卒業論文

題目

キャッシュのアクセス解析による キャッシュ領域の割り当て手法の 提案と評価

指導教員

佐々木 敬泰 助教

2018 年

三重大学 工学部 情報工学科
コンピュータアーキテクチャ研究室

田路 徹哉 (412837)

内容梗概

近年，スマートフォンやパソコンなどのプロセッサでは高性能化のために様々な手法が提案されている．それらの高性能化手法の一つとして複数のコアを用いて並列に処理を行うマルチコア化が挙げられる．しかし，マルチコア化では複数のメモリアクセスが同時に発生するため，データの競合が発生しやすくなり，キャッシュミス率が増大する問題点がある．メモリアクセスは性能のボトルネックであるため，高性能化を達成するためにはメモリアクセスの低減，すなわちキャッシュミス率の低減が重要である．キャッシュミス低減手法の一つにキャッシュ・パーティショニングがある．キャッシュ・パーティショニングは各コアの負荷に応じてウェイを動的に割り当てることで，キャッシュ領域を効率的に扱い，全体のキャッシュミスの削減を図っている．しかし，ウェイ単位で割り当てを行うため，あるコアのアクセスがウェイの特定の部分に集中することにより，ウェイの一部しか使用していない場合，他のコアが未使用の領域を使用できない．そこで当研究室では，キャッシュをウェイより細かい単位である「セル」に分割して動的にコアに割り当てるセル・アロケーションキャッシュを提案しているが，キャッシュミス率が高いコアにキャッシュの容量を割り当て過ぎてしまい，その結果，全体のキャッシュミス率の改善を妨げる場合があるという問題がある．そこで，本研究ではキャッシュのアクセス解析を用いて，キャッシュミス率が低いコアに必要なキャッシュの容量を推定することで，キャッシュミス率が低いコアにも十分にキャッシュの容量を割り当てる手法の提案をする．その結果，従来のセル・アロケーションキャッシュと比較して，キャッシュミス率が平均 4.1 %，最大 7.5 %増加したため，さらにその原因を調査し，改善案を提案する．

Abstract

Recently, various methods for improving computial performance have been proposed. One of these various methods is Multi-core. Multi-core can execute processes in parallel by using multiple processors. However, it occurs many memory access simultaneously. As a result, cache miss rate is increased. Moreover, cache miss and memory access are occur at the same time. It is important to reduce memory access, because memory access has interfered with improving the computial performance. One of the methods of reducing cache miss rate is Cache-Partitioning. This method allocates ways to each core on demand. However, because of allocating ways to each cores, in spite of using only specific part of the way, other cores can't use regions which are not used. To solve this problem, our laboratory proposes Cell-Allocation cache. However, Cell-Allocation cache allocates cache capacity to the core which is the worst cache miss rate, hence it may interfere with improving the whole of cache miss rate. To solve this problem,this paper propose the method of allocating cache capacity to the core of lower cache miss rate by using the analysis of the tendency of accessing to cache. As a result, cache miss rate has increased 4.1 % on average and 7.5 % on maximum compared with the conventional Cell-Allocation cache. Therefore, this paper search the causes of increasing cache miss rate and propose the improvement plans.

目 次

図 目 次

表 目 次

1　はじめに

近年，スマートフォンやパソコンなどのプロセッサでは高性能化のために様々な手法が提案されている．それらの高性能化手法の一つとして複数のコアを用いて並列に処理を行うマルチコア化が挙げられる．しかし，マルチコア化では複数のメモリアクセスが同時に発生するため，データの競合が発生しやすくなり，キャッシュミス率が増大する問題点がある．メモリアクセスは性能のボトルネックであるため，高性能化を達成するためにはメモリアクセスの低減，すなわちキャッシュミス率の低減が重要である．キャッシュミス低減手法の一つにキャッシュ・パーティショニング [1] がある．キャッシュ・パーティショニングは各コアの負荷に応じてウェイを動的に割り当てることで，キャッシュ領域を効率的に扱い，全体のキャッシュミス削減を図っている．しかし，ウェイ単位での割り当てを行うため，あるコアのアクセスがウェイの特定の部分に集中することにより，ウェイの一部しか使用していない場合，他のコアが未使用の領域を使用できない．そこで当研究室では，キャッシュをウェイより細かい単位である「セル」に分割して動的にコアに割り当てるセル・アロケーションキャッシュ[2] が提案されている．この手法により，キャッシュ・パーティショニングの問題点を解決し，キャッシュミス率を低減することには成功

