番目のエントリは 3 回目以降の更新の結果である．3 番目のエントリは 2 回目の更新で負のフラグを立てて，3 回目のアクセスでも負の結果を得たため，UNI_DIREC は変わっていない．4 番目のエントリは 50 回更新される何処かのタイミングで，UNI_DIREC の不一致があり，無効にセットされている．以上の更新は観測フェーズ終了まで続けられる．

観測フェーズはステップ 1 で観測した後方分岐命令が一定回数実行されるまで続けられる．観測フェーズが終了した後，最適化ルーチンはステップ 3 に処理を遷移する．

表 1 SimpleScalar シミュレーションパラメータ

issue width	4
decode width	4
ruu size	16
lsq size	8
dl1 size	64KB, 32B block, 4-way, 1-cycle access
il1 size	64KB, 32B block, 4-way, 1-cycle access
ul2 size	2MB, 32B block, 8-way, 6-cycle access
memory access latency	[first]:120 [inter]:12cycles
memory access bus width	8B

3.3.3 ステップ 3: 履歴テーブルの検査とバイナリコードへの修正

観測フェーズが終了すると，最適化ルーチンは履歴テーブルをテーブルウォークし，検査を行う．テーブルの中から時間的局所性の低い命令を見つける目的で，NUM_EXE が観測フェーズのループバック数以上☆であり，かつ，UNI_DIREC が POSITIVE もしくは NEGATIVE である命令を検索する．条件に一致するエントリが見つかった場合は，MEM_INST_PC の指す番地の命令を IB 命令に置換する．

HDOS は部分的なメモリアクセストレースのみを扱うため，ステップ 1 で検出したループのスコープを超えるような，長期的なデータ再利用性の有無を解析することは難しい．しかしながら，大規模なデータセットへのアクセスは容量性ミスを誘発し，有効な時間的局所性を持たないアクセスである可能性が高いことは予測できる．この予測のために NUM_EXE を命令抽出条件に加え，大規模なデータセットへのアクセスを行うメモリアクセス命令に適用箇所を限定している．

4. 評　　価

本節では，HDOS の L2 キャッシュにおける消費電力削減効果を評価をする．

4.1 評価環境及び評価方法論

シミュレーションによる評価は Simple Scalar3.0[8] で行った．対象命令セットは Alpha 命令セット[9] である．シミュレーションの方式は sim-outorder で行った．シミュレーションパラメータを 1 に示す．

評価は SPEC CPU 2000[10] ベンチマークの中から 19 のアプリケーションを選択し，行った．バイナリは SimpleScalar のサイト[11] よりプリコンパイルバイナリを取得し，評価に使用した．全てのアプリケーションは 20 億命令実行完了まで実行した．

UDT ハードウェアとトラップハンドラ (最適化ルーチン) はシミュレータコード内に直接記述し，シミュレータ上でバイナリ修正を行った．このため，シミュレーション上，最適化によるソフトウェアオーバヘッドは無い．本計測では，このシミュレータ上の実行をバイナリ修正後の実行の評価とみなした．HDOS の最適化時に使用するパラメータは，後方分岐ループバック回数閾値を 100 回，メモリアクセスを観測するためのループバック回数を 100 回とした．

対象とする電源制御可能な L2 キャッシュのシミュレーションを行うために，SimpleScalar のキャッシュシミュレーションモデルに修正を加え，IB 命令で A ビット操作が可能な L2 キャッシュを仮定してシミュレーションを行った．

SimpleScalar は命令単位のシミュレーションを行い，クロックサイクル単位でのシミュレーションを行わないため，正確な消費電力評価を行うためには広範囲にわたるシミュレータコードの修正を要する．本稿では，正確な消費電力評価を行わず，約 1000 クロックサイクル毎にアクティブな (A ビットが 1 である)L2 キャッシュブロック数を算出し，その平均をとることで，おおよその消費電力削減効果を見積もる．

