Vol.2014-ARC-208 No.9
2014/1/23

アクセス制限を有するビクティムキャッシュを利用した連想度可変キャッシュメモリ

渡邊 恭成[1,a)]　中野 秀洋[1,b)]　宮内 新[1,c)]

概要：本稿では, 各キャッシュエントリに対して, それぞれの使用頻度に基づくアクセス制限のビットを設けた, 連想度可変キャッシュメモリ機構を提案する. 本機構は, 各 L1 キャッシュエントリとビクティムキャッシュエントリへのアクセスを最小限とすることによって, 十分な性能を維持しつつ消費電力の削減を図ることができる. シミュレーションを行い, 提案機構の有効性を確認する.

キーワード：N ウェイセットアソシアティブ，連想度可変化，ビクティムキャッシュ，低消費電力

Associativity-Variable Cache Using Victim Cache with Access Limitations

Abstract: This paper proposes an associativity-variable cache memory scheme introducing a bit for access limitations in each cache entry, based on its frequency of use. This scheme can reduce the energy consumption by restricting access for each L1 cache entry and each victim cache entry, keeping enough performances. By performing experiments, the effectiveness of the proposed scheme is confirmed.

Keywords: N way set-associative, Associativity-variable, Victim cache, Low energy consumption

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, トランジスタ回路の大規模集積化が実現されている. これに伴い, 新たなハードウェアサポートの追加や, 高いクロックレートの実現によってプロセッサの性能は飛躍的に向上している. しかし, より多くの機能が追加されることにより, クロック周波数の向上も関係して回路におけるスイッチングが増大し, それに伴う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. このため, 多くの文献で消費電力を削減する手法が提案されてきた. そして, それらの多くは N ウェイセットアソシアティブキャッシュを前提としている.

フィルタキャッシュと呼ばれる小容量の 0 次キャッシュを用いる手法 [1] は, 1 次キャッシュへのアクセス数を削減することで消費電力を削減している. V ウェイキャッシュ [2] はタグエントリを追加し, 各タグエントリに対応付けられたポインタによってアクセス箇所を一意に決定する. そのためデータ配列へ並列アクセスを行う必要性がなく消費電力を削減することができる. Gated-V_{dd}[3] は未使用のキャッシュブロックを電圧源から遮断することでリーク電力を削減することに焦点を当てており, 多くの手法へと応用されている. その他にも, 最も利用履歴の古いブロックのみを検索するブロック置換方式 [4] や, アプリケーションの動作に基づいてラインサイズを調整する手法 [5] は, ハードウェアコストと下位階層のメモリのトラフィックを削減することができる.

N ウェイセットアソシアティブキャッシュは, 連想度の増加によって競合ミスを抑えることができる. しかし, N ウェイセットアソシアティブの性能を最大限に発揮するには, 複数の配列へと並列にアクセスを行わなければならず,

[1] 東京都市大学
Tokyo City University
[a)] yasumasa@ic.cs.tcu.ac.jp
[b)] nakano@ic.cs.tcu.ac.jp
[c)] miyauchi@ic.cs.tcu.ac.jp

1 回のアクセスに対して大きな消費電力を必要とする.

上記 [1]-[5] の手法はいずれも消費電力を削減することができるが, これらはキャッシュの構成が固定的であるためアプリケーションによっては連想度が不適切となり, 不要なアクセスが頻繁に発生する場合がある. アプリケーションごとに要求される連想度は異なるため, キャッシュの連想度は動的な再構成が行われるべきである. 以降では連想度の再構成に関する研究を紹介し, その有効性について述べる.

ミス数とアクセスの消費電力はトレードオフの関係にあり, 連想度を大きく設定することはキャッシュミスの削減を可能とするが, 同時に消費電力の増加を招く. しかし, キャッシュメモリではミスの発生が致命的であるため, 連想度はある程度大きく設定されることが多い. 連想度を大きく設定した場合, 単純なアプリケーションの実行時に多くのウェイは利用されず, 記憶領域を浪費することとなる. そのため, 浪費されるブロックへのアクセスを削減することができる連想度可変化の技術は, 消費電力の削減に非常に有効な手段であると言える.

