Vol.2018-ARC-232 No.6
2018/7/30

マルチベクトルコアプロセッサの共有キャッシュ構成に関する一検討

高屋敷 光[†1,a)]　佐藤 雅之[†1,b)]　小松 一彦[†1,c)]　江川 隆輔[†1,d)]　小林 広明[†1,e)]

概要：本報告では，将来のベクトルプロセッサにおいてもコア数が増大することを見据え，高いメモリ性能要求に応えられるキャッシュ構成を検討する．コア数の増加に伴い増加するコンフリクトミスの削減を目的に，ハッシュ関数を用いてインデックスを取得する Skewed Cache 構成とその置換ポリシに着目し，マルチベクトルコアプロセッサにおける有用性を評価した．評価結果から，Skewed Cache 構成ではステンシル計算において理論値とほぼ同等のヒット率が得られることを明らかにした．

キーワード：HPC, ベクトルアーキテクチャ，キャッシュ，Skewed

Takayashiki Hikaru[†1,a)]　Sato Masayuki[†1,b)]　Komatsu Kazuhiko[†1,c)]　Egawa Ryusuke[†1,d)]
Kobayashi Hiroaki[†1,e)]

1. はじめに

ベクトル型スーパーコンピュータは高いメモリバンド幅を必要とする科学技術計算で広く用いられている．科学技術計算による数値シミュレーションは多様な学術分野で活用され，その適用範囲の拡大と高精度化・高度化が精力的に進められており，より高い性能を有するベクトル型スーパーコンピュータが強く求められている．

その要求に応えるため，プロセッサ内のコアを複数用意して演算性能を高めるマルチコア技術が採用されている．マルチコア技術は複数コアによる処理の同時並列実行を可能とすることから，演算性能を引き上げる一翼を担っている．一方，各コアの演算に必要なデータを供給するメモリシステムにおいては，メモリバンド幅の性能向上はチップに実装可能な入出力ピンの制約などにより限られている．このため，高いメモリデータ転送能力を維持するためにマルチバンクのキャッシュメモリが採用されている．これらの傾向はこれからも続くものと考えられ，将来のベクトル型スーパーコンピュータにおいても，より多くのコアや多くのバンクを持つキャッシュが搭載されると予想される．

しかし，ベクトル型スーパーコンピュータにおいて，多コア・多バンク構成となった場合の最適なキャッシュ構成は明らかになっていない．そこで本報告では，高いメモリ性能を要求されるステンシル計算カーネルを対象として，多数のコアを搭載するマルチベクトルコアプロセッサにおけるキャッシュ構成を検討する．予備評価では，コア数が増加するにつれて，共有キャッシュにおいてセット競合によるコンフリクトミスが増加し，キャッシュヒット率が大きく低下した．

そこで，多数のコアを搭載するマルチベクトルコアプロセッサにおいて，低連想度でもヒット率の低下を抑えるために，Skewed-Associativity[1] を用いた共有キャッシュ (以下，Skewed Cache) の構成を検討する．また，Skewed Cache の構成において，性能と実装コストに大きく影響を与えるハッシュ関数と置換ポリシを検討した．評価結果から，Skewed Cache 構成をとることにより，ステンシル計算カーネルにおいてほぼ理想的なヒット率を得られることがわかった．

本報告の構成は以下のとおりである．第 1 章では本報告の概要を述べた．第 2 章では，想定しているマルチベクトルコアプロセッサの概要について述べ，予備評価によりコンフリクトミスが増加しキャッシュヒット率が低下することを述べる．第 3 章では本報告で用いる Skewed-Associativity

†1 現在，東北大学
Presently with Tohoku University
a) hikaru.takayashiki.p5@dc.tohoku.ac.jp
b) masa@tohoku.ac.jp
c) komatsu@tohoku.ac.jp
d) egawa@tohoku.ac.jp
e) koba@tohoku.ac.jp

ⓒ 2018 Information Processing Society of Japan

を用いたキャッシュについて検討する．第 4 章ではシミュレーションによる評価を行い，その結果について議論・考察する．第 5 章ではまとめを述べる．

2. マルチベクトルコアプロセッサにおける課題

ベクトル型スーパーコンピュータに搭載されているベクトルプロセッサは，ベクトル演算器をもち，その演算能力を発揮できるように設計されたアーキテクチャを持つプロセッサである. ベクトル型スーパーコンピュータはデータレベルでの並列性を利用し，高速かつ高効率で計算を行うことが可能である．近年，シングルコアでの性能向上の限界からマルチベクトルコア化が進んでおり，そのコア数は増加傾向にある [2], [3].

ベクトル型スーパーコンピュータは演算性能とバランスした高いメモリバンド幅を実装することで，高い演算性能と実行効率を実現している．しかし，半導体製造技術の微細化，マルチコア化により演算性能が飛躍的に向上する一方で，メモリ性能は伸び悩んでいることから，ベクトルアーキテクチャにおいてメモリバンド幅が性能のボトルネックとなっている．このメモリバンド幅不足を補うために，近年ベクトルプロセッサ向けのキャッシュメモリが搭載されている．

2.1 想定するマルチベクトルコアプロセッサの概要

図 1 に本報告で想定するベクトルプロセッサの構成を示す．本ベクトルプロセッサはメインメモリと多数のキャッシュバンク，多数のベクトル演算コアにより構成される．ベクトルコアは大きく分けて，スカラプロセシングユニットとベクトルプロセシングユニットから構成される．スカラプロセシングユニットは命令のフェッチ・デコード及び，命令発行，およびスカラ演算を行う．

ベクトルプロセシングユニットは，ベクトル演算器及びロードストアユニットとベクトルレジスタから成る．ベクトル演算器は，複数のデータに対して一括して演算を行うことで，要素のデータに対する演算を単一のベクトル命令で実行可能とする．例えば，配列同士を加算する場合，1 要素ずつ逐次に演算するのではなく，最大 256 要素を一命令で演算・処理する．これはスカラプロセッサに拡張機能として搭載されるベクトル拡張命令と比較して，大量のデータを一度に計算できる．