したが，キャッシュミス率が高いコアにキャッシュの容量を割り当て過ぎてしまい，その結果，全体のキャッシュミス率の改善を妨げる場合があるという問題がある．そこで，本研究ではキャッシュのアクセス解析を用いることにより，キャッシュミス率が低いコアに必要なキャッシュの容量を推定することで，キャッシュミス率が低いコアにも十分にキャッシュの容量を割り当てる手法の提案をする．

2　キャッシュの概要と関連研究

2.1　キャッシュ

本章では，提案手法の説明に先立ち，まず一般的なキャッシュシステムについて概括する．

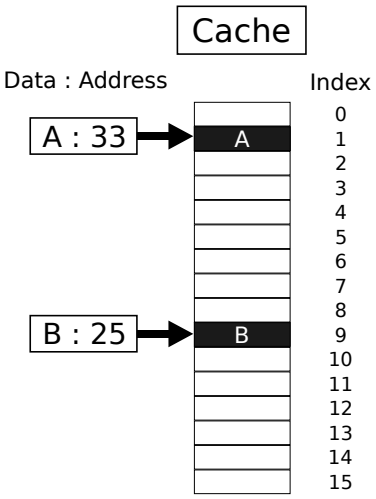

図 2.1: キャッシュへのアクセス [2]

キャッシュとはコアと主記憶の間にある，小容量で高速にアクセス可能なメモリのことである．キャッシュには,「インデックス」と呼ばれるキャッシュの番地があり，インデックスにあるデータの格納領域,「エントリ」がある．インデックスにアクセスした際に，目的のデータが格納されていた場合を「ヒット」，目的のデータが格納されていない場合は「ミス」と一般的に定義されている．キャッシュミスが発生する場合，主記憶からデータを読み込む．コアとキャッシュの間のアクセスは高速であ

るが，キャッシュと主記憶の間のアクセスは低速である．性能向上においては，キャッシュと主記憶の間のアクセスはボトルネックであるため，これを低減する必要がある．

アクセス先，即ちインデックスは，主記憶のアドレスとキャッシュのインデックス数の剰余を取ることで求められる．具体的にはインデックスを index，アドレスを address，インデックス数を set，剰余演算を mod とすると，下記の式 (1) で求まる．

$$\text{index} = \text{address} \bmod \text{set} \tag{1}$$

例えば図 2.1 のインデックス数が 16 のキャッシュの場合を考える．データ A のアドレスが 33 の場合，式を用いると $33 \bmod 16 = 1$ と求められる．同様にデータ B のアドレスが 25 の場合，$25 \bmod 16 = 9$ と求められる．

キャッシュはデータを格納する場合，アクセスしたインデックスのエントリに空きがなければ，格納しているデータを追い出すよう設計されている．式 (1) により，インデックスは一意に決まるが，異なるアドレスから同じインデックスが求められ，データの競合が発生する場合がある．図 2.1 のような各インデックスのエントリ数が 1 のキャッシュは，競合する場合が多く，キャッシュミスが発生しやすい．追い出したデータの必要性が大きく再使用の頻度が高ければ，さらにキャッシュミスが発生する．

そのため，一般的なデータの格納方式として，同一のインデックスに複数のエントリを配置する，セットアソシアティブ方式が使用されている．

2.2 セットアソシアティブキャッシュ

セットアソシアティブ方式では，同一のインデックスに複数のエントリを配置することにより，同一のインデックスに複数のデータを格納することができる．このエントリ群を「ウェイ」と呼び，この方式を用いたキャッシュをセットアソシアティブキャッシュと呼ぶ．セットアソシアティブキャッシュの概念図を図 2.2 に示す．

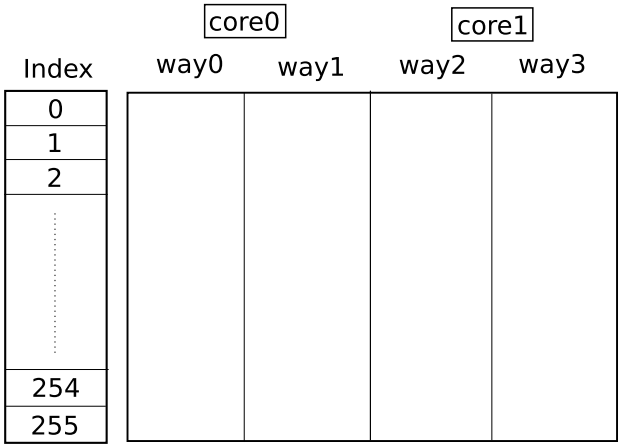

図 2.2: セットアソシアティブキャッシュの概要 [3]