キャッシュメモリの連想度可変化は決して新しい技術ではなく, 多くの研究者によって様々な手法が提案されている. Chuanjun らによって提案された再構成キャッシュ [6] は, アプリケーションごとに制御信号によって連想度を可変化させる. この機構は利用するアプリケーションのプロファイリングが必要であるため, ヒューリスティックな手法を用いて構成を変更する機構 [7][8] も提案されている. これらの機構は多くのアプリケーションに対して有効性が確認されているが, 連想度とともにアクセスできる容量が変更されてしまう. Smart cache[9] は, 連想度だけでなくセット数を再構成することを可能としたため, より多くのアプリケーションへと対応することができる. また, この機構での連想度の変更は並列アクセス数を変更するだけであり, すべてのウェイにデータを格納することができる. そして, 文献 [6]-[8] の機構と比べて有効であることが述べられている. 予測機構に基づいて一つのウェイにアクセスする手法 [10]-[12] は, アクセス時に擬似的にダイレクトマップ方式として扱うため, 予測が成功した際に消費電力を大幅に削減することができる. 非対称キャッシュ [11] は予測機能だけでなく, それぞれのウェイで異なるブロックサイズを持つという特徴があり, 小容量のブロックが多く利用されるようにブロックの置換を行うことでさらに消費電力を削減する. ウェイ共有キャッシュ [13] は, 定められたセット間でウェイを共有することで連想度を変更することができ, 利用率の低いセットは連想度が小さく設定される. この手法は一部のウェイを共有することによりキャッシュメモリの総容量を削減することができ, ハードウェアコストとアクセスによる消費電力を削減することができる. しかし, 総容量を削減することにより, わずかではあるが性能の低下を招く. 文献 [14] は連想度を従来の半分, セット数を倍とし, セットの半分を共有用とすることで連想度の可変化を図っている. そしてウェイのポインタにより連想度を変更する. その他の手法と比べこの手法は変更の自由度は高いが, 連想度の増加によるリダイレクトでレイテンシが増加する可能性がある.

連想度の可変化は消費電力の削減に有効であることは上記で述べたが, 従来の多くの手法は可変のパターンが少ないこと, アプリケーションに対して静的な変更であること, 性能を犠牲にすることを前提としていることが問題である. 要求される連想度はアプリケーションごとに異なるが, さらに追及するとキャッシュのセット, アプリケーションのスレッドごとに異なる. 可変のパターンが少ない, もしくは静的な変更ではスレッドレベルで連想度を最適化することはできないため, 消費電力削減の余地を残すこととなる. また性能の低下を前提とした場合は, キャッシュを除いた他の回路の使用頻度が増すため, 理想的な消費電力の削減とは言い難い. そこで本稿では, より柔軟で多くのアプリケーションに対応することのできる動的連想度可変キャッシュを提案する.

2. 提案機構

N ウェイセットアソシアティブキャッシュ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 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すべてのブロック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 であり, この方法は各ブロックに新たにフラグを設ける. そして, 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. そのためコストが少なく, また, 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. 本稿ではこのフラグ参照型 LRU を用いることを前提とし, 新たな連想度可変キャッシュメモリを提案する. また, 置換ブロックの選択対象が複数ある場合, その対処法はいくつか存在するが, 本研究ではそれらのブロックからランダムに選択を行うものとする.

2.1 フラグを用いたブロックアクセス制限

フラグ参照型 LRU はデータ利用履歴の新古を判別することができる. データのアクセスには局所性があるため, フラグの立っているキャッシュ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. そこでブロック置換用のフラグとは別に, 新たにアクセス制限用のフラグを各ブロックに設け, ブロック置換用のフラグがリセットされる際にその状態をアクセス制限用のフラグとしてコピーする. そして次回以降のアクセスはアクセス制限用フラグの立っているブロックのみに行う. アクセス制限が行われた状態でキャッシュミスが発生した場合, アクセス制限されたウェイに要求されたデータが存在する可能性がある. そのため, 上記の際にはそのセットのアクセス制限を解除し, 再度通常の N ウェイアクセスを行うように戻す.

