C-006

資源利用率に基づく連想度可変キャッシュメモリ

Associativity-Variable Cache Memory Based on Resource Usage Rate

渡邊 恭成† 中野 秀洋‡ 宮内 新‡
Yasumasa Watanabe Hidehiro Nakano Arate Miyauchi

1．はじめに

半導体のプロセス技術の進歩によりマイクロプロセッサの集積度が増加し，性能が飛躍的に向上した．しかし，これによって単位時間当たりの処理量が増え，スイッチング損失による消費電力の増加が問題となってきている．特に，キャッシュメモリはマイクロプロセッサの消費電力の大部分を占めているため，近年ではキャッシュメモリの低消費電力化に対する関心が高まってきている．

キャッシュメモリの多くは N ウェイ・セットアソシアティブで構成され，連想度を増加させることによって競合ミスの削減を図っている．しかし，複数の記憶配列へ並列にアクセスを行う N ウェイ・セットアソシアティブでは，消費電力を単独の論理回路レベルで妥当に制御することができないため，不要な電力がアクセスごとに消費される．消費電力とミス率は一般的にトレードオフの関係にあり，連想度を最適な状態に設定することは困難である．そのため，多くの研究者によって連想度を可変化する研究が行われてきた[1]-[4]．このうち，文献[1]では最大でスイッチング損失を 64%削減できることが述べられている．しかし，[1]-[4]のいずれの手法も連想度の変化のパターンがある程度固定的であり，様々なアプリケーションに対して柔軟に対応することはできない．

本稿ではキャッシュブロックの利用率に基づいて，動的かつ柔軟にキャッシュメモリの連想度を変化させるアルゴリズムを提案する．キャッシュメモリのアクセスには局所性があり，最近利用されていないブロックは今後も利用される可能性が低い．この原理に基づきそのブロックへのアクセスを禁止することで消費電力の削減を図る．しかし，単に利用率に基づいてアクセスを制限するだけでは，ミスの増加によって性能を低下させる可能性がある．このため Victim Cache を用いてミス数の削減も図る．評価実験を行い提案手法の有効性を示す．

2．資源利用率に基づく連想度可変キャッシュメモリ

2.1 フラグ参照型 LRU

N ウェイ・セットアソシアティブキャッシュでは下位の階層のメモリとデータの置換を行う際に，LRU アルゴリズムで置換するキャッシュブロックを決定することが一般的である．LRU とは最近利用されていないブロックを決定するアルゴリズムである．LRU によって対象となったブロックは局所性により今後利用される確率が低い．このため，LRU は多くの場合で高い効果を発揮する．しかし，LRU を実装するためには各ウェイのアクセス履歴を記録する必要があり，通常はコスト的に困難である．そのため多くの場合は擬似的な LRU が用いられる．その一つがフラグ参照型 LRU である．この方法では，アクセスされたブロックのフラグを立て，定期的にキャッシュ内のすべてのフラグをリセットすることを繰り返す．すなわち，フラグによりアクセス履歴の「新古」のみを判別する．フラグリセットのタイミングを適切に設定することで高い効果を発揮することができる．本稿ではフラグ参照型 LRU を用いることを前提としてアルゴリズムを提案する

† 東京都市大学　大学院　工学研究科
‡ 東京都市大学　知識工学部　情報科学科

2.2 Victim Cache

Victim Cache とは数エントリから成るフルアソシアティブキャッシュであり，最上位のキャッシュから置換対象となったデータが一時的に保存される[5]．Victim Cache を用いた場合，仮想的に連想度を増加させることができるため競合ミスを削減することができる．しかし，キャッシュにアクセスされるたびに Victim Cache にもアクセスが行われるため，消費電力が増加する．

