Vol.2011-ARC-194 No.6
2011/3/11

データ符号化によるラスト・レベル・キャッシュの回路面積削減

横山弘基[†1]　堀部悠平[†1]
三輪忍[†1]　中條拓伯[†1]

近年，ラスト・レベル・キャッシュは大容量化している．ラスト・レベル・キャッシュが大容量化することにより回路面積が巨大化し，消費電力の増大・レイテンシの増加を招いている．そこで我々は，ラスト・レベル・キャッシュの回路面積を削減することを目的として研究を行う．我々はデータの冗長性に着目し，上位ビットが 0 で連続しているデータを圧縮する手法を提案する．

Data Compression on Last Level Cache for Reducing Hardware Amount

Hiroki Yokoyama,[†1] Yuhei Horibe,[†1] Shinobu Miwa[†1]
and Hironori Nakajo[†1]

The capacity of last-level cache has grown recently. The last-level cache on processor chips continues to occupy increasing amounts of chip area. It causes that high power consumption and long latency. Our goal is reducing hardware amount of last-level cache. We propose a method which compresses the data that has zero in upper bits.

1. はじめに

近年，キャッシュは大容量化している．微細化技術が向上し，より多くのトランジスタを搭載することが可能になっている．特に，最下層に位置するキャッシュであるラスト・レベル・キャッシュは大容量であり，より多くのデータを保持できる．例えば，最近のプロセッサである Intel の Core i7 は 8MB もの容量を持つ．そのため，コアよりも回路面積は大きく，チップ面積の半分以上を占有している．

ラスト・レベル・キャッシュの回路面積が巨大化することにより，消費電力の増大・レイテンシの増加を招いている[1)]．消費電力が増大する理由として，搭載されるトランジスタ数の増加が挙げられる．トランジスタ数が増加することでリーク電流は増加し，発熱量も大きくなる．また，レイテンシが増加する理由として，配線遅延の増加などが挙げられる[2)]．微細化にともなってゲート遅延よりも配線遅延の影響が大きくなり，レイテンシは増加する．実際に，Core i7 におけるラスト・レベル・キャッシュのレイテンシは最短で 30-40 サイクルにも及ぶ．

ところで，キャッシュに保持されるデータには規則性や冗長性などの偏りがあると考えられる．例えば，上位ビットが冗長なデータである場合，下位ビットだけをラスト・レベル・キャッシュに保持すれば良い．

我々は冗長なデータを圧縮することでラスト・レベル・キャッシュの回路面積を削減する方法を提案する．冗長なデータを圧縮できれば，ライン・サイズが縮小する．その結果，従来と同数のキャッシュ・ラインを保持してもラスト・レベル・キャッシュの回路面積は小規模にできる．小さな回路面積で実現できればレイテンシが短縮し，省電力化や性能の改善も期待できる．

以降，2 章ではデータ圧縮をキャッシュに適用した関連研究について述べ，3 章では提案手法の概要，ハードウェア構成について詳しく述べる．4 章では，予備評価について述べる．最後に，5 章でまとめる．

2. 関連研究

データを圧縮してキャッシュで保持する技術は過去にも研究されてきたが，それらはいずれもプロセッサ全体のパフォーマンス向上を狙ったものである．パフォーマンスを向上させるためにはキャッシュのヒット率が重要となる．ヒット率を向上させるためにはより多くのデータをキャッシュに保持すれば良い．しかし，容量を増やすとレイテンシが増加し，パフォーマンスに影響してしまう．そのため，データを圧縮して保持すれば，ハードウェア量を増加させずにヒット率を向上させることができる．

Gupta らは消費電力を抑えつつパフォーマンスを向上させる目的で，L1 キャッシュ内に保持されるデータを圧縮する手法を提案している[3)]．Gupta らはプログラム内で頻繁に使

