C-023

サクセッサ情報を用いた命令フェッチ電力削減の検討

Considerations of Reduction for Instruction Fetch Energy Using Successor Information

請園　智玲 †
Tomoaki Ukezono

1. まえがき

1.1. 高連想度キャッシュメモリ

Strong ARM[1] や XScale[2] などの組込みプロセッサはキャッシュミス削減を狙う目的で，32Way などの比較的高い連想度でキャッシュメモリを実装する．これはハードウェアサイズと引き換えにキャッシュメモリの効率的な利用を優先した構成であり，半導体の高集積化が進むなか，高周波数駆動が必要とされない組込みプロセッサにとって，有効なキャッシュ構成法と言える．一方で，タグ検索には CAM を利用することから，高連想度のキャッシュメモリには CAM による動的電力のオーバヘッドが存在する．タグ検索に要する動的電力はタグのビット幅と連装度に応じて増加することから，高連想度キャッシュはその両方の電力増加要因を併せもつ．

これまでに，このような高連想度のキャッシュの動的電力の削減手法がいくつか提案されている．最後に参照された Way が次に参照される Way であると予測し，precharge を制御してそれ以外の Way への参照を制限する手法 [3] や，タグ検索の結果をキャッシュ内に記憶し，タグ参照をバイパスする手法 [4] などがある．これらの手法は命令キャッシュに対する手法であり，命令参照の局所性をもとにした予測にもとづいている．

1.2.L0 命令キャッシュの動的電力の削減効果

命令参照に限定する場合，予測をもとにせず，極めて小さな L0 命令キャッシュを高連想度 L1 キャッシュの上位に配置して，L1 キャッシュへの参照回数を削減し，タグ参照のための動的電力の削減を実現する単純な手法が考えられる．図 1 に極めて小さいサイズ (2 または 4 ブロック) の L0 フルアソシアティブ構成での命令キャッシュの L1 への参照削減の効果を示す．

図 1 で示される結果の計測環境は 3.1 節で示すシミュレーション環境により観測した．L0 命令キャッシュのブロックサイズは 32 バイトである．横軸は MiBench[5] のアプリケーションを示し，縦軸は MiBench のそれぞれのアプリケーションを実行した場合の L1 命令キャッシュへの参照削減率 (= L0 命令キャッシュのヒット率) を示す．図の黒い棒は L0 命令キャッシュが 2Way のフルアソシアティブ (2Way-Full) の場合，灰色の棒は L0 命令キャッシュが 4Way のフルアソシアティブ (4Way-Full) の場合の削減率である．図から，2 エントリのみを用意した 2Way-Full 場合でも，ほとんどのアプリケーションで 80%以上の L1 命令キャッシュへの参照を削減 (フィルタ) していることがわかる．2Way-Full では，全てのアプリケーションの平均で約 84%の L1 命令キャッシュ参照を削減していた．4Way-Full では，全てのアプリケーションの平均で 88%の L1 命令キャッシュへの参照を削減している．例えば，L1 キャッシュが 32Way で構成されている場合，タグ比較は 32 のキャッシュブロックに対して行わなければならないが，2Way-Full で L0 命令キャッシュを構成しヒットした場合，タグ比較対象が 1/16(6.25%) に減少し，タグ比較のための動的電力が減少する．仮に 84%の参照が L0 でヒットし，比較対象キャッシュブロック数に比例して電力が削減されると仮定すれば，タグ参照に要する動的電力は L0 導入前と比べ，この非常にシンプルな方法で 21.25%にまで抑えられる見積もりを立てることができる．

1.3. 本研究の目的と提案手法

図 1 の結果でたった数 Way のフルアソシアティブキャッシュが 80%以上のヒット率を生み出した理由は，命令参照の局所性にある．命令キャッシュにロードされたブロックはプログラムカウンタの値 (命令参照アドレス) の変動特性から，分岐命令を Taken 実行しない限り，しばらく参照され続ける特性 (空間的局所性) をもつ．また，命令参照はループ回数の多いのループの内側の命令 (ホットトレース) を支配的に参照し続ける傾向にあり，キャッシュはループ実行中にそのホットトレースをキャッシュに格納できれば，キャッシュミスは，ほぼ発生しない．極めて小容量のキャッシュで 80%以上もの参照をカバーできたのは，MiBench のような組込み向けのプログラムのホットトレースの大半が 2 ブロックに収まるほど小さいものであることに起因している．