このキャッシュを用いることにより，図 2.1 のキャッシュでは必要なデータが追い出されることが頻発していたが，エントリが複数あるため，必

要なデータを多く格納することができる．データを追い出す場合は，アクセスしたインデックス内の最も古いエントリのデータを追い出す．

しかし，マルチコア環境下ではコアが増えるためアクセス数が増加する．異なるコア同士が同じインデックスにアクセスする場合もあり，データの競合が発生しやすいという問題がある．

2.3 キャッシュ・パーティショニング

キャッシュの高性能化手法の一つとしてキャッシュ・パーティショニング [1] がある．この手法は，コアにウェイを動的に割り当て，各コアの負荷に応じてウェイ数を調節し，各コアに最適なキャッシュ領域を確保する手法である．例として，2 コアでのキャッシュ・パーティショニングによる割り当てを図 2.3 と図 2.4 に示す．

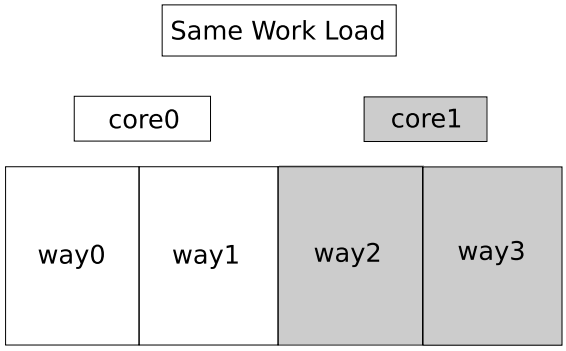

図 2.3: 同負荷における割り当て [3]

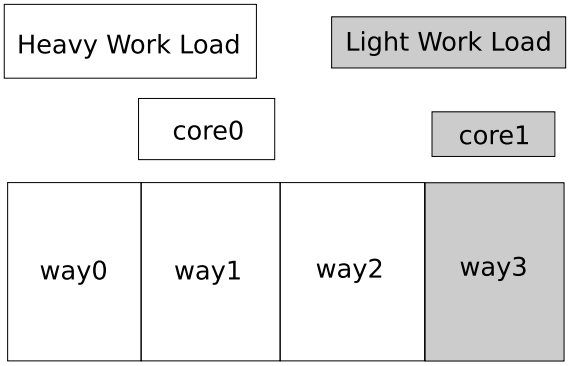

図 2.4: 異なる負荷における割り当て [3]

キャッシュ・パーティショニングでは負荷が同程度の場合，図 2.3 のように均等に各コアにウェイを割り当てる．また，負荷が異なる場合，図 2.4 のように負荷が大きい方にウェイを割り当てる．このように各コアに最適なキャッシュ領域を動的に割り当てることで，キャッシュ領域を効率的に扱い，全体のキャッシュミスの削減を図っている．

しかし，キャッシュ・パーティショニングでは，ウェイ単位でキャッシュを割り当てるが，ウェイ単位では分配粒度が粗く，あるコアのアクセスがウェイの特定部分に集中し，そのウェイの一部分しか使用していないのにも関わらず，他のコアが空いている領域を利用できない問題がある．

3　セル・アロケーションキャッシュ

3.1　セル・アロケーションキャッシュの概要

前章の問題を解決するために，当研究室ではセル･アロケーションキャッシュを提案している．セル・アロケーションキャッシュは，一個のウェイをさらに細分化領域である「セル」を単位として各コアにキャッシュを割り当てる手法である．セル・アロケーションキャッシュの概念図を図 3.5 に示す.

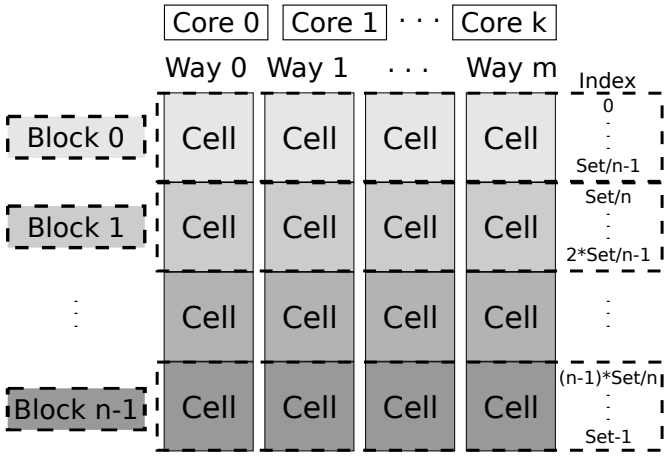

図 3.5: セル・アロケーションキャッシュの概念図 [2]