この制限方法は記録したフラグ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の増加は最小限である. さらに 2^N 通りの可変のパターンを実現できるため, 極めて柔軟性が高い. しかし, アクセス制限時にミスが生じると 2 段階のアクセスが発生し, これによるサイクル数の増加が問題となる. 単純なアプリケーションの場合はわずかなサイクル数の増加であると考えられるが, 複雑なアプリケーションでは連想度の構成が頻繁に変更される可能性があるため, サイクル数の増加量が無視できない. 性能の犠牲による消費電力の削減は望ましいものではない. そこでアクセス制限を行いつつも, 同時にサイクル数の増加を抑制できる機構を次項で述べる.

2.2 ビクティムキャッシュを利用した連想度可変キャッシュメモリ

あるセットのアクセス制限用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く, 容量が不足している可能性が高い. そこで, すべての制限用フラグが立っているセットのみビクティムキャッシュ [15] へのアクセスを許可し, 競合ミスの削減を図る.

ビクティムキャッシュは, 小容量のフルアソシアティブキャッシュであり, L1 キャッシュで置換されたデータが保存される. L1 キャッシュのアクセスと同時にビクティムキャッシュへアクセスを行うことで仮想的に連想度を増加することができるため, ビクティムキャッシュは競合ミスの削減に効果的である. 従来のビクティムキャッシュは毎回アクセスが行われるため, アクセスあたりの消費電力が増加するという欠点を持つ. しかし, 提案機構はビクティムキャッシュのアクセスを, フラグがすべて立っているセットのみに制限することで消費電力の増加を抑制し, さらに利用率の低いデータの格納を防ぐことができる. 提案機構の概念図を図 1 に示す. L_i はアクセス制限用フラグの信号であり, 置換用フラグである R_i のリセットもしくはミスが発生するまで保持される. ビクティムキャッシュへのアクセス信号はセット内のすべての制限用フラグに対して AND をとるだけで良い. そのため, ビクティムキャッシュの利用に関してもハードウェアコストの増加はわずかであると言える.

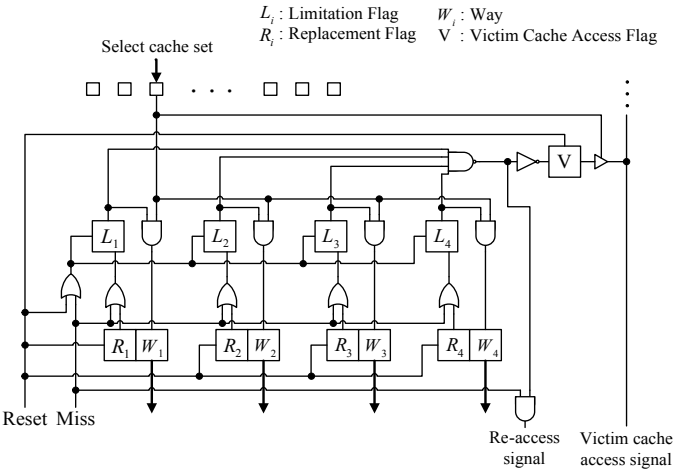

図 1 提案機構の回路図
Fig. 1 Proposed cache architecture

提案機構はビクティムキャッシュの利用を制限することで下位階層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, 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

提案機構の有効性を検証するために, simplescalar[16] に提案機構を追加したプロセッサを用いて実験を行った. 評価するキャッシュメモリは一般的な N ウェイセットアソシアティブキャッシュ (Common Cache : CC), 従来のビクティムキャッシュを搭載したキャッシュ (Conventional Victim Cache : CVC), 容量を 1 ウェイ分 (1/N) に縮小したダイレクトマップキャッシュ (Direct Mapped : DM), アクセス制限の機能のみを持ったキャッシュメモリ (Limitation Only : LO), アクセス制限とビクティムキャッシュの機能を持つ提案機構 (Limitation and Victim Cache : LVC) とした. なお, これらのキャッシュメモリの置換方式はすべてフラグ参照型 LRU とした. 表 1 にプロセッサの構成を, 表 2 に使用したベンチマークプログラムを示す. 表 1 における 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である. 下位階層のメモリへのアクセスはせず, 同じセットへの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キャッシュであるため 1 Cycle とした. よって, ビクティムキャッシュと L1 キャッシュは並列にアクセスが行われることから, サイクル数の増加は発生しない. 遅延や消費電力に関するオーバーヘッドは, 提案機構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.