複数の演算を一度に処理するには，複数のデータを一括して演算器に供給する必要がある．そのためベクトルプロセシングユニットは，ロードストアユニットを持つ．ロードストアユニットはベクトルロード/ストア命令からメモリアドレスを生成し，最大 256 要素分のメモリアクセス要求を同時にメモリシステムへ送る．メモリシステムから要求されたデータが送られてくると，そのデータはロードストアユニットに格納され，全てのデータが揃い次第ベクトルレジスタへと転送される．

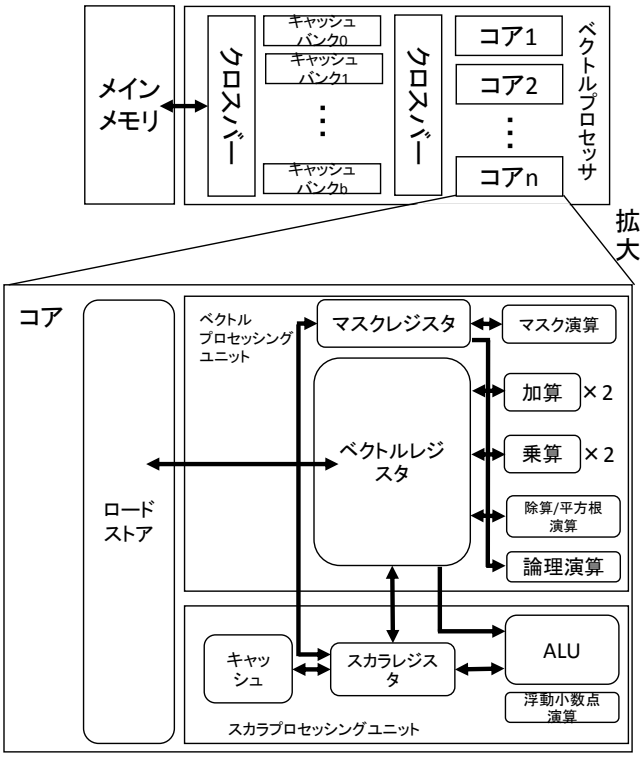

図 1　ベクトルアーキテクチャの概念図

ベクトルレジスタは，各ベクトル演算に使われるオペランドを格納するレジスタである．ベクトルレジスタからベクトル演算器へパイプライン的にデータを供給することによってベクトル演算器を常に稼働させ，効率良く計算を行うことが出来る．ベクトル演算器には加算・乗算などを担当する演算器が複数個搭載されており，それぞれの演算器は独立して動作する．そのため，複数種類の演算を並列して行うことが可能である．

ベクトルマスクレジスタは，ベクトル演算の実行結果をマスクするためのレジスタである．ベクトルマスクレジスタを利用することで，条件分岐などが含まれるループをベクトル演算を用いて一括して投機的に実行できる．

近年，演算性能への高い要求により，ベクトルプロセッサにおいてもマルチコア技術が採用されている．ベクトルコアを複数持つことにより，スレッドの同時並列実行が可能になるため演算性能が向上する．NEC SX-ACE 以降では複数のベクトルコアを搭載しており，SX-ACE では 4 つ，最新機の SX-Aurora TSUBASA は 8 つのベクトルコアが搭載されている．このコア数の増加傾向は今後も続くと考えられる．

2.2 マルチベクトルコアプロセッサのキャッシュ構成

演算性能の向上に伴い，データ供給能力への要求は更に高まっている．しかし，演算性能の向上に対してメモリ性能の向上は遅れていることから，近年ではメモリア

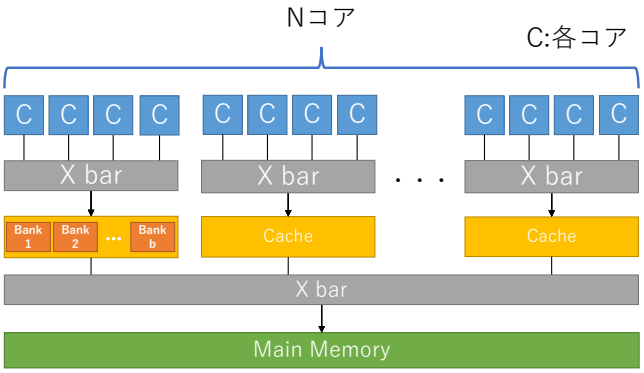

図 2　4 コア共有キャッシュの例

クセス性能を改善するために，ベクトルプロセッサ向けのキャッシュメモリが搭載されている．例えば，NEC SX シリーズにおいて SX-9 以降のベクトルプロセッサでは，Assignable Data Buffer (ADB) と呼ばれるオンチップメモリをプロセッサ内部に備えている [2], [4]．ADB はメモリインターリーブ方式を採用し，マルチバンク構成である．また，ADB は内部的に，セットアソシアティブ方式を採用したキャッシュメモリである．本報告で想定するマルチベクトルコアプロセッサに搭載されるキャッシュメモリの構成は，ADB と同様であるものとする．

将来的にコア数が増大した場合，チップの面積には限りがあるため，コアとキャッシュを接続するオンチップネットワークのコストは抑制されなければならない．このため，本報告では図 2 に示すように，一部のコア間でのみキャッシュを共有するような構成を想定する．図 2 は総コア数が N コアある場合に，4 コアごとにキャッシュを共有した例である．

N コアを搭載するマルチベクトルコアプロセッサにおいて，1 つのキャッシュに M コアずつ接続する場合，プロセッサ内のキャッシュ数 C は次式で表される．

$$C = \frac{N}{M} \tag{1}$$

ここで，M は図 2 においては，一つのキャッシュに接続しているコア数 4 を意味する．N は一定とすれば，式 (1) よりキャッシュへ接続するコア数 M によってシステム内のキャッシュ数が変わることが分かる．