セルとは複数のエントリをまとめたグループのことである．また，破線のように, ウェイ方向に並んだセルのグループをブロックとする. セルには特定のコアのみがラインリプレースできる固有セルと，通常キャッシュと同様にアクセスできる共有セルの 2 種類があり，キャッシュミス率とデータ共有率を用いて各コアにセルを割り当てる．後章の第 4 章で記

述する提案手法では，キャッシュの容量の推定の際に各コアに割り当てるキャッシュの容量を変化させる．しかし，共有セルを用いる場合，共有セルはどのコアからでもアクセスが可能なため，実際に各コアが使用出来るキャッシュの容量が割り当て時より増えてしまい，各コアに必要なキャッシュの容量を求めるのを妨げる．そのため，本研究では共有セルを用いないため，共有セルを使用した割り当て手法は記載しない．

また，セル・アロケーションキャッシュのキャッシュの割り当てにはアクセス履歴と疑似的ウェイ拡張を用いるセルの割り当てもある．しかし，アクセス履歴を用いた場合，固有セルを共有セルに変化させる処理が含まれるためこれを用いない．また，第 4.1 節の調査では，インデックスを求める際に第 2.1 節の式 (1) を用いるが，第 2.1 節の式 (1) により求められるインデックスとは異なるインデックスにデータを格納するのを可能にする疑似的ウェイ拡張を用いた場合は，格納先の変更によりアクセスパターンが変化するため，調査により判明したキャッシュミス率の傾向とは異なる可能性がある．加えて，キャッシュミス率の傾向のパターンが増えるほど，各コアに割り当てるキャッシュの容量の推定に必要な情報が多くなり，プログラムでの実装に掛かる時間が大きくなる．そのため，本研究の時間の都合上，これも用いない．従って，どちらも本研究では用

いないため記載しない．

ここで，各コアのセルの割り当てについて説明する．各コアのセルの割り当ては，キャッシュミス率が最も高いコアはキャッシュの容量を必要とし，キャッシュミス率が最も低いコアはキャッシュの容量を必要としない．

最初に各コアのキャッシュミス率を比較し，キャッシュミス率が最も高いコアと最も低いコアを求める．次に，ブロック毎で最もキャッシュミス率の高いコアが有しているセルを対象に，キャッシュミス率が最も大きいブロックと二番目に大きいブロックを求める．キャッシュミス率が最も大きいブロックに最もキャッシュミス率が低いコアのセルがあれば，その内のセルで，最もアクセスの古いセルの割り当て先をキャッシュミス率が最も高いコアに変更する．なければ，二番目のブロックでも同様に行う．それでもなければ，最もキャッシュミス率が低いコアに割り当てられているセルの内で，最もアクセスの古いセルの割り当て先をキャッシュミス率が最も高いコアに変更する．キャッシュミス率が大きいブロックはアクセスが集中しているため，この割り当てを用いて，アクセスが集中している部分にキャッシュを割り当てる．また，この割り当てを一定間隔で行うことで，動的に各コアが必要とするセルを割り当てることができる．

このように，セル・アロケーションキャッシュは，キャッシュミス率が

高いコアに優先的にセルを割り当て，キャッシュミス率が低いコアにはあまり割り当てないことにより，各コアの負荷に応じたセルの割り当てを行うことができる．

また，セル・アロケーションキャッシュと通常キャッシュのセルの有無以外の違いは，キャッシュミス時のデータの格納先を決める場合に，各コアの固有セル内のエントリに優先して書き込む点である．もし，書き込み先のインデックスにアクセスしたコアの固有セルがなければ，通常キャッシュと同様に他のエントリに空きがあればそこに書き込み，空きがなければエントリの中からデータを追い出す．

3.2 問題点

従来のセル・アロケーションキャッシュでは，キャッシュミス率が高いコアに優先的にセルを割り当て過ぎる問題がある．これは，セル割り当てアルゴリズムによりキャッシュミス率が高いコアに優先的にセルを割り当てているが，キャッシュミス率が改善していないのにセルを割り当てると全体のキャッシュミス率改善を妨げる場合がある．本来なら十分にキャッシュの容量がコアに割り当てられている場合，キャッシュミス率が改善するコアはあったが，従来のセル割り当てアルゴリズムによりキャッシュミ

ス率が高いコアに割り当て過ぎると，全体のキャッシュ容量は限られているため，キャッシュミス率が改善するコアに十分な容量を割り当てられなくなり，キャッシュミス率が改善しなくなる．従って，キャッシュミス率が高いコアだけでなく，キャッシュミス率が改善できそうなコアにも割り当てることで，キャッシュミス率を改善できると考えられる．