3.1 ミス数

図 2 に一般的なキャッシュメモリの結果を 100 として正規化したミス数の比率を示す. DM は他のキャッシュと比

表 1　プロセッサの構成
Table 1　Processor configuration

Parameter	Cnfigurations
Type	L1 Instruction Cache
Size	16 KBytes
Associative	4
Block Size	32 Bytes
Flag Reset Interval	8192
Victim Cache Entry	16
L1 Cache Latency	1 Cycle
Re-access Latency	1 Cycle
Victim Cache Latency	1 Cycle
L2 Cache Size	256 KBytes
L2 Cache Latency	6 Cycles
Main Memory Latency	32 Cycles

表 2　シミュレーションで用いたベンチマークプログラム
Table 2　Benchmarks used in simulations

Name	Application
Dhrystone	Integer benchmark program
FFT	Fast fourier transform
Rijndael	Private key encryption algorithm
SHA	Secure hash algorithm
Camellia	Common key cryptosystem
Dijkstra	Single-source shortest path search algorithm
String Search	Boyer-Moore string search algorithm

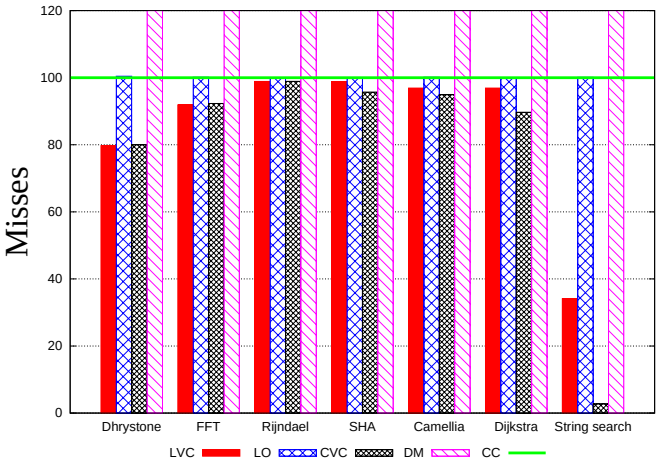

図 2　正規化された各機構のミス数
Fig. 2　Nomalized number of misses in each architecture

表 3　ビクティムキャッシュのアクセス数
Table 3　Victim cache accesses

	CVC	LVC	Savings
Dhrystone	58,481,027	1,271,088	97.8%
FFT	68,767,584	34,021,298	50.5%
Rijndael	71,549,290	71,531,447	0.0%
SHA	33,676,727	7,940	99.9%
Camellia	4,590,341	1,815,985	60.4%
Dijkstra	55,136,809	663,662	98.8%
String Search	12,167,340	4,201,693	65.5%

べ容量が小さいため, すべてのベンチマークで大幅にミス数が増加している．LO は最大容量が一般的なキャッシュメモリと同等のため, ミス数は改善されない．LVC および CVC はすべてのベンチマークにおいてミス数を削減できた．また, LVC はビクティムキャッシュの利用がある程度制限されているが, 多くのベンチマークで CVC に近い性能を示している．

表 3 に LVC と CVC のビクティムキャッシュへのアクセス数を示す．LVC のビクティムキャッシュへのアクセス数は Rijndael を除いて大幅に削減されている．このことは, 不要なビクティムキャッシュへのアクセスを行わないため, 消費電力を大幅に削減できることを意味する．また, LVC と CVC のミス数が同程度であることから, ミスは特定のセットで頻繁に発生していることがわかる．Rijndael に関しては, LVC のアクセス数はほとんど削減されていない．これは, キャッシュメモリの容量が不足しているためである．すなわち, 初期動作の除いて常にビクティムキャッシュが稼働し, LVC と CVC との差がほとんど生じなかったと考えられる．