さらに，キャッシュの合計バンク数 B は固定とし，1 つのキャッシュが持つバンク数 b を次式で表す．

$$b = \frac{B}{C} = \frac{BM}{N} \tag{2}$$

N と B は一定であるから，式 (2) よりキャッシュへ接続するコア数 M とキャッシュ当たりのバンク数 b が変化する．容量はバンク当たり固定であるため，キャッシュ当たりのバンク数によってキャッシュ容量が決定される．

2.3　マルチベクトルコアプロセッサにおけるキャッシュ構成の現状と課題

これまでベクトルプロセッサにおけるキャッシュの有用性を検証する研究は数多く行われている [2], [5], [6], [7], [8]．撫佐ら [6] はマルチベクトルコアプロセッサにおいて，4 コアにおける共有キャッシュを検証し，キャッシュを 2 コアと 4 コアで共有することにより，キャッシュ無しの場合と比較して 15〜40%ほど性能向上できることを示している．これはキャッシュをより多くのコアで共有することで，隣接コア間のデータを再利用することが出来るためである．それにより，キャッシュのヒット率が向上し，性能向上が得られることを示している．

一方で撫佐らによる共有キャッシュの検討は，4 コアまでしか行われていない．現代のプロセッサのコア数は増加傾向にあり，今後も増え続けると考えられる．そのため，さらに多くのコア数で検討が必要である．また，キャッシュ設計にあたって連想度が考慮されていない．

江川ら [7], [8] はマルチベクトルコアプロセッサにおいて，16 コアまでの共有キャッシュを実アプリケーションを用いて検証し，コア数および共有キャッシュ容量を増加させることによりキャッシュヒット率を高め，実行効率を向上させることで消費電力を削減できることを示している．また，検討の結果，16 コアにおいては 8MB の共有キャッシュ構成が最も効率の良い計算が出来るとしている．

しかし，これらの共有キャッシュの検討はベースアーキテクチャが SX-8 であり，想定しているコア性能やメモリバンド幅も，今後予想されるマルチベクトルコアプロセッサとは乖離している可能性がある．また，連想度が考慮されておらず，コア数およびキャッシュ容量を変化させた評価に留まっている．

キャッシュの連想度は一般的に，高いほどヒット率が得られることが知られている．今後コア数が増加していく中で共有キャッシュにより性能向上を得るためには，連想度によるヒット率の変化も考慮する必要がある．そこで，本報告では科学技術計算で用いられるステンシル計算カーネルを用いて，共有キャッシュ構成の検討を行う．

2.4　7 点ステンシル計算を用いたキャッシュ構成の検討

2.4.1　7 点ステンシル計算の概要

ステンシル計算は科学技術計算における代表的な計算カーネルで，熱伝導計算をはじめとしたさまざまな計算の中核を担っている．ステンシル計算のような代表的な計算カーネルを用いて共有キャッシュ構成を検討することで，マルチベクトルコアプロセッサの性能向上に繋がると考えられる．

7 点ステンシル計算の一例の疑似コードを Algorithm 1 に示す．本報告で用いる 7 点ステンシル計算は，ある要素と x,y,z 軸方向の隣接する要素の計 7 点の算術平均を算出

Algorithm 1 7 点ステンシル計算

```
for z = 0, 1, ...Nz do
  for y = 0, 1, ...Ny do
    for x = 0, 1, ...Nx do
      b[z][y][x]=(a[z][y][x]+a[z][y][x-1]+a[z][y][x+1]+
                 a[z][y-1][x]+a[z][y+1][x]+
                 a[z-1][y][x]+a[z+1][y][x] )/7
    end for
  end for
end for
```

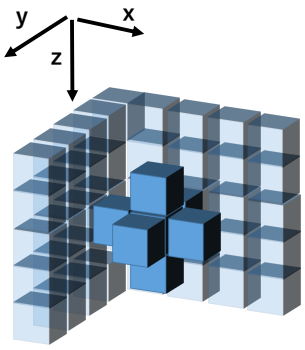

図 3　7 点ステンシル計算概念図

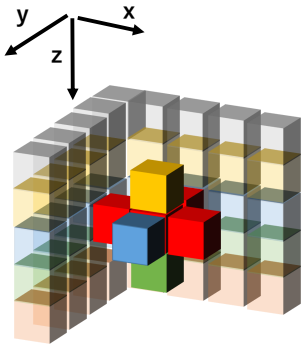

図 4　キャッシュを共有した場合の 7 点ステンシル計算概念図

する．これを x,y,z 軸方向に繰り返す．7 点ステンシル計算においては，ある要素と隣接している要素の計 7 点が計算に利用されることになる．

図 3 に 7 点ステンシル計算の計算する要素の概念図を表す．図 3 の半透明なブロックは計算空間内の要素を表しており，色が濃い要素を計算に用いることを示している．中心の注目している要素とその隣接している要素を用いて計算し，次に隣の要素を中心として隣接要素を用いて計算する．これをすべての空間において繰り返す．