2.3 提案アルゴリズム

2.1 節で述べたフラグ参照型 LRU はデータのアクセス履歴の新古を判別することができる．フラグの立っているブロックは今後利用される確率が高く，フラグの立っていないブロックは今後の利用率が低いことを意味する．そこで，フラグをリセットする際にブロックの状態をチェックし，フラグの立っているブロックのみにアクセスを制限する．提案手法の単純化した回路図を図 1 に示す．Flag 0-3 は LRU のフラグをリセットする直前に記憶した信号であり，次のリセットもしくはミスが発生するまでこの信号は保持される．この手法は記憶した信号をそのまま制御信号として扱うことができるため，ハードウェアコストは最小限である．一方，アクセスが制限された状態でミスが起きた場合，その他のブロックでヒットする可能性があるため，制限を解除し再度アクセスを行う．

あるセットのブロックのフラグがすべて立っていた場合，そのセットの利用率は特に高いことを意味する．そこで，すべてのブロックのフラグが立っていた場合は Victim Cache へのアクセスを許可し，競合ミスの削減を図る．Victim Cache へのアクセス信号はセット内のすべての

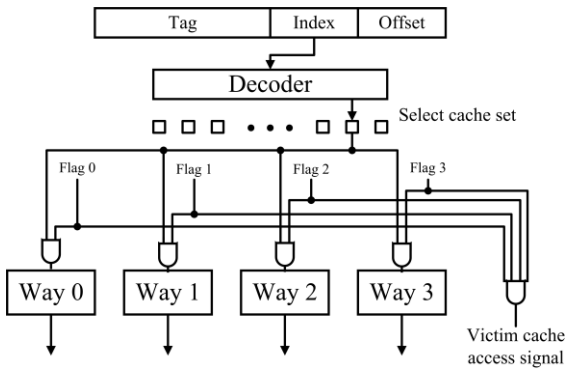

図 1：提案手法の回路図

フラグに対して AND をとるだけで良い．従来の手法では，Victim Cache は毎回アクセスが行われるため消費電力が問題となっていたが，本手法では利用率の高いセットにアクセスする場合のみに利用を制限する．そのため，消費電力の増加は最小限に抑えることができる．ただし，Victim Cache の利用を制限することで下位のメモリとデータの整合性が失われる可能性があるため，本手法は書き込みが行われない命令キャッシュに適用することを前提とする．

3. 実験

プロセッサシミュレーションツールである SimpleScalar に提案手法を導入し評価実験を行った．実験環境を表 1 に示す．なお，表中の Flag Reset Interval は事前実験により最も良い値を使用した．また，Re-access Latency は，ミスによってアクセス制限が解除されたときのアクセスレイテンシを表す．下位のキャッシュへアクセスせずに同じセットに再度アクセスを行うため，このレイテンシは 1 Cycle とした．また Victim Cache Latency に関しても，極めて小規模な構成であるため 1 Cycle とした．オーバーヘッドに関してはフラグ参照型 LRU が他の LRU と比べて非常に単純であることと，提案手法に対するハードウェアコストがわずかであることから，本実験では考慮しないものとした．提案手法の比較対象として，連想度可変や Victim Cache の機能を持たない一般的なキャッシュを用いた．ただし，このキャッシュはフラグ参照型 LRU ではなく，最も古いブロックを置換する最適な LRU の機能を持つものとした．使用するアプリケーションは *Dhrystone*, *Fast Fourier Transform (FFT)*, *Rijndael*, *Secure Hash Algorithm (SHA)*, *Camellia*, *Dijkstra*, *String search* とした．

図 2 に一般的なキャッシュに対する提案手法のミス比を示す．提案手法は多くのアプリケーションにおいてミス数を削減することができた．*SHA* ではミス数が増加しているが，これは *SHA* が非常に単純なプログラムであるためミスの絶対数が少なく，LRU アルゴリズムの違いによって現れた差であると考えられる．図 3 にブロックの平均稼働率を示す．なお，一般的なキャッシュではセット内の全てのブロックにアクセスするため稼働率は 100%となる．この結果より，提案手法では稼働率を大きく削減できたことがわかる．ミス数が増加していた *SHA* においては稼働率を 30%以下まで削減できている．図 2，3 の結果から，単純なプログラムではミス数を大きく削減することはできないが，稼働率を大きく削減することが可能であるということがわかる．