3.2　サイクル数

図 3 に一般的なキャッシュメモリの結果を 100 として正規化したサイクル数の比率を示す．CVC が最も良い結果であり, 一般的なキャッシュメモリと比べ最大で約 3%ほどサイクル数を削減できている．LO はミス数に関しては図 2 に示したように一般的なキャッシュメモリと変わらないものの, Re-access によるレイテンシが存在するため, アプリケーションによってはサイクル数が増加する．Re-access に関しては LVC も同様であるが, ビクティムキャッシュを利用することでミス数を削減することができるため, 最大で約 2%ほどサイクル数を削減することができた．しかし, Dijkstra では一般的なキャッシュメモリと比べサイクル数が増加する結果となった．これはミス削減の絶対数が少なく, Re-access による増加がそれを上回ったためであると考えられる．

3.3　アクセスエネルギー

キャッシュアクセスによる動的消費電力を CACTI[17] を用いて評価した．表 4 に各構成の動的消費電力を示す．なお, CACTI では 3 ウェイの消費電力は求めることができないため, 他の設定によって得られた結果から近似的に求めた．また, 表 4 における Savings は一般的なキャッシュメモリからの消費電力の削減量を表している．

図 4 に L2 キャッシュへのアクセスを含めた動的消費電力を, 表 5 にアクセスあたりの平均動的消費電力を示す．多くのベンチマークで DM が最も良い結果となったが, こ

表 4　アクセスあたりの動的消費電力
Table 4　Dynamic energy consumption for each access

	4way+VC	16KB 4way	12KB 3way	8KB 2way	4KB 1way	256KB 4way (L2)
Energy [nJ]	0.0449	0.0386	0.0286	0.0185	0.0114	0.157
Savings	-16.2%	0.0%	26.0%	52.1%	70.6%	-306.0%

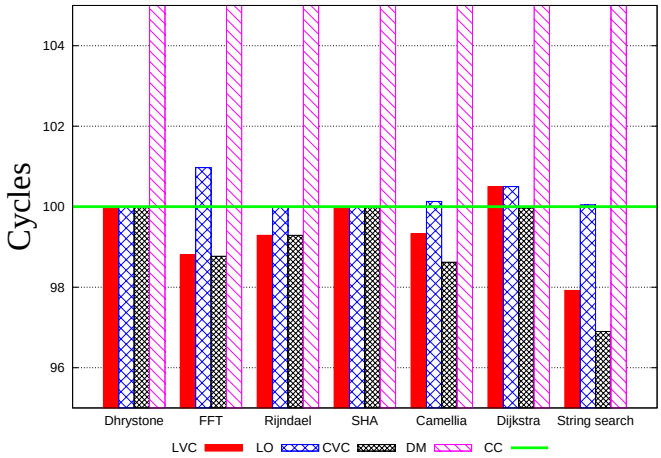

図 3　正規化された各機構のサイクル数
Fig. 3　Nomalized number of cycles in each architecture

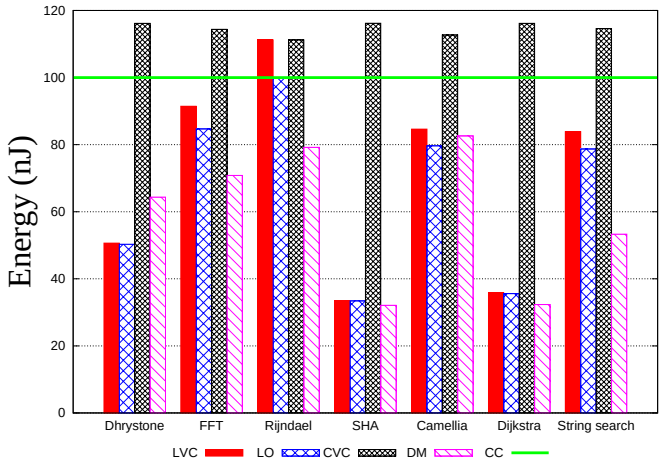

図 4　正規化した各機構の動的消費電力
Fig. 4　Nomalized dynamic energy consumption in each architecture