しかしながら，残りの 20%のミス率を埋めるために，L0 命令キャッシュのサイズを増大することは消費電力や面積対性能の効率の問題から，本末転倒となる．本研究はこの極めて小さいサイズの L0 命令キャッシュをそのまま活かして，L0 命令キャッシュのヒット率を向上させることを目的とする．

2. サクセッサを用いた L0 命令キャッシュのヒット率向上

2.1.L0-L1 命令キャッシュ間のプリフェッチ

本研究は 1.2 節で示した極めて小さいサイズの L0 命令キャッシュのヒット率を向上させるために，L1 命令キャッシュから L0 命令キャッシュへのプリフェッチを提案する．一般的には，プリフェッチは主記憶 (DRAM) からキャッシュメモリにデータを読み出す場合に，メモリアクセスレイテンシを隠蔽する用途で用いられる手法であるが，本研究はこれをキャッシュメモリ間で行うことにより，高連想度のキャッシュタグ比較の動的電力削減に用いる．

図 2 に本研究が提案する L0-L1 命令キャッシュ間プリフェッチの概要を示す．L0 命令キャッシュはヒット時に

†北陸先端科学技術大学院大学

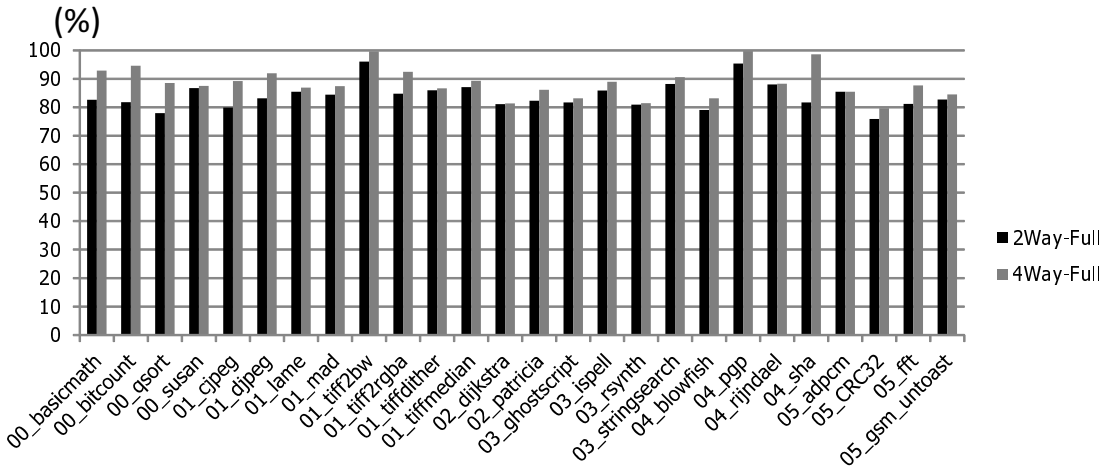

図 1: 極めて小さいサイズの L0 フルアソシアティブ命令キャッシュの L1 参照数の削減効果.

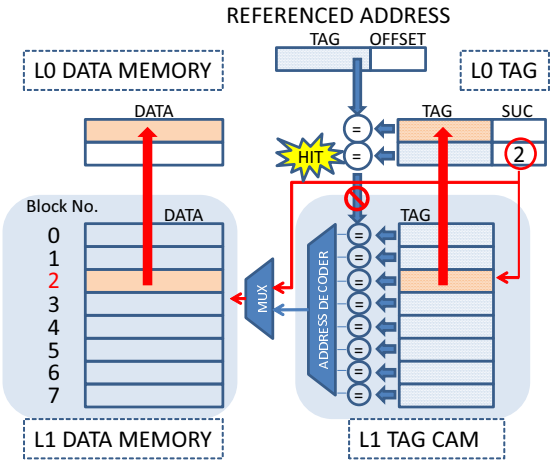

図 2: L0-L1 命令キャッシュ間プリフェッチの概要.