2.4.2　共有キャッシュを考慮した 7 点ステンシル計算

ステンシル計算は隣接要素への連続的なアクセスを行うことから，あるスレッドが隣接する要素にアクセスしていれば，そのデータがキャッシュ内に既に配置されている可能性が高い．このとき，他のスレッドがデータを再利用するためには，共通データをキャッシュで共有出来るように並列化を行う必要がある．そのため，最外ループ毎の計算を各コアに割り当てるように並列化を行う．ここで，キャッシュを共有するコア数を m として，最外ループの k 番目の繰り返しを担当するコア番号を $c\,(c = 0, 1, 2, ..., m-1)$ とすると，その関係は以下の通りとなる．

$$c = k \mod m \tag{3}$$

なお，　mod　は剰余演算を示す．

次に 7 点ステンシル計算のヒット率の理論値を算出する．図 4 に 7 点ステンシル計算においてどの要素がキャッシュにヒットするかを示す．図 4 において，赤色はキャッシュを共有しない場合でもキャッシュヒットする箇所を示す．濃い黄色と緑色の要素はキャッシュを共有した場合に，それぞれ他のコアがキャッシュにデータを持ってくることによって更にキャッシュヒットする箇所を示す．濃い紺色のブロックはキャッシュの共有・非共有に関わらずキャッシュミスするブロックを示している．半透明なブロックは，各コアで計算を担当する領域の違いを示している．

x 軸方向にループが進んだ後に，y 軸方向に進行するので，キャッシュを共有しない場合は a[z][y][x]，a[z][y][x-1]，a[z][y][x+1]，a[z][y-1][x] の 4 点がキャッシュにヒットする．そのため 4/7(57.14%) のヒット率が得られる．なお．本報告において，z 軸方向の一層分のデータはキャッシュに載りきらないと仮定する．

一方，キャッシュを共有する場合は，上記のデータの再利用に加えて，隣接コアのデータもアクセス可能になる．よって，a[z-1][y][x]，a[z+1][y][x] の 2 点が追加でヒットする．しかし，端部のコアでは隣接するコアが 1 つしかないため，a[z-1][y][x]，a[z+1][y][x] のどちらか片方が必ずミスする．これより，7 点ステンシル計算は最大で 2/7(28.57%) のキャッシュ間共通データを持つ計算カーネルである．

ここで，キャッシュを共有するコア数を m とすると，z ループの m 回の繰り返しを m コアで並列計算する場合，$7m$ 回のアクセスがある．端部以外のコアは 6 要素へのアクセスがキャッシュヒットになるが，端部にあたる 2 つのコアのキャッシュヒットは 5 要素になる．これより，全コアでは，1 回のステンシル計算におけるヒット数は $6(m-2)+5\times 2$ となる．よって，7 点ステンシル計算のヒット率 H_7 は

$$H_7 = \frac{6(m-2)+5\times 2}{7m} = \frac{6}{7} - \frac{2}{7m} \tag{4}$$

と表すことが出来る．

これより，ステンシル計算においては共有キャッシュ構成を考えることで，共有するコア数から理論的にヒット率を予想することが出来る．式 (4) から，コア数 m を増やせば増やすほど 6/7(85.71%) に漸近することが分かる．そこで，マルチベクトルコアプロセッサのシミュレーションによりヒット率の検討を行う．

2.4.3　予備評価

図 2 に示した構成におけるキャッシュを，複数のコアで共有することを考える．共有する際は前節で検討したようにデータを共有できるように並列化する．ステンシル計算においては，式 (4) よりコア数が多いほど高いヒット率が予想出来る．一方で，連想度もキャッシュの設計において考慮する必要がある．予備評価では，1 つのキャッシュを共有するコア数を 1, 2, 4, 8, 16, 32 と変化させ，キャッシュの連想度は 4，8，16，32 と変化させた場合に，キャッシュヒット率がどのように変化するかを検証する．

予備評価の結果を図 5 に示す．このとき，縦軸はキャッシュヒット率，横軸は 1 つのキャッシュを共有するコア数と連想度を示す．図 5 から分かるように，理論値と比べて，

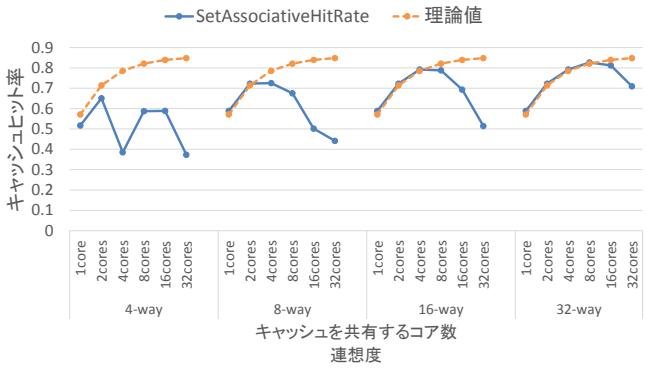

図 5　ヒット率と連想度の関係

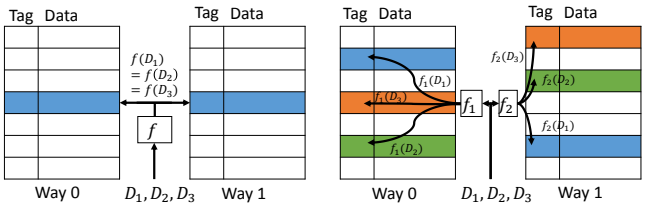

図 6　2-way セットアソシアティブキャッシュと 2-way Skewed Cache の概念図

共有コア数が多く連想度が低いほどキャッシュヒット率が低下している．これは，7 点ステンシル計算を共有キャッシュでデータを共有できるように並列化して実行した場合，同一セットへのアクセスが複数コアから同時に発行され，コンフリクトミスが発生するためである．

コンフリクトミスを減らすための方法の一つとして，連想度を高めることが挙げられる [9]．しかし，大容量かつ多バンクのキャッシュにおいて，コストや消費電力，チップ上での面積の問題から，高い連想度の実現が難しい．そのため，本報告では連想度を高めること無く，キャッシュのコンフリクトミスを減らす方法について検討を行う．

3. Skewed-Associativity を用いたマルチベクトルコアプロセッサのキャッシュ構成

Skewed-Associativity [1] は連想度を高めずにコンフリクトミスを減らすための手法である．本報告では，マルチベクトルコアプロセッサ向けのキャッシュ構成において低連想度でコンフリクトミスを減らすために，Skewed Cache を採用した共有キャッシュ構成を検討する．