4.1.1 シミュレーション結果

図 6 に平均アクティブブロック率を示す．252.eon, 256.bzip2，189.lucus に関して，平均アクティブ率が 50%以下となっており，L2 キャッシュの半分以上の領域の電源供給が断たれている．特に 252.eon に関して，19%以下のアクティブブロック率が観測された．252.eon は時間的局所性の低いデータを主に取り扱うアプリケーションであり，HDOS がそれを検出できていたため，積極的に電源制御が働いたと考えられる．それ以外のアプリケーションに関しても，90%〜50%の平均アクティブ率が観測された．INT アプリケーションの平均アクティブ率は 68.50%，FP アプリケーションの平均アクティブ率は 81.65%，全 19 アプリケーションの平均アクティブ率は 74.73%である．

本稿の L2 キャッシュ電力削減手法はインクルージョンプロパティを保全するため，キャッシュのブロックエントリの電力供給を断っても，タグはあたかもそのキャッシュデータがあるかのように存在し続ける．LRU 値も電力制御に関係なく，変動しない．これは，電力制御する場合と，しない場合で，L2 キャッシュミス率が同じであることを意味している．このため，本提案が実行性能を向上させることは無い．逆に，ミスシャットダウンとライトバックによる実行性能の低下

☆ NUM_EXE はステップ 1 で検出されないほどループバック回数の少ないループを内包する場合に，観測フェーズのループバック回数より上回る

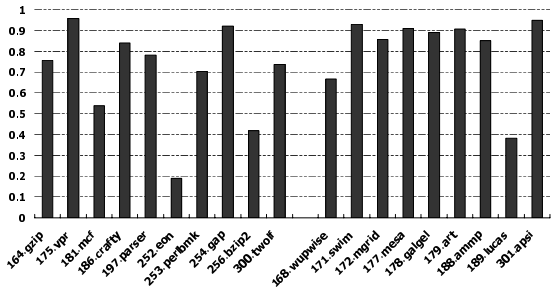

図 6 SPEC CPU 2000 における平均アクティブブロック率.

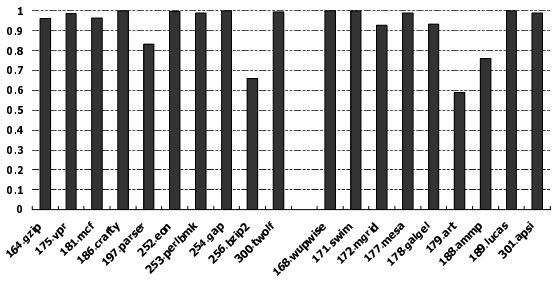

図 7 SPEC CPU 2000 における IPC 性能比.

は有りえる．これら要因による IPC 性能の低下を図 7 にを示す．図は電源制御を行った場合の IPC 性能に対する電源制御を行わない場合の IPC 性能の比である．

197.parser，256.bizp2，179.art，1188.ammp で，性能低下が著しい．それ以外のアプリケーションでは目だった性能比が見られず，性能比はおおむね 90%以内に収まっている．INT アプリケーションの平均 IPC 性能比は 93.85%，FP アプリケーションの平均 IPC 性能比は 90.99%，全 19 アプリケーションの IPC 性能比は 92.47%である．全体平均で，性能低下は 8%以内に抑えられている．

本提案は L2 キャッシュのリーク電力の削減を目的とする．リーク電力は時間に比例して増大することから，図 6 で示した通り，実行性能の低下，つまり，指定された命令数を実行し終わるまでのクロックサイクル数の増加はリーク電力の増大につながる．そこで，L2 キャッシュのリーク電力の指標として，電源制御を行った場合のリーク電力消費に対する電源制御を行わない場合のリーク電力消費の比を図 8 に示す．