†1 東京農工大学大学院 工学府

われるデータ（Frequent Value）が存在することに着目し，それらを圧縮して保持することでより多くのデータを保持できるとしている．

Gupta らによると，プログラムごとに Frequent Value は異なる．例えば，gcc では 0，-1，2 など 10 種類のデータがメモリの約 50%を占有していることがわかっている．そのため，１つのキャッシュ・ライン内に複数の Frequent Value が存在する可能性は高い．

Frequent Value を数ビットに圧縮すれば，より多くのデータを保持できる．数ビットのデータを保持した場合，従来よりもライン・サイズは小さくなる．ライン・サイズが小さくなることで，従来と同数のキャッシュ・ラインを保持しても全体の容量は小さくて済む．その分，より多くのキャッシュ・ラインを保持できる．

Gupta らが提案した手法では，1 つのラインを 2 つの圧縮ラインで共有することで，より多くのデータを保持している．Frequent Value がメモリを占める割合から，ライン・サイズを半分まで圧縮できるとしている．そのため，仮にキャッシュ・ラインに保持するデータが全て Frequent Value であった場合，2 倍のキャッシュ・ラインを保持できる．

一方，Wood らは復元にかかるペナルティに着目し，ラスト・レベル・キャッシュにおいてデータを圧縮するか動的に選択する手法を提案している[4]．元々ヒット率が高いプログラムに対しては，圧縮してより多くのデータを保持するメリットは少ない．データを圧縮してもヒット率は改善されず，むしろ復元によるペナルティの影響が大きくなってしまう．そのため，プログラムによって圧縮した効果とコストのバランスをとる必要がある．

動的に選択する手法は，圧縮した効果とコストでメモリ・アクセスを区別し実現している．メモリ・アクセスがあった場合はキャッシュを監視し，圧縮したためヒットしたか，圧縮しないためミスしたか，圧縮に関係なくヒットまたはミスしたかを調べる．圧縮に関係なくヒットまたはミスした場合，圧縮によってヒット率は改善されないため，復元によるペナルティが大きくなる．そのため，このようなメモリ・アクセスに対してはデータを圧縮しない．

3. 提 案 手 法

我々はデータを圧縮することでラスト・レベル・キャッシュの回路面積を削減することを提案する．上位ビットまで使用しない冗長なデータを圧縮し，キャッシュに保持する．その結果，従来よりも小さな記憶領域で同数のデータを保持できると考えられる．本章ではまず本手法の概要について述べる．次に構成・動作について詳しく述べる．

3.1 概　　要

データには冗長性があると考えられる．プログラムの多くは変数の初期化で 0 を使うことが多い．また，整数演算を行うプログラムでは上位ビットまで使用するデータの割合は低い．そのため，上位ビットが冗長であるデータは数多く存在すると考えられる．

上位ビットが冗長なデータである場合，下位ビットだけをラスト・レベル・キャッシュに保持すれば良い．例えば図 1 のように，書き込むキャッシュ・ラインが（0x00000000，0x00000001，0x000000002，0x12345678）を保持しているとする．それぞれのデータに着目すると，0x00000000，0x00000001，0x00000002 は上位ビットが 0 で連続していることが分かる．このような冗長性のあるデータは下位数ビットが必要な情報であって，上位ビットは必要な情報ではない．

圧縮データと非圧縮データを混合して 1 つのキャッシュ・ラインに保持すると，ライン・サイズが可変長になる．例えば，図 1 に示すキャッシュ・ラインに保持されているデータを圧縮すると，冗長性のあるデータは 2 ビットとなり，冗長性のないデータは 32 ビットになる．その結果，ライン・サイズは合計で 38 ビットとなる．仮にキャッシュ・ライン内のすべてのデータが冗長性のあるデータであった場合，8 ビットとなる．このように，圧縮できるデータの数によってライン・サイズは変化する．

サイズが可変長だとマッピングが複雑になってしまう．データの更新により圧縮サイズが変化したとき，データの再配置が必要になる．例えば，図 1 の先頭データが冗長性のないデータに更新されたとする．この場合，先頭データを圧縮できないため，圧縮データを格納していた領域に先頭データは保持できない．そのため，残りのデータの配置を変え，保持できる領域を確保する必要がある．また，どこまでが 1 つのラインか判別できないため，ラインごとにサイズを保持する必要がある．再配置が起きた場合，更新されたサイズも記録し直さなければならない．