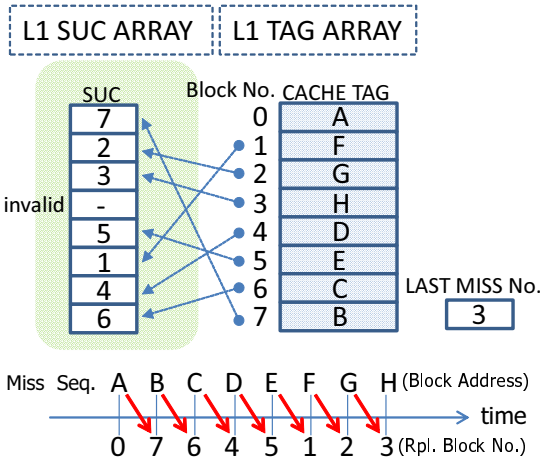

図 3: L1 におけるサクセッサ情報の収集.

L1 キャッシュからプリフェッチを行う．L0 TAG の SUC フィールド (サクセッサフィールド) にプリフェッチのための情報が格納されており，この情報は L1 TAG と L1DATA のインデックスを示している．これを用いることで，L1 タグ参照を必要としない L0-L1 命令キャッシュ間プリフェッチを実現する．L1 DATA MEMORY の読み出しアドレス入力には，通常，L1 TAG CAM から出力されるアドレスが入力されるが，これにマルチプレクサを介在させ，他方の候補に SUC フィールドを入力し，選択式とする．また，通常のプログラム実行では L1 TAG CAM へのアドレス入力による TAG の読み出しラインは使用しないが，MIPS の CACHE 命令 [7] のように特権命令がキャッシュのブロック番号を指定してキャッシュタグを読み書することができることから，L0-L1 命令キャッシュ間プリフェッチのための設計にこの読み出しラインを流用することは比較的容易である．

2.2.L1 命令キャッシュ内でのサクセッサ情報の収集

本研究におけるサクセッサ情報は L1 におけるミスの履歴である．サクセッサ情報 (SUC フィールド) は L0 と L1 の両方のキャッシュブロック毎に存在する．L0 キャッシュにおけるサクセッサ情報の利用方法は図 2 の L0 プリフェッチの概要で既に示した．一方，L1 キャッシュにおける SUC フィールドはサクセッサ情報の収集のために利用される．図 3 にサクセッサ情報の収集の例を示す．

図 3 の上部の図は L1 TAG ARRAY に対応するサクセッサ情報を，下部は上部のサクセッサ情報を収集した場合の L1 キャッシュミスのシークエンスを示す．メモリブロックを識別するブロックアドレスは A，B，C と表現される．上部の図では CACHE TAG の左側の SUC フィールドが対になるサクセッサ情報を格納している．例えば，A のタグの横にある値”7” の SUC は A が参照された後には，7 番目の L1 キャッシュブロック（すなわちブロックアドレス B のキャッシュブロック）

が参照されるという予測を示している．

この SUC を格納するために，図中で LAST MISS No. と名付けられた特殊なレジスタを用いる．LAST MISS No. は最後にミスをした時に置換対象となったキャッシュブロックの番号を常に記憶している．例えば，下部の図で A のブロックアドレスでキャッシュミスが発生し，0 番目のキャッシュブロックが置換対象になった場合，LAST MISS No. には 0 が入る．次に，B のブロックアドレスでキャッシュミスが発生し，置換対象が 7 番目のキャッシュブロックになった場合，置換対象の決定で算出した”7” の値を LAST MISS No. に格納された 0 というキャッシュブロック番号に従い，0 番目の L1 SUC ARRAY に格納する．この仕組みを導入することで，L1 キャッシュはミスでロードしたキャッシュブロックの順序を記憶しておくことができる．

SUC フィールドのビット幅は L1 命令キャッシュがもつブロック数によって決定される．(例：32 ブロックなら 5 ビット，64 ブロックなら 6 ビット) これに加え，無効 (Invalid) を示す 1 ビットが必要となる．L1 命令キャッシュにおいては最後にロード されたブロックはサクセッサ情報を持たないので，Invalid ビットが 1 と示される．一方，L0 命令キャッシュにおいては，Invalid ビットが 0 の場合はプリフェッチを発行し，Invalid ビットが 1 の場合はプリフェッチを発行しない制御に用いる．