Skewed Cache をマルチベクトルコアプロセッサにおけるキャッシュとして用いることによって，連想度を高めずコンフリクトミスを減らし，高いキャッシュヒット率を目指す．

3.1 Skewed Cache の概要

Skewed Cache はそれぞれのウェイにおいて別の出力をするハッシュ関数を用いてセットを選ぶことにより，コンフリクトミスを削減する．図 6 に 2-way セットアソシアティブキャッシュと 2-way Skewed Cache の違いを概念図

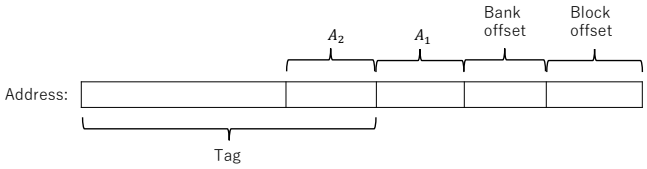

図 7　oDisp に用いるビットフィールド

で示す．図 6 において，アドレス D_1,D_2,D_3, はセットアソシアティブキャッシュにおいて同一のセットに配置される．そのため，コンフリクトミスにより，最初に挿入された D_3 は追い出されてしまう．一方で，Skewed Cache においては，それぞれのウェイで用いられるハッシュ関数が異なるため，別のセットへと配置される．その結果，セットアソシアティブキャッシュに比べて大幅にコンフリクトミスを削減可能である．

Skewed Cache を用いてヒット率の向上を得るためには，ハッシュ関数と置換ポリシの 2 点が重要となる．まず，ハッシュ関数である．Skewed Cache の性能は，ハッシュ関数に大きく影響される．ハッシュ関数の出力は偏りなく均等であるのが望ましい．偏りがある場合，同じセットに配置されることが多くなりコンフリクトミスを抑制できない．さらに，それぞれのウェイにおいて異なる出力をするハッシュ関数を簡単に作れることが望ましい．

次に，置換ポリシである．Skewed Cache の大きな難点の一つとして，2-way より大きい連想度では置換ポリシに Least Recently Used (LRU) ポリシを実装するのが難しい点が挙げられる．一般的な LRU は挿入順序を利用して実装される．セットアソシアティブキャッシュは置換が起きる際，全てのウェイで同一のセットが置換候補となるため，同一セット間での挿入順序だけを保存しておけばよい．しかし，Skewed Cache はアクセスされるアドレスによってウェイごとに対応するセットが異なるため，挿入順序を用いて置換対象を決めることが難しい．全ブロックの挿入順序が保存できれば LRU を実現できるが，ブロックごとにタイムスタンプ等を保存する必要がありコストが嵩んでしまう．

Skewed Cache の性能と実装コストに大きく影響を与えるハッシュ関数と置換ポリシの実装について，次節以降詳細な検討を行う．

3.2 ハッシュ関数

一般的なハッシュ関数としては，排他的論理和 (Exclusive Or, XOR) を用いたハッシュ関数が用いられる．しかし，XOR を用いたハッシュ関数は，特定の長さのストライドを入力した場合，出力に偏りが生まれてしまうことが知られている [10]．また，XOR を用いたハッシュ関数はそれぞれのウェイで異なる出力を得るために，ビットローテー

ⓒ 2018 Information Processing Society of Japan

ション[*1]などを用いて入力ビットを変更する．しかし，これらの処理は入力ビット数が少ない場合，やがて一巡して同じ入力となってしまう．

そのため，本報告ではハッシュ関数に，Odd-Multiplier Displacement Hashing Function (oDisp) [10] を用いる．oDisp は XOR を用いたハッシュ関数に比べ，多くの入力に対して分布が一様で偏りが少ない．そのため，XOR を用いたハッシュ関数が苦手とするような特定長のストライドを含むアクセスパターンにおいても，oDisp によって更にコンフリクトミス削減を期待できる．

図 7 に oDisp におけるインデックス算出に用いられるアドレスフィールドを示す．また，oDisp を用いて取得したインデックスは次式で表現される．

$$index = (o \times A_2 + A_1) \bmod n_{set} \quad (5)$$

ここで，o は任意の奇数，A_1, A_2 は図 7 におけるアドレスのフィールドを示し，n_{set} は各ウェイにおけるセット数を示す．なお，A_1,A_2 のビット長は $\log_2(n_{set})$ である．

本ハッシュ関数の特徴として，ハードウェアコストが非常に少ないことが挙げられる．式 (5) は任意の奇数と A_2 をかけたものを A_1 と足し合わせ，セット数で余りを求めることを意味する．任意の奇数を選んで乗算し足し合わせるには，乗算器などは用いず論理シフトと加算器 2 つで実現可能である．さらに，それぞれのウェイにおいて異なる奇数を選ぶことで，ウェイごとにハッシュ関数を変えるのも容易である．

3.3 置換ポリシ

LRU に代わる Skewed Cache 向けの置換ポリシとして，Not Recently Used (NRU) をベースとした Not Recently Used Not Recently Written (NRUNRW) [11] が F. Bodin らにより提案されている．NRUNRW の詳細な動作を図 8 に示す．これはキャッシュブロック当たり 1 ビットを Recently Used Bit として付加することによって，最新のキャッシュブロック以外を置換対象として選ぶ．もし判別が出来ない場合はダーティではないブロックを選択するポリシである．また，Recently Used Bit はキャッシュにアクセスされた回数が $Cache\,block\,size/4$ に達するたびに全てリセットされる．

NRUNRW の利点は，キャッシュブロック当たり 1bit と少ないハードウェアコストで実現できる点である．しかし，このポリシは NRU にランダムリセットを組み合わせたポリシであり，最終的には LRU に及ばないことが知られている [12]．