4　キャッシュのアクセス解析によるキャッシュ領域の割り当て

4.1　キャッシュのアクセス解析

従来のセル・アロケーションキャッシュの問題を解決するため，本研究ではキャッシュの容量毎のキャッシュミス率の解析から各コア毎に必要なキャッシュの容量を推測し，セル・アロケーションキャッシュの改善を目指す．そこで，様々なベンチマークプログラムを用いて，キャッシュの容量を変化させた場合のキャッシュミス率の傾向を調査する．これはプログラムのキャッシュへのアクセスパターンにはある程度の規則性があり，その規則性が分かれば各コア毎に必要なキャッシュの容量が推測できると考えたためである．その結果として，幾つかのパターンが分かった．キャッシュの容量毎のキャッシュヒット率の推移を図 4.6 に示す．視認性を向上させるため，キャッシュミス率ではなくキャッシュヒット率で示す．

図 4.6 のようにキャッシュの容量毎のキャッシュヒット率には様々な傾向が見られる．その一つとして，キャッシュの容量を与えるとキャッシュヒット率は改善するが，ある一定のキャッシュの容量以上を与えると改善しづらい傾向を示すものがある．また，キャッシュの容量を増やしてもキャッシュヒット率が低いまま変化しない場合もある．従って，キャッシュ

の容量を増やしてもキャッシュヒット率が変化しない傾向が現れれば，ある一定のキャッシュの容量以上を割り当てるのは非効率的である．しかし，キャッシュの容量を増やせばキャッシュヒット率が改善し続ける傾向もあり，必要なキャッシュの容量が分からないものもある．そのため，キャッシュの容量を全て割り当てても，キャッシュヒット率が十分に改善しないことがあるため，このような傾向を持つコアにキャッシュの容量を優先的に割り当てるのも非効率的である．

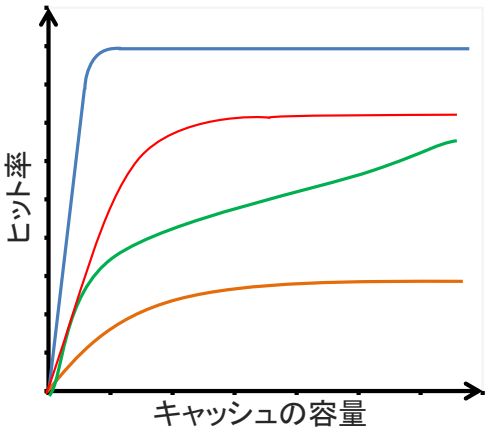

図 4.6: ヒット率の傾向

マルチコア環境では各コアにこれらの傾向が現れ，キャッシュの容量を奪い合っている．そのため，従来のセル・アロケーションキャッシュの割り当てに従って，キャッシュの容量を増やしてもキャッシュヒット率が低いまま変化しない傾向を持つコアにキャッシュの容量を割り当てると，一

定のキャッシュの容量を与えることでキャッシュヒット率が十分に改善しそうな傾向を持つコアにキャッシュの容量を割り当てられなくなるため，全体のキャッシュミス率が改善しなくなることが裏付ける．

4.2 キャッシュ領域の割り当て

前節の調査結果を基に，キャッシュミス率が低いコアに対して，キャッシュミス率が改善しづらくなるキャッシュの容量を推測し，残りをキャッシュミス率が高いコアに与える手法を提案する．提案手法により，実際にターゲットプログラムを実行する場合は，以下の手順で最適なキャッシュ容量の割り当てを行う．まず，一定間隔後にキャッシュミス率が高いコアと低いコアのグループ分けを行う．これは，プログラムを実行直後にキャッシュミス率を測定しても，プログラムの実行直後は初期化を行うため，キャッシュへのアクセスパターンが安定しないためである．グループ分けは，4 コアの場合，キャッシュミス率が一番高いコアと一番低いコアを 1 グループとし，残りをもう 1 グループとする．4 コアの場合，各グループに割り当てられているキャッシュはキャッシュ全体の容量の半分ずつとなり，各グループに割り当てられているキャッシュをグループ内のコアで割り当てる．次にグループ内のキャッシュミス率が低いコアに対し，

一定期間毎にセルの割り当てを変化させることで容量を変更し，キャッシュミス率の変化を計測する．キャッシュミス率が高いコアも同様にコアに割り当てられている容量やキャッシュミス率は変化するが，この変化は計測しない．このようにして得られた結果と事前に取得した傾向パターンから最適なキャッシュの容量を予測する．これは，キャッシュミス率が低いコアはキャッシュミス率が改善しづらい傾向が現れやすく，必要な容量の推測が容易であることに基づいているためである．また，キャッシュミス率が低いコアは容量を多く必要としないことが多いため，キャッシュミス率が低いコアがいずれかの傾向を示しても，対応できると考えたためである．予測が完了し，キャッシュミス率が低いコアに十分な容量を割り当てることができれば，各グループ内のキャッシュミス率が高いコアに残りを割り当てる．