2.3. サクセッサ情報の L0 命令キャッシュへのロード

L1 キャッシュ内で生成された SUC は L0 命令キャッシュがミスしたタイミングで L0 命令キャッシュタグとセットで SUC フィールドにロード される．(例えば，図 3 におけるタグ:A と SUC:7 がセット，タグ:F と SUC:2 がセットなど) これは通常のキャッシュミスハンド リングとほぼ同様である．サクセッサは L0 キャッシュでヒットした時にのみ，プリフェッチのための情報として使用される．

3.CPU シミュレーションによる評価

3.1. 評価環境

本研究は L0 命令キャッシュのヒット率を向上させることで，タグ比較のための電力を削減することを目的とする．本稿では，提案手法の電力評価の前に，提案手法でヒット率向上が達成できるかを検討する．

L0 命令キャッシュのヒット率向上を計測するために，ARM 命令セットアーキテクチャのプロセッサシミュレーションを SimpleSclar/ARM[8] を用いて行った．計測対象となるベンチマークは組込み向けのベンチマークである MiBench[5] を用いた．実行バイナリは MiBench[5] の HP[6] より，MiBench Version 1.0 の ARM 用プリコンパイルバイナリを取得し使用した．

SimpleSclar/ARM のキャッシュシミュレーション部には 2 節で示した L0-L1 命令キャッシュ間プリフェッチハードウェアをシミュレーションモデルに加えた．L0 命令キャッシュはインクルージョンプロパティで L1 命令キャッシュの一部のデータをコピーする．命令キャッシュシミュレーションによる計測は 2 つのキャッシュ構成で行った．一つは，L0-L1 命令キャッシュ間プリフェッチを行わない場合の L0 命令キャッシュのミス率の計測である．これは，提案手法の性能比較の対象となる．もう一つは，比較対象と同一の L0 キャッシュ構成で，提案手法である L0-L1 命令キャッシュ間プリフェッチを行った場合の L0 命令キャッシュのミス率の計測である．プロセッサシミュレーション実行方式は sim-outorder を使用した．表 1 に SimpleSclar/ARM の命令キャッシュのシミュレーションパラメータを示す．

表 1: SimpleSclar/ARM の命令キャッシュのシミュレーションパラメータ.

ifqsize/decode/commit width	4
il0 size	128B
il0 block size	32B
il0 way	4 (Full-Assoc.)
il0 Replacement	LRU
il1 block size	32B
il1 size	4KB
il1 way	128 (Full-Assoc.)
il1 Replacement	LRU

表 2: アプリケーション名の接頭語によるベンチマークの分類分け.

分類	接頭語
Automotive/Industrial	00_
Consumer	01_
Office	02_
Network	03_
Security	04_
Telecomm	05_

3.2. 評価

図 4 に提案手法による L0 命令キャッシュミス率の変化を示す．横軸は MiBench のアプリケーションを示している．各アプリケーション名の接頭語である数字は表 2 で示す分類によって与えられる．縦軸は MiBench のそれぞれのアプリケーションを実行した場合の L0 命令キャッシュミス率を示す．図の黒い棒は L0 命令キャッシュが提案手法と同容量の場合で L0-L1 命令キャッシュ間のプリフェッチ制御を行わない場合の L0 命令キャッシュミス率 (Base)，灰色の棒は L0 命令キャッシュに提案手法である L0-L1 命令キャッシュ間のプリフェッチ制御を組み込んだ場合の L0 命令キャッシュミス率 (Proposed) をそれぞれ示している．

00_basicmath と 01_tiff2bw を除いたアプリケーションで，提案手法は L0 命令キャッシュのミス率の削減に有効であるとことが観測できた．最大の性能向上は 05_adpcm であり，約 50%の L0 命令キャッシュミスを削減した．アプリケーション全体の平均では，約 15%の L0 命令キャッシュミスを削減した．