よって，本手法では圧縮データは圧縮したライン・サイズで固定されたキャッシュで保持する．冗長性がみられず，圧縮できないデータは非圧縮データ専用の記憶領域に保持する．

3.2 構　　成

提案手法の構成を図 1 に示す．エンコーダ／デコーダは冗長なデータの圧縮・復元を行う．圧縮キャッシュは圧縮データを保持するキャッシュである．非圧縮データ専用 RAM は圧縮できなかったデータを保持する RAM である．フリーリストはエントリに空きがある非圧縮ラインのアドレスをリストで管理する．

それぞれに必要な容量はどれだけ上位ビットが 0 で連続しているかによる．そのため，本章では 32 ビット・データを N ビットまで圧縮できると仮定し，構成について説明する．どれだけ上位ビットが圧縮できるかは，後の 4 章で明らかにする．

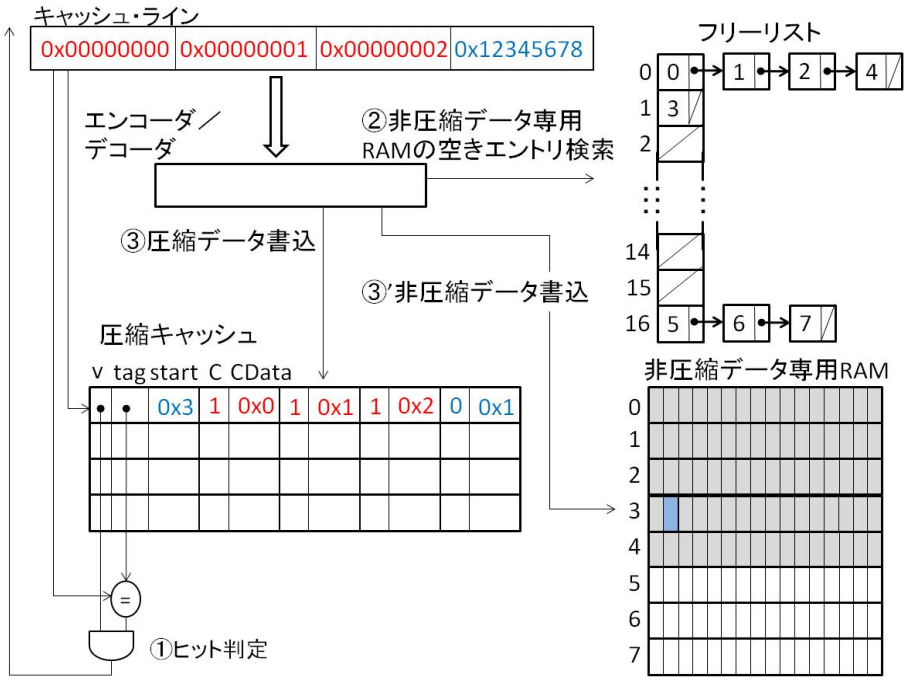

図 1 書き込みの動作

3.2.1 非圧縮データ専用 RAM

非圧縮データ専用 RAM のエントリ数は，N ビットまで圧縮できるデータが全データの何割占めているかに依存する．例えば，上位 28 ビットまで圧縮できるデータが全データの 6 割を占めていた場合，残り 4 割のデータを非圧縮データ専用 RAM で保持する．そのため，従来の 4 割のエントリ数で非圧縮データ専用 RAM を構成すれば良い．

非圧縮データはライン単位で管理し，保持する．非圧縮ラインのフィールドは図 2 のようになる．各非圧縮ラインは 2^{32-N} 個の非圧縮データを保持できる．また，各非圧縮データは有効フラグ（V）を持つ．有効フラグによって，エントリの空きを判別する．