提案手法は同セットへの再アクセスや Victim cache へのアクセスがあるため，本来は不必要なサイクルが発生する．しかし，資源利用率に基づいて連想度を最適な状態へと変更することができるため，再アクセスと Victim Cache へのアクセスが最小限となり，性能の劣化を抑制することができる．そして *Rijndael* のようなミス率の高いアプリケーションでは，メインメモリのアクセスレイテンシが支配的であるため，ミス数を削減することによりサイクル数を大幅に削減することができる．

表 1：実験環境

Cache Size	16KB
Associative	4
Block Size	32B
Flag Reset Interval	16384 Accesses
Victim Cache Entry	4
Re-access Latency	1 Cycle
Victim Cache Latency	1 Cycle
L2 Cache Latency	6 Cycles
Main Memory Latency	20 Cycles

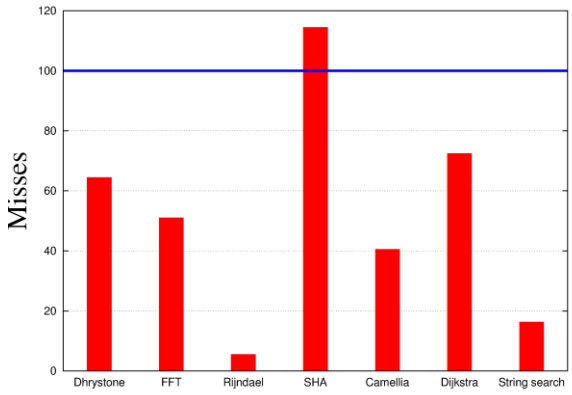

図 2：提案手法のミス数

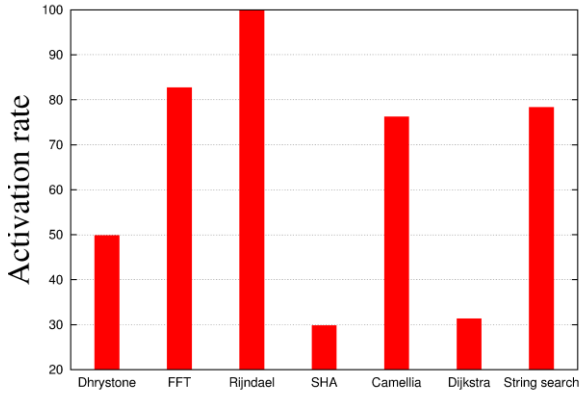

図 3：提案手法におけるキャッシュブロックの平均稼働率

4. むすび

本稿では資源の利用率に基づいて連想度を可変化するキャッシュメモリを提案した．提案手法は多くのアプリケーションでミス数とキャッシュブロックの稼働率を大きく削減することができた．今後の課題としてオーバーヘッドの考慮，消費電力の評価，様々なアプリケーションでの実験などが挙げられる．

参考文献

[1]C. Zhang, F. Vahid and W. Najjar, “A Highly Configurable Cache for Low Energy Embedded Systems”, *ACM Transaction on Embedded Computing Systems*, Vol. 4, No. 2, pp. 363-387, (2005).

[2]K. T. Sundarajan, T. M. Jones and N. Topham, “Smart Cache: A Self Adaptive Cache Architecture for Energy Efficiency”, *International Conference on Embedded Computer Systems*, pp. 41-50, (2011).

[3]K. Inoue, T. Ishihara and K. Marukami, “Way-Predicting Set Associative Cache for High Performance and Low Energy Consumption”, *Proc. International Symposium on Low Power Electronics and Design*, pp.273-275, (1999)

[4]C. J. Janraj, T. V. Kalyan, T. Warrier and M. Mutyam, “Way Sharing Set Associative Cache Architecture”, *International Conference on VLSI Design*, pp. 251-256, (2012)

[5]N. P. Jouppi, “Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers”, *Annual International Conference on Computer Architecture*, pp. 364-373