そこで，本報告では現実的なハードウェアコストで LRU と同等のヒット率を得るために，Skewed Cache に用いる

[*1] ビット列のシフトを行い，最後のビットを最初のビットに移動させる処理

ブロック挿入時及びキャッシュヒットした場合
(1) Recently Used Bit を 1 にセット
キャッシュミスした場合
次の優先度の順に置換対象が選ばれる．
(1) 候補のセットの中から Recently Used Bit が 0 のブロックをランダムに選ぶ
(2) Recently Used Bit が 1 かつダーティではないブロックをランダムに選ぶ
(3) Recently Used Bit が 1 かつダーティなブロックをランダムに選ぶ
キャッシュアクセス回数
$Cache\,block\,size/4$ に達した場合，すべてのブロックの Recently Used Bit に 0 をセット

図 8 NRUNRW ポリシの振る舞い

キャッシュヒットした場合
(1) RRPV に 0 をセットする
キャッシュミスした場合
(1) 候補のセットから RRPV が 3 のブロックを探す
(2) もし RRPV が 3 のブロックが見つかったらステップ 5 へ
(3) 候補のセット全ての RRPV を 1 加算
(4) ステップ 1 に戻る
(5) ブロックを置換して RRPV を 2 にセット

図 9 SRRIP ポリシの振る舞い

置換ポリシとして Static Re-Reference Interval Prediction (SRRIP) ポリシ [13] を検討する．SRRIP は NRU ポリシを拡張した形で実装される置換ポリシで，単純なアルゴリズムと低ハードウェアコストで LRU に劣らないヒット率を得られる．SRRIP の処理の流れを図 9 に示す．SRRIP はキャッシュブロック当たり M-bit の Re-Reference Prediction Value (RRPV) を用意し，そのブロックが次回のいつ参照されるのかを予測する．挿入する際には，該当ブロックは近いうちに置換されると予想し，ヒットした場合のみまた参照されると予想する．そのため，キャッシュヒットしたブロックのみがキャッシュ内にとどまり，それ以外のブロックは置換される．本報告ではハードウェアコストを抑えるために，RRPV は 2 ビットとした．

SRRIP ポリシにより，キャッシュヒットしたブロックは追い出されにくくなり，キャッシュヒットしないブロックは比較的早い段階で追い出されるようになる．また，キャッシュブロックごとにビットを保持するため，Skewed Cache のように対応するセットがウェイごとに異なる場合でも置換すべきブロックを判断できると考えられる．

4. 性能評価

4.1 評価環境

マルチベクトルコアプロセッサにおける Skewed Cache

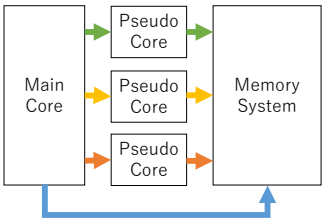

図 10　疑似コアシミュレータの概念図

表 1　シミュレータのシステム構成

ベースアーキテクチャ	NEC SX-ACE
コア数	32
キャッシュを共有するコア数	1,2,4,8,16,32
メインメモリ	DDR3-SDRAM
ベクトルキャッシュ	SRAM
キャッシュ合計サイズ (1 バンク当たり)	32MB(128KB)
連想度	4,8,16,32
ブロックサイズ	128Bytes
MSHR 数 (各 Target 数)	8(8)
システム B/F	0.25B/F

の効果を確認するために，ベクトルアーキテクチャのシミュレータを用いて評価を行う．シミュレータは汎用アーキテクチャシミュレータである gem5 [14] のフレームワークに基づいて開発されている．シミュレータは，ベクトル型スーパーコンピュータ SX-ACE が出力するトレースデータを入力として受け取る．トレースデータをもとにプロセッサの内部の各ユニット占有状態をシミュレートし，性能や実行時間等を算出する．シミュレータはベクトル演算器の動作とメモリアクセスの発行をシミュレートするプロセッサ部と，キャッシュ及びメインメモリの挙動をシミュレートするメモリ部に大きく分かれている．

マルチベクトルコアプロセッサのシミュレーションは上記シミュレータのプロセッサ部に疑似コアを実装することで行った．疑似コアを用いたシミュレーションの概念図を図 10 に示す．疑似コアとはメモリアクセスのみを発行するコアであり，シンプルな実装により高速なシミュレーションが可能である．疑似コアはメモリアクセスを発行する際にメインコアからアドレスを受け取り，特定のシフト幅だけアドレスをシフトする．これにより，メインコアと同じアクセスパターンを違うアドレスに対して行うことが出来る．なお，本報告で対象とするステンシル計算では，Algorithm 1 の最外ループである z ループで並列化を行うことを仮定する．このためシフト幅は z ループ 1 回分のデータ量とした．

想定するマルチベクトルコアプロセッサのシステム構成を表 1 に示す．コア数は将来のマルチベクトルコアプロセッサを考慮し 32 コアを搭載しているものとし，コア内の構成は NEC 社の SX-ACE に準ずるものとする [2]．また，SX-ACE の Byte/Flop 値 (以下 B/F 値) は 1.0，SX-Aurora TSUBASA の B/F 値は約 0.5 であることを踏まえ [3]，今後も演算性能とメモリ性能の乖離は進むと仮定し，評価に用いるシステムの B/F 値は 0.25 に設定する．

キャッシュの連想度は 4, 8, 16, 32，キャッシュを共有するコア数は 1, 2, 4, 8, 16, 32 として評価を行う．キャッシュのバンク数は全体で 256 個とし，バンク当たりの容量は 128KBytes とした．キャッシュ容量は各コアあたり 1MB(8 バンク) の容量とし，共有キャッシュの場合はコア数 ×1MB の容量とした．

キャッシュは Skewed Cache のほかに，比較対象として通常のセットアソシアティブキャッシュを用意する．置換ポリシは本報告の検討対象である SRRIP の他，比較対象として NRUNRW と LRU を実装した．なお，Skewed Cache において挿入順序を利用した LRU 置換は実現困難なため，タイムスタンプベースで LRU 置換を行う理想的なポリシとした．