リーク電力比は INT アプリケーションで平均 72.99%，FP アプリケーションで平均 93.42%，全 19 アプリケーションの平均で 82.67%となった．179.art と 188.ammp を除いたアプリケーションでは図 6 で示した平均アクティブ率と概ね同じ傾向であることがわかる．一方，179.art と 188.ammp は比が 1 を超えていることから，明らかにリーク電力は増加している．

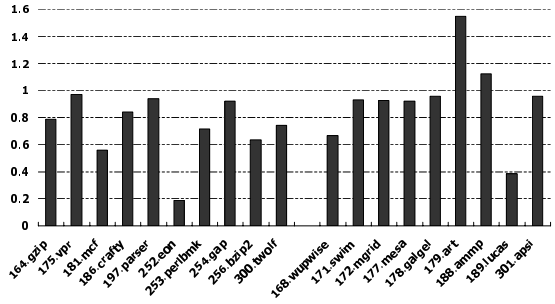

図 8 SPEC CPU 2000 における L2 キャッシュ上の消費リーク電力比.

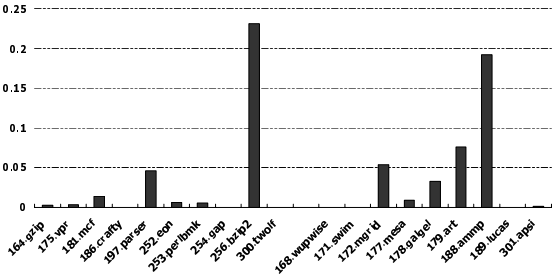

図 9 SPEC CPU 2000 におけるミスシャットダウン率.

特に，179.art のリーク電力比は 1.5 倍に達している．このリーク電力増大を説明するために，ミスシャットダウン率を図 9 を示す．ミスシャットダウン率はミスシャットダウン回数／ L2 参照回数で計算した．

ほとんどのアプリケーションで，ミスシャットダウン率が 5%以下であるのに対して，256.bzip2，179.art，188.ammp において，高いミスシャットダウン率が観測されていることがわかる．この 3 アプリケーションに着目して，図 6 の平均アクティブブロック率を見ると，256.bzip2 では，ほぼ 60%のブロックの電力供給が遮断されているのに対して，179.art，188.ammp では，電力供給が遮断されているブロックは 15% 未満である．図 9 から考察すると，256.bzip2 では，時間的局所性の低いブロックが多数存在し，そうでないブロックも一定数存在した．結果として HDOS はそれらを区別できずに電源供給を断ち，ミスシャットダウンが頻発したと考えられる．これによる性能低下は図 7 において顕著に現れているが，それを上回るほど，平均アクティブブロック率が低いため，リーク電力比は低い値を示した．一方，179.art，188.ammp では，平均アクティブブロック率が 15%未満であるため，ミスシャットダウンが原因となる性能低下によるリーク電力増大を打ち消すことができず，図 8 の結果となったと考えられる．

本稿の結果では，252.eon や 189.lucas のように，

極端に良い傾向を示すアプリケーションがある反面，179.art や 188.ammp のように明らかにリーク電力消費が増大する結果が存在した．これは図 9 で示したミスシャットダウンが支配的要因となっており，本稿で示した HDOS の解析では，これらのアプリケーションが要求するアクセスパターンを見極めるためには不十分であるといえる．この主たる要因として考えられるのが，ロード／ストア命令間のキャッシュブロックの共有である．3.3 節で示した最適化アルゴリズムによるメモリアドレストレースの解析は個々のロード／ストア命令のキャッシュミスでロードされるブロックの時間的局所性のみを対象としていたため，ループの中に存在する複数のロード／ストアのアクセスが交差して，長期的なデータ再利用性を作り出すケースを解析していない．179.art や 188.ammp ではそのケースが顕著であったのではないかと考察する．今後，HDOS の時間的局所性解析を改善する必要がある．