れは他の回路の消費電力やリーク電力を考慮していないためである．ミス数が大幅に増加する DM は，キャッシュ以外の回路を利用する頻度が他の機構に比べて高い．キャッシュ以外の回路の消費電力の合計は，一般にキャッシュの消費電力と同等もしくはそれ以上となるため，この消費電力を軽視することはできない．また，サイクル数の増加に伴ってリークによる消費電力が増加することが考えられる．上記の理由により DM は総消費電力に関して優れているとは言えない．対して CVC はミス数，サイクル数に関してはともに削減することができるが，アクセスあたりの消費電力が他の機構と比べて大きい．一方，LO と LVC の消費電力は DM に近い優れた結果であると言える，LO と LVC を比較すると，ビクティムキャッシュを持たない LO が良い結果であるが，LVC はそれに近い結果となった．また，表 4，表 5 より，LO と LVC は Rijndael を除いて 16KB 4way 構成の消費電力を下回っていることがわかる．つまり，LO と LVC は多くのベンチマークにおいて，CC でミスが発生しない場合の理想的な消費電力よりも優れた結果であると言える．Rijndael では LVC は消費電力が増加しているが，これはキャッシュ構成の容量が不足しており，常にビクティムキャッシュへのアクセスが行われているからである．そのため，さらに容量を増加した場合には他の機構と比べ良い結果を得ることができると考えられる．LVC はミス数，サイクル数の両面においても優れているため，総合的に優れていると言える．

図 5 にアクセス制限用フラグの遷移などから求めた LVC の連想度の比率を示す．この図から，一つのアプリケーションの実行中に複数の連想度が設定されていることがわかる．特に FFT や Camellia，String search ではビクティムキャッシュの利用率がそこそこ高く，容量が不足しているセットがあると予測できるが，その一方で 3 ウェイ以下の連想度が半数を占めていることがわかる．また，SHA と Dijkstra に関しては多くがダイレクトマップでアクセスされているが，2 ウェイのアクセスが 2 割弱存在するため，ダイレクトマップのキャッシュ構成では競合ミスが発生し，2 ウェイセットアソシアティブでは記憶領域の浪費が多くなることが予想される．提案機構ではセットごとに独立な連想度を設定でき，アプリケーション実行中に変更することができるため，多くのアプリケーションに対応することができる．

表 5　アクセスあたりの平均動的消費電力 (nJ)
Table 5　Average dynamic energy consumption for each access (nJ)

	CC	CVC	DM	LO	LVC
Dhrystone	0.0387	0.0449	0.0234	0.0194	0.0196
FFT	0.0416	0.0477	0.0276	0.0351	0.0380
Rijndael	0.0537	0.0598	0.0388	0.0537	0.0598
SHA	0.0387	0.0449	0.0123	0.0129	0.0129
Camellia	0.0455	0.0514	0.0347	0.0362	0.0385
Dijkstra	0.0387	0.0450	0.0124	0.0138	0.0139
String Search	0.0391	0.0449	0.0202	0.0307	0.0328
All benchmarks	0.0423	0.0484	0.0242	0.0288	0.0308