4.3 必要なキャッシュ容量の予測

キャッシュの容量の予測を行うアルゴリズムを図 4.7 に示す.

```
main()
If(Current Cycle > Interval){
  assign Middle.MissRate;
  if(Decided = False){
    decide();
    allocate;
  }
}

decide()
foreach LowerMissRateCore{
  if(Middle.MissRate ≒ Upper.MissRate)
    Upper = Middle;
  else if(Middle.MissRate > Upper.MissRate)
    Lower = Middle;
  Middle.Capacity = (Upper.Capacity + Lower.Capacity) / 2;
  set allocate_value;
  judge();
}

judge()
if(Upper.Capacity - Lower.Capacity = 1Cell){
  Decided = True;
  if(Middle.MissRate > Upper.MissRate){
    Middle = Upper;
    set allocate_value;
  }
}
```

図 4.7: 擬似関数によるアルゴリズム

アルゴリズムは基本処理を行う関数 main，キャッシュミス率を比較して，結果に応じてキャッシュの容量の変更をする関数 decide，キャッシュの

容量が決まったかを判定する関数 judge の 3 つによって成り立つ．キャッシュの容量が増えにくくなる容量の予測は，グループ化した際のキャッシュの容量を最大とし，キャッシュの容量の最小が 0 とする範囲内で以下の手順により行う．

まず，図 4.7 の関数 main が実行され，範囲内の中間値のキャッシュの容量のキャッシュミス率の測定を行う．次に，図 4.7 の関数 decide が実行され，中間値の容量のキャッシュミス率を範囲の最大値のキャッシュミス率と比較し，結果に応じてキャッシュの容量を変更するために必要な情報を求める．中間値の容量のキャッシュミス率が最大値のキャッシュミス率より高ければ，キャッシュの容量を増やすことでキャッシュミス率が改善する傾向にあると考えられる．そのため，現在割り当てられているキャッシュの容量より多く必要なので，次回の割り当てはキャッシュの容量を増やす．この場合，現在割り当てられているキャッシュの容量より少なくなることはないため，最小値を現在割り当てられているキャッシュの容量に更新する．また，キャッシュミス率がほぼ等しければ，キャッシュミス率が改善しにくい傾向にあると考えられる．そのため，キャッシュの容量は余分であるので，次回の割り当てはキャッシュの容量を減らす．この場合，現在割り当てられているキャッシュの容量より多くなることはないため，

最大値を現在割り当てられているキャッシュの容量に更新する．そして，キャッシュの容量の変更，すなわち，各コアに割り当てるセルの個数の変更に必要な情報を求める．必要な情報は，セルの割り当てをし直すセルの個数の情報であり，実際のセルの割り当てにはこの情報を基に行う．セルの割り当てをし直すセルの個数は，最小値と最大値のいずれかが変更することにより，中間のキャッシュの容量が変化するので，変化前と変化後の差分をとることで求められる．最後に，図 4.7 の関数 judge を実行し，最大値と最小値を確認し範囲の変更がこれ以上できなくなるかを判断する．範囲の変更を行えない場合で，キャッシュの容量を増やすと判断されていた場合，割り当てるキャッシュの容量を最大値に変更する．このような流れを必要なキャッシュの容量が求まるまで一定間隔毎に繰り返すことで，キャッシュミス率が低いコアに必要なキャッシュの容量が求められる．

もし，最大値が変化しなかった場合で，最大値を超えて割り当てると，キャッシュミス率が高いコアがキャッシュミス率が改善し続ける傾向にある場合，そのコアのキャッシュミス率の改善を妨げてしまう．反対に改善しにくい傾向にある場合は，改善しにくくなる傾向が現れるまで十分にキャッシュの容量を与えられず，キャッシュミス率が十分に改善しなくなってしまう場合もある．そのため，最大値が変化しなかった場合はグルー

プ内で均等に割り当てる．

また，容量を変更する際のセルの割り当ては以下のように行う．キャッシュの容量を増やす場合は，グループ内のキャッシュミス率が高いコアが所持するセルの割り当て先を変更したい容量分キャッシュミス率が低いコアに変更する．キャッシュの容量を減らす場合は，反対に，グループ内のキャッシュミス率が低いコアが所持するセルの割り当て先を変更したい容量分，キャッシュミス率が高いコアに変更する．割り当て先を変更するセルの選定は，従来のセル・アロケーションキャッシュの割り当てと同様にLRU 方式を用いて選定する．