4.2　評価結果

セットアソシアティブキャッシュ及び Skewed Cache の各ポリシにおいて，共有するコア数を変化させた場合の各連想度におけるヒット率を，図 11 に示す．図 11 において縦軸はヒット率を表し，横軸はキャッシュを共有するコア数と連想度を示している．

まず，セットアソシアティブキャッシュと Skewed Cache+SRRIP を比較する．セットアソシアティブキャッシュでは連想度が低い場合にはヒット率が得られていないが，Skewed Cache にすることによってヒット率の向上が得られていることが分かる．また，Skewed Cache+SRRIP のヒット率は，理論値に非常に近い．特に，連想度が低く共有するコア数が多い場合はヒット率の差が顕在化する．連想度 4,8 で 32 コアでキャッシュを共有した場合，40%以上のヒット数向上が得られているほか，連想度 16 で 32 コア共有をした場合においても約 30%のヒット数向上が得られている．これより，Skewed Cache+SRRIP が低連想度のままコンフリクトミスを削減するために有効であることを示している．

次に，セットアソシアティブキャッシュと Skewed Cache+LRU を比較する．LRU ポリシを用いた場合，Skewed Cache を用いることで，セットアソシアティブキャッシュや Skewed Cache+SRRIP よりも高いヒット率を維持可能であることが分かる．連想度が 4 の場合，32 コアで共有した時にセットアソシアティブキャッシュより Skewed Cache が最大で 40%以上キャッシュヒットが増加しており，連想度 32 の場合においても，キャッシュを 32 コアで共有する場合に 10%以上のヒット数向上が得られている．これは，タイムスタンプベースの LRU を採用した Skewed Cache+LRU ではブロックが理想的に置換され，連想度に依らずコンフリクトミスを削減可能なためである．

ⓒ 2018 Information Processing Society of Japan

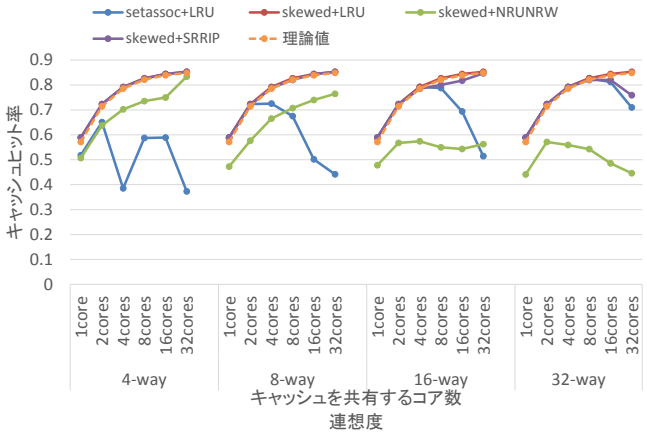

図 11 キャッシュを共有するコア数を変化させた場合のヒット率

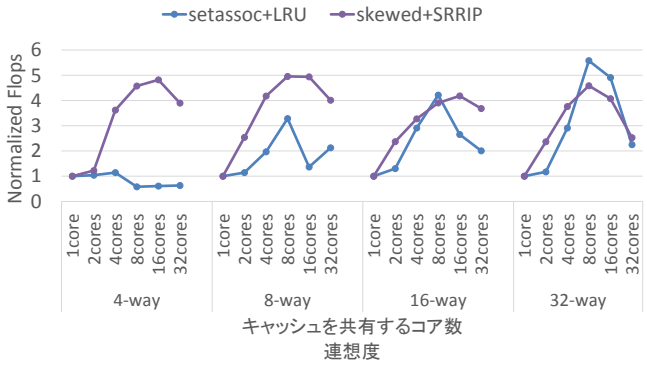

図 12 キャッシュを共有するコア数を変化させた場合の性能

次に，Skewed Cache のそれぞれのポリシについて比較する．まず，LRU は理論値とほぼ同等のヒット率を達成していることが分かる．また，SRRIP も連想度が 4 と 8 の場合には理論値に追従していることが分かる．一方で，連想度が 32 の場合には，キャッシュを共有するコア数が増加するにつれて，SRRIP では徐々にヒット率が低下する．これは連想度が多い場合，容量が同等であればウェイ当たりのブロック数が減少する．そのため，Skewed Cache が各ウェイでハッシュ関数によって選ぶことが出来るセットが減り，同じセットが何度も選ばれる可能性が高くなる．SRRIP では，あるブロックが候補として何度も選ばれると，RRPV が加算される機会が増えて置換対象として選ばれる可能性が高くなることから，必要なブロックが追い出されてしまったのではないかと考えられる．また，NRUNRW はいずれの場合においても低いヒット率となった．これは，Recently Used Bit が一定アクセス毎にリセットされるため，そのたびにランダムに置換ブロックを選択されるためだと考えられる．

図 12 に共有するコア数を変化させた場合の各連想度における性能の変化を，それぞれのキャッシュ構成において評価したものを示す．ここで，セットアソシアティブキャッシュと Skewed Cache+SRRIP を比較する．セットアソシアティブキャッシュは一般的に用いられるキャッシュ構成であり，Skewed Cache+SRRIP は現実的な実装コストで高いヒット率を実現可能な共有キャッシュ構成である．図 12 において縦軸は演算性能を表しており，それぞれの連想度においてキャッシュを私有 (1 コアで共有) した場合で正規化した性能である．横軸はキャッシュを共有するコア数と連想度を示している．図 12 より，連想度が低い場合においては Skewed Cache がセットアソシアティブキャッシュよりも高い性能を示している．一方で十分に連想度がある場合は，Skewed Cache とセットアソシアティブキャッシュはそれほど性能差が存在しない．これは，連想度が十分あればコンフリクトミスはそれほど起こらないためだと考えられる．