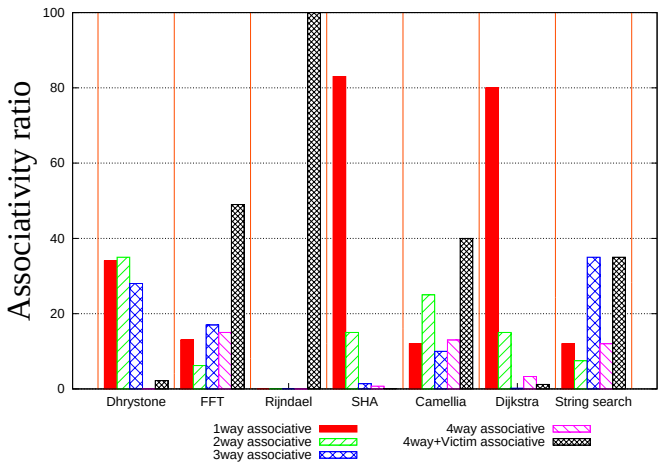

図 5 LVC における連想度比率
Fig. 5 Associativity ratio in LVC

4. おわりに

本稿では, フラグ参照型 LRU を用いた新たな連想度可変キャッシュメモリを提案した. シミュレーションの結果, ビクティムキャッシュを用いた機構ではアクセスの消費電力を削減しつつ, 性能を向上させることができた. 今後の課題としては, オーバーヘッドの考慮やより多くのアプリケーションでの実験, 様々なキャッシュ構成での実験, 他の従来の手法との比較などが挙げられる.

参考文献

[1] Y. J. Park, H. J. Choi, C. H. Kim, and J. M. Kim, "Energy-aware Filter Cache Architecture for Multicore Processors," *IEEE International Symposium on Electronic Design, Test & Applications*, pp. 58–62 (2010).
[2] M. K. Qureshi, D. Thompson, and Y. N. Patt, "The V-Way Cache : Demand-Based Associativity via Global Replacement," *Proceedings of the 32nd International Symposium on Computer Architecture*, pp. 544–555 (2005).
[3] M. Powell, S. H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar "Gated-Vdd : A Circuit Technique to Reduce Leakage in Deep-Submicro Cache Memories," *ACM/IEEE International Symposium on Low Power Electronics and Design*. pp. 90–95 (2000)
[4] H. Ghasemzadeh, S. Mazrouee, and M. R. Kakoee, "Modified Pseudo LRU Replacement Algorithm," *Proceedings of the 13th IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, p. 6 pp. –376 (2006).
[5] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji, "Adapting Cache Line Size to application Behavior," *Proceedings of the 13th International Conference on Supercomputiong*, pp. 145–154 (1999).
[6] C. Zhang, F. Vahid, and W. Najjar, "A Highly Configurable Cache Architecture for Embedded Systems," *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pp. 136–146 (2003).
[7] C. Zhang, F. Vahid, and R. Lysecky, "A Self-Tuning Cache Architecture for Embedded Systems," *ACM Transactions on Embedded Computing Systems*, Vol. 3 No. 2 pp. 407–425 (2004).
[8] C. Zhang, F. Vahid, and W. Najjar, "A Highly Configurable Cache Architecture for Low Energy Embedded Systems," *ACM Transactions on Embedded Computing Systems*, Vol. 4 No. 2 pp. 363–387 (2005).
[9] K. T. Sundararajan, T. M. Jones, and N. Topham, "Smart Cache: A Self Adaptive Cache Architecture for Energy Efficiency," *IEEE International Conference on Embedded Computer Systems (SAMOS)*, pp. 41–50 (2011).
[10] K. Inonue, T. Ishihara, and K. Murakami, "Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 273–275 (1999).
[11] Z. Hu, S. Kaxiras, and M. Martonosi, "Improving Power Efficiency with an Asymmetric Set-Associative Cache," *High Performance Memory Systems*, pp. 83–100 (2003).
[12] C. H. Kim, S. W. Chung, and C. S. Jhon, "PP-cache: A partitioned power-aware instruction cache architecture," *Microprocessors and Microsystems*, Vol. 30 No. 5 pp. 268–279 (2006).
[13] 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).
[14] M. Tang, and X. Lin, "A Novel Scheme to Balance the Cache Sharing in High Performance Computing System," *The 30th IEEE International Conference on High Performance Computing and Communications*, pp. 695-701, (2008).
[15] N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *Proceedings. 17th Annual International Symposium on Computer Architecture*, pp. 364-373, (1990).
[16] "Simplescalar," *http://www.simplescalar.com.*
[17] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.1. Technical Report HPL-2008-20," *HP Laboratories Palo Alto*, (2008).