3.2.2 圧縮キャッシュ

圧縮キャッシュは従来のラスト・レベル・キャッシュと同様，ライン単位でデータを管理する．ライン・サイズは，キャッシュ・ラインで保持するデータ数を従来と同じ 16 とすると，$16 \times (32 - N)$ ビットとなる．

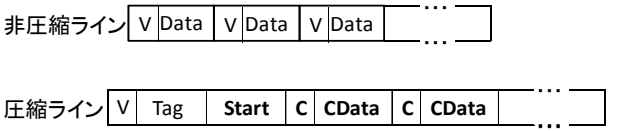

図 2 非圧縮ライン（上）と圧縮ライン（下）

圧縮ラインのフィールドは図 2 のようになる．追加したフィールドは圧縮フラグ（C），スタート・ビット（Start）である．圧縮フラグは圧縮データ（CData）であるかを示す．圧縮できなかったデータに対しては 0 を割り当てる．また，圧縮できなかったデータの領域は，データの代わりに非圧縮ライン内オフセットを保持する．スタート・ビットは非圧縮データ専用 RAM 内の非圧縮ライン・アドレスを保持する．

3.3 動　　作

上位のキャッシュから圧縮キャッシュに書き込む際，まず従来のキャッシュと同様にヒット判定を行う．ヒットした場合はそのエントリに圧縮ラインを書き込む．また，ミスした場合はリプレースが発生する．

ヒット判定を行った後，エンコーダ／デコーダによって書き込むラインに冗長なデータがあるか判別される．冗長性のあるデータの場合は上位 N ビットを取り除き，データを圧縮する．そして，圧縮ラインを圧縮キャッシュに書き込む．冗長性のないデータがあった場合，そのようなデータの個数に対応したフリーリストにアクセスし，空きがある非圧縮ラインのアドレスを取得する．その後，非圧縮データ専用 RAM にアクセスし，取得した非圧縮ラインに非圧縮データを書き込む．非圧縮データ専用 RAM への書き込みが終了した際，書き込んだ非圧縮ラインのアドレスと，非圧縮ライン内オフセットを圧縮キャッシュに書き込む．

圧縮キャッシュにラインを書き込む際の具体例を，図 1 を用いて説明する．圧縮キャッシュは 4 ライン保持でき，各ラインは圧縮データを 4 つまで保持できるものとする．非圧縮データ専用 RAM は 8 つの非圧縮ラインを保持でき，各ラインは非圧縮データを 16 個まで保持できると仮定する．フリーリストは非圧縮ラインで保持するデータ数に対応するため，16 でインデクス付けされている．

上位のキャッシュから図 1 に示すラインを圧縮キャッシュに書き込むとき，タグを比較しヒットしたとする．その後デコーダによってライン内に冗長でないデータ（0x12345678）が存在することがわかる．非圧縮データ専用 RAM に書き込むため，フリーリストにアクセ