また，各連想度で共有するコア数が 16 と 32 の場合において，32 コアの場合のヒット率が高いにもかかわらず，16 コアの場合の性能の方が高い．これは，キャッシュバンクのコンフリクトが原因である．本報告で想定しているマルチベクトルコアプロセッサのキャッシュ構成は，マルチバンク構成を想定しているため，メモリアドレスに応じてそれぞれのキャッシュバンクへアクセス要求が行われる．そのため，多くのコアが一つのキャッシュを共有した場合，同一バンクにアクセス要求を発行し，コンフリクトを起こしてしまう可能性が高まる．また，今回のキャッシュ構成では，バンク当たりのライトバッファサイズは 16 である．そのため 32 コアで共有した場合に，コンフリクトにより一部のバンクにアクセスが集中すると不足が起こり，性能低下を引き起こしていると考えられる．

5. おわりに

マルチベクトルコアプロセッサにおいてコア数が増加した場合，連想度の不足によりコンフリクトミスが増加してキャッシュヒット率が低下してしまう．そこで，本報告では，マルチベクトルコア向け共有キャッシュに，Skewed-Associativity を採用し，置換ポリシに SRRIP ポリシを用いた Skewed Cache を検討した．シミュレーションを用いた評価の結果，連想度を低くした場合においてもコンフリクトミスを削減し，ヒット数が最大で 40%ほど向上することを明らかにした．今後の課題としてハードウェアコストの見積もりや，エネルギの見積もりを組み合わせて，最適なキャッシュ構成を見出すことなどが挙げられる．

参考文献

[1] Seznec, A.: A Case for Two-way Skewed-associative Caches, *Proceedings of the 20th Annual International Symposium on Computer Architecture*, ISCA '93, New York, NY, USA, ACM, pp. 169–178 (online), DOI: 10.1145/165123.165152 (1993).

[2] Egawa, R., Komatsu, K., Momose, S., Isobe, Y., Musa, A., Takizawa, H. and Kobayashi, H.: Potential of a Modern Vector Supercomputer for Practical Applications: Performance Evaluation of SX-ACE, *J. Supercomput.*, Vol. 73, No. 9, pp. 3948–3976 (online), DOI:

10.1007/s11227-017-1993-y (2017).

[3] NEC Official Press: SX-Aurora TSUBASA Features. http://jpn.nec.com/hpc/sxauroratsubasa/features/.

[4] Soga, T., Musa, A., Okabe, K., Komatsu, K., Egawa, R., Takizawa, H., Kobayashi, H., Takahashi, S., Sasaki, D. and Nakahashi, K.: Performance of SOR methods on modern vector and scalar processors, *Computers & Fluids*, Vol. 45, No. 1, pp. 215 – 221 (online), DOI: https://doi.org/10.1016/j.compfluid.2010.12.024 (2011). 22nd International Conference on Parallel Computational Fluid Dynamics (ParCFD 2010).

[5] Soga, T., Musa, A., Shimomura, Y., Egawa, R., Itakura, K., Takizawa, H., Okabe, K. and Kobayashi, H.: Performance evaluation of NEC SX-9 using real science and engineering applications, *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp. 1–12 (online), DOI: 10.1145/1654059.1654088 (2009).

[6] Musa, A., Sato, Y., Soga, T., Okabe, K., Egawa, R., Takizawa, H. and Kobayashi, H.: A Shared Cache for a Chip Multi Vector Processor, *Proceedings of the 9th Workshop on MEmory Performance: DEaling with Applications, Systems and Architecture*, MEDEA '08, New York, NY, USA, ACM, pp. 24–29 (online), DOI: 10.1145/1509084.1509088 (2008).

[7] Egawa, R., Funaya, Y., i. Nagaoka, R., Musa, A., Takizawat, H. and Kobayashi, H.: Design and early evaluation of a 3-D die stacked chip multi-vector processor, *2010 IEEE International 3D Systems Integration Conference (3DIC)*, pp. 1–8 (online), DOI: 10.1109/3DIC.2010.5751448 (2010).

[8] Egawa, R., Funaya, Y., Nagaoka, R., Endo, Y., Musa, A., Takizawa, H. and Kobayashi, H.: Effects of 3-D stacked vector cache on energy consumption, *2011 IEEE International 3D Systems Integration Conference (3DIC), 2011 IEEE International*, pp. 1–6 (online), DOI: 10.1109/3DIC.2012.6263026 (2012).

[9] Hill, M. D. and Smith, A. J.: Evaluating associativity in CPU caches, *IEEE Transactions on Computers*, Vol. 38, No. 12, pp. 1612–1630 (online), DOI: 10.1109/12.40842 (1989).

[10] Kharbutli, M., Solihin, Y. and Lee, J.: Eliminating Conflict Misses Using Prime Number-Based Cache Indexing, *IEEE Trans. Comput.*, Vol. 54, No. 5, pp. 573–586 (online), DOI: 10.1109/TC.2005.79 (2005).

[11] Bodin, F. and Seznec, A.: Skewed associativity improves program performance and enhances predictability, *IEEE Transactions on Computers*, Vol. 46, No. 5, pp. 530–544 (online), DOI: 10.1109/12.589219 (1997).

[12] Seznec, A.: A New Case for Skewed-Associativity, Research Report RR-3208, INRIA (1997).

[13] Jaleel, A., Theobald, K. B., Steely, Jr., S. C. and Emer, J.: High Performance Cache Replacement Using Re-reference Interval Prediction (RRIP), *SIGARCH Comput. Archit. News*, Vol. 38, No. 3, pp. 60–71 (online), DOI: 10.1145/1816038.1815971 (2010).

[14] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D. and Wood, D. A.: The Gem5 Simulator, *SIGARCH Comput. Archit. News*, Vol. 39, No. 2, pp. 1–7 (online), DOI: 10.1145/2024716.2024718 (2011).

ⓒ 2018 Information Processing Society of Japan