5. おわりに

本稿では，動的最適化システム HDOS を L2 キャッシュの低消費電力化に適用する手法を示した．提案手法の目的は省ハードウェア資源によるキャッシュメモリの低消費電力化である．HDOS をキャッシュのリーク電力削減に適用することによって，事前実行でメモリアクセストレースを収集する必要が無くなり，ソフトウェア生産性を向上させている．

本研究では，HDOS によるキャッシュのリーク電力削減の手段として，L2 キャッシュにロードせず，L1 キャッシュにロードする IB 命令を提案し，HDOS の最適化アルゴリズムとして長期的なデータ再利用性の無いアクセスを発生させる命令を特定するアルゴリズムと，IB 命令を使用したバイナリの修正を提案した．

本稿の評価では，SPEC CPU 2000 の中から 19 のアプリケーションを用い，最大で 81%，全アプリケーション平均で 18%の L2 キャッシュのリーク電力削減効果が得られることを示した．個々のアプリケーションの結果から，リーク電力削減に良い傾向を示すアプリケーションが存在する一方で，逆にリーク電力が増大するアプリケーションが存在することが確認された．本稿では，ロード／ストア命令間のキャッシュブロック共有に原因があると考察した．

HDOS は部分的なメモリアクセストレースによる予測であるため，ミスシャットダウンが存在する．しかしながら，HDOS の特長である，部分的なメモリアクセストレース収集方法と，最適化の自動化による生産性の向上はソフトウェアによるキャッシュの電源制御を現実的なものにしている．

今後の課題として，ミスシャットダウンを減らすための HDOS の最適化アルゴリズムの洗練が挙げられる．また，今回の評価はアクティブなブロック数のみに着目したが，今後はキャッシュブロックの ON／OFF 時に発生する動的電力とミスシャットダウン時のリフィルに伴う動的電力を考慮して評価していく予定である．

謝辞 本研究の一部は科研費若手研究 (B)(20700045)「低消費電力高機能リコンフィギュラブルメモリシステムの研究」の一環として行われた．

参 考 文 献

1) Powell, M., Yang, S.H., Falsafi, B., Roy, K., Vijaykumar, T.N., “Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories”,Proc. of ISLPED, pp.90-95, 2000.
2) Kaxiras, S., Hu, Z., Martonosi, M.” Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power”, Proc. of 28th ISCA, pp.240-251, 2001.
3) Kiyofumi Tanaka，Takenori Fujita,“ Leakage Energy Reduction in Cache Memory by Software Self-Invalidation ”, Proc. of 12th Asia-Pacific Computer Systems Architecture Conference (ACSAC), LNCS 4697, ISBN 3-540-74308-1, Springer, pp.163-174, 2007.
4) Lebeck, A.R., Wood, D.A.: Dynamic Self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors, Proc. of ISCA, pp.48-59, 1995.
5) Lai, A.C., Falsafi, B.: Selective, Accurate and Timely Self-Invalidation Using Last-Touch Prediction, Proc. of ISCA, pp.139-148, 2000.
6) 請園 智玲, 田中 清史, “データプリフェッチ最適化のためのバイナリ変換手法”, 先進的計算基盤システムシンポジウム (SACSIS 2008), pp.187-194, 2008.
7) 請園 智玲, 田中 清史, “バイナリ変換によるデータプリフェッチのためのハードウェア削減手法”, 情報処理学会論文誌 コンピューティングシステム (ACS 28 号) Vol.2 No.4 1-14 Dec. 2009.
8) D.Burger, T.Austin, and S.Bennett. Evaluating future microprocessors: The simplescalar toolset. Tech Report CSTR-96-1308, Univ. of Wisconsin, CS Dept., July 1996.
9) Alpha Architecture Reference Manual, THIRD EDITION. ALPHA ARCHTECTURE COMMITTEE, Digital Press, ISBN 1-55558-202-8.
10) http://www.spec.org
11) http://www.simplescalar.com