このようにして，キャッシュミス率が高いコアにキャッシュの容量を割り当て過ぎず，キャッシュミス率が低いコアに対しても十分にキャッシュの容量の確保ができる．

5　性能評価

5.1　評価環境

改良したセル・アロケーションキャッシュをトレースドリブン型キャッシュシミュレーターに実装し，シミュレータ上でベンチマークを実行し評価を行う．評価項目は高性能を目標とするため，キャッシュミス率とし，比較対象は，従来のセル・アロケーションキャッシュとする．ベンチマークには先行研究と同様の，姫野ベンチマーク [4] と Splash2 ベンチマーク [5] の中からワーキングセットの異なる 2 種類を選んで組み合わせたものを用いる．その他の環境は表 5.1 に示す．

表 5.1: 評価環境

コア数	4 コア
キャッシュ容量	256KB × 8 ウェイ

5.2　評価結果

図 5.8 に評価結果を示す．

従来のセル・アロケーションキャッシュと比較して，キャッシュミス率が平均 4.1 %，最大 7.5 %増加した．

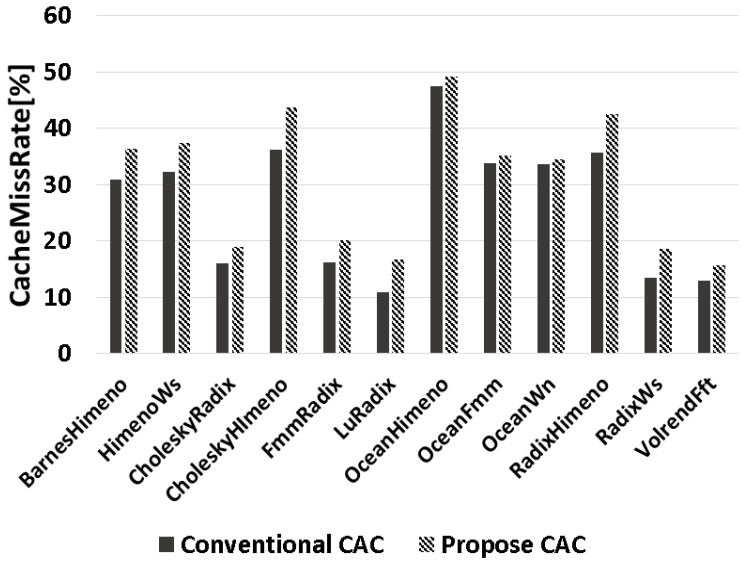
60
50
40
30
20
10
0
CacheMissRate[%]
BarnesHimeno
HimenoWs
CholeskyRadix
CholeskyHImeno
FmmRadix
LuRadix
OceanHimeno
OceanFmm
OceanWn
RadixHimeno
RadixWs
VolrendFft
Conventional CAC
Propose CAC

図 5.8: 評価結果

5.3　考察

前節の図 5.8 より，提案手法は全てのベンチマークにおいてキャッシュミス率が増加した．そこで，以下にその原因を考察する．各コアのキャッシュの容量，キャッシュミス率，アクセス数の推移を見ると，キャッシュミス率が低いコアではアクセス数が少なくなる場合が多く，逆にキャッシュミス率が高いコアではアクセス数が多いため，全体のキャッシュミス率の低減に繋がらなかったためだと考えられる．