00_basicmath と 01_tiff2bw は性能低下が確認できる．これは，L0-L1 命令キャッシュ間プリフェッチの影響で

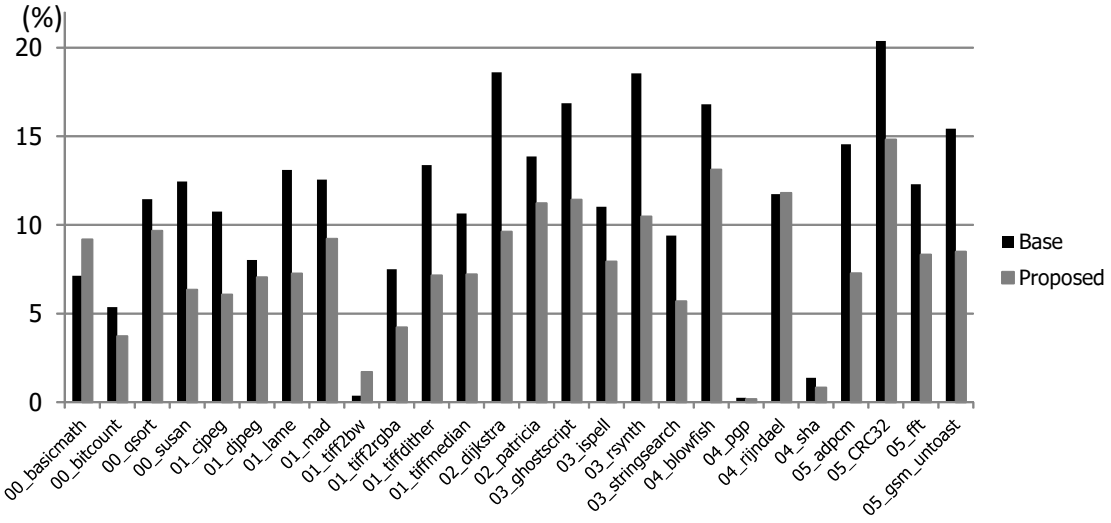

図 4: 提案手法による L0 命令キャッシュミス率の変化.

live time[9] 中のメモリブロックをプリフェッチしたメモリブロックが置換し，追い出していることが原因であると考えられる．

4. おわりに

本稿は高連想度のキャッシュメモリを構成した場合における，タグ参照に必要な電力を削減するために，極めて小さな L0 キャッシュメモリと高連想度の L1 キャッシュメモリ間で，L1 キャッシュ内のサクセッサ情報を用いてプリフェッチを行い，L0 キャッシュのヒット率を向上させる手法を提案した．

本稿の評価は提案手法は L0 命令キャッシュミスを最大で約 50%，平均で約 15% 削減することが可能であることを示した．この結果は提案手法が有効に機能していることを示しており，これによる電力削減の可能性があることを示している．

今後の課題として，電力見積もりが上げられる．本研究は L0 命令キャッシュと L1 命令キャッシュの両方に SUC フィールドが必要となることから，このメモリ部分の静的消費電力が存在するため，ヒット率向上による電力削減効果を減らすことが予想できる．これらの影響を考察するために，今後は CACTI[10] による消費電力評価を行うと共に，1.1 節で示した 2 つの先行研究との比較評価を行う．

参考文献

[1] J. Montanaro, et al., ”A 160-MHz, 32-b, 0 5-W, CMOS RISC Microprocessor”, IEEE Journal of Solid-State Circuits, Vol. 31, No. 11, Nov. 1996, pp.1703–1714, 1996.

[2] Intel Corporation, “Intel XScale Microarchitecture Technical Summary”, in http://int.xscale-freak.com/XSDoc/PXA27X/XScaleDatasheet4.pdf, 2000.

[3] A. Ma, M. Zhang and L. Asanovic, “Way memoization to reduce fetch energy in instruction caches”, ISCA Workshop on Complexity Effective Design, July, 2001.

[4] A. Veidenbaum, and D. Nicolaescu “Low Energy, Highly- Associative Cache Design for Embedded Processors” Proc. of IEEE ICCD, pp. 332–335, 2004.

[5] M.Guthaus, J.Ringenberg, D.Ernst, T.Austin, T.Mudge, R.Brown, “MiBench: A free, commercially representative embedded benchmark suite”, Proc. of IEEE Intl. Workshop on Workload Characterization, 2001 (WWC-4).

[6] http://www.eecs.umich.edu/mibench/

[7] “MIPS32^{TM} Architecture For Programmers Volume II: The MIPS32^{TM} Instruction Set”, MIPS Technologies Inc., in http://www.cs.tau.ac.il/ afek/ MipsInstructionSetReference.pdf, 2003.

[8] http://www.simplescalar.com/v4test.html

[9] S. Kaxiras , Z. Hu and M. Martonosi, “Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power”, Proc. of the 28th Ann. Int ’l Symp. Computer Architecture(ISCA 2001), 2001.

[10] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, “Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0,” Proc. of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, pp.3–14, 2007.