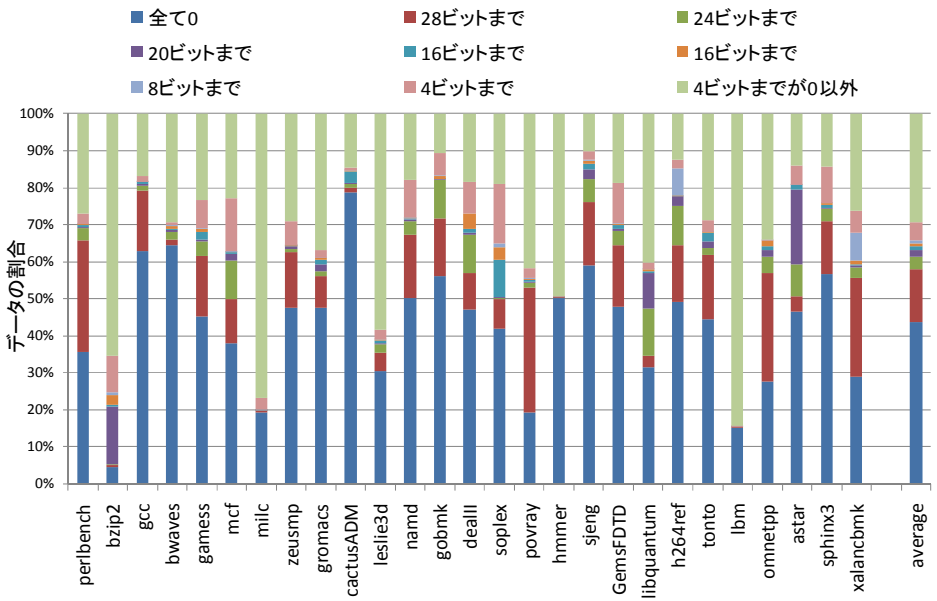

図 3 予備評価結果

スし，1 個空きエントリがある 3 番の非圧縮ラインのアドレスを取得する．そして，非圧縮データ専用 RAM にアクセスし，有効ビットが 0 のエントリに非圧縮データを書き込む．その後，圧縮ラインを圧縮キャッシュに書き込む．スタート・ビットには非圧縮ラインのオフセット（0x3）が，圧縮フラグが 0 のデータ部には非圧縮ライン内オフセット（0x1）が書き込まれる．

4. 予 備 評 価

我々は，どれだけ上位ビットが 0 で連続しているかを調査した．シミュレータは鬼斬弐[5] を用い，対象はラスト・レベル・キャッシュ（L2，4MB）とする．ライン・サイズは 64 Byte/Line とする．

27 本のベンチマークを実行した結果を図 3 に示す．平均を見ると，全データのうち，約 58%のデータは上位 28 ビットまで 0 で連続していることが分かった．そのため，上位 28 ビットまで圧縮することは効果的であり，残り 4 ビットだけラスト・レベル・キャッシュに保持すれば良い．キャッシュ・ラインに保持される全てのデータが上位 28 ビットまで 0 で連続している場合，ライン・サイズを 8 バイトまで圧縮することが可能である．

この結果より，非圧縮データ専用 RAM のエントリ数，圧縮キャッシュに追加したフィールドに必要なビット数を求めることができる．非圧縮ラインは 16（$= 2^{32-28}$）個のデータを保持できる．また，圧縮できないデータは 42%あるため，2MB の RAM があれば非圧縮データを保持するには十分であると考えられる．非圧縮データ専用 RAM を 2MB と想定すると，32K エントリ非圧縮ラインを保持できる．そのため，圧縮キャッシュのスタート・ビットは 15 ビットとなる．

提案手法に必要なビット数を求められたため，提案手法の回路面積を見積もることができる．追加したフィールドも含めて検証した結果，従来のラスト・レベル・キャッシュが 4MB であった場合，提案手法では 29.3%まで削減できることが分かった．

5. ま と め

我々はラスト・レベル・キャッシュの回路面積を削減することを目的として研究を行っている．本稿ではデータを圧縮することで回路面積を削減することを提案した．我々はデータの冗長性に着目し予備評価を行った結果，全データの 58%が上位 28 ビットまで 0 で連続していることが分かった．具体的なハードウェア構成を考え，圧縮できないデータも含めて回路面積を見積もった結果，従来の 29.3%回路面積を削減できることが分かった．

謝辞 本研究の一部は，文部科学省共生情報工学推進経費による．

参 考 文 献

1) Kim, N.S.: Circuit and microarchitectural techniques for processor on-chip cache leakage power reduction, PhD Thesis (2004). AAI3121968.
2) Beckmann, B. M. and Wood, D. A.: Managing Wire Delay in Large Chip-Multiprocessor Caches, *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture* (2004).
3) Yang, J., Zhang, Y. and Gupta, R.: Frequent value compression in data caches, *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture* (2000).
4) Alameldeen, A. R. and Wood, D. A.: Adaptive Cache Compression for High-Performance Processors, *SIGARCH Comput. Archit. News* (2004).
5) 塩谷亮太，五島正裕，坂井修一：プロセッサ・シミュレータ「鬼斬弐」の設計と実装，先進的計算基盤システムシンポジウム SACSIS2009 (2009).