例として，キャッシュアクセスの内訳を図 5.9 を示す．

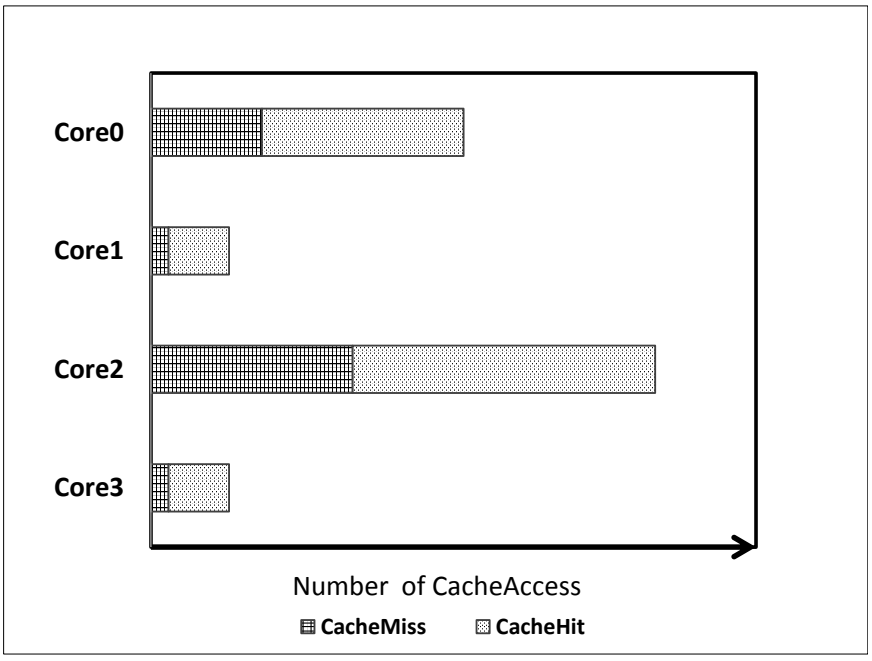

図 5.9: キャッシュアクセスの内訳

図 5.9 により，アクセス数が多い場合，キャッシュミス率が改善すればヒット数もその分増え，反対に，アクセス数が少ない場合，キャッシュミス率が改善してもヒット数はあまり増えないのが分かる．また，提案手法により各コアをグループ化する際に，コア 0 のキャッシュミス率が 30 %，コア 1 のキャッシュミス率が 3 %，コア 2 のキャッシュミス率が 33 %，コア 3 のキャッシュミス率が 3 %であるなら，キャッシュミス率が低いコア 1 とコア 3 に対してキャッシュの容量の操作を行う．しかし，アクセス数とキャッシュミス数を確認すると，コア 0 のキャッシュミス数が 30000 でアクセスが 230000，コア 1 のキャッシュミス数が 10 でアクセスが 300，コア 2 のキャッシュミス数が 300000 でアクセスが 900000，コア 3 のキャッシュミス数が 10 でアクセスが 300 であるのが分かった．そのため，全体のキャッシュミス率を計算すると 29 %となり，全体のキャッシュミス率の改善の効果が小さい．

従って，各コアに必要なキャッシュの容量を求める際に，キャッシュミス率だけでなくアクセス数を考慮することで，改善できると考えられる．

また，今回用いたベンチマークでは，キャッシュの傾向の変化が想定よりも大きく，各コアに必要なキャッシュの容量が変化し続けるため，今回の手法で各コアに対して定めたキャッシュの容量では対応できなかったと考えられる．

例として，時間の経過によるキャッシュミス率の推移を図 5.10 を示す．

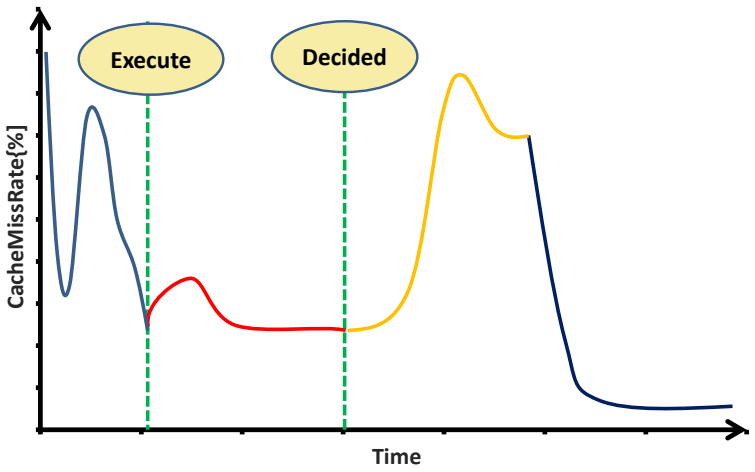

図 5.10: 時間経過でのキャッシュミス率の推移

プログラムの実行直後では変数の初期化を行い，キャッシュへのアクセスの傾向が定まらないため，図 5.10 の左端部分のようにキャッシュミス率が不安定である．そのため，提案手法は初期化後に実行される．各コアにキャッシュの容量を割り当てると，図 5.10 の点線に囲まれた部分の

ようにキャッシュミス率が安定する．しかし，キャッシュへのアクセスの傾向は時間と共に変化し，各コアのキャッシュミス率が大きく変化していることが判明した．キャッシュミス率が提案手法を実行した時より大きければ，割り当てられているキャッシュの容量では不足し，反対に，キャッシュミス率が提案手法を実行した時より小さければ，割り当てられているキャッシュの容量では余分になると考えられる．

このように，キャッシュへのアクセスの傾向が変化してからの各コアのキャッシュミス率と，提案手法を実行した時の各コアのキャッシュミス率に差異が生じている．そのため，キャッシュの容量を確保するべきコアも変化してしまう。

従って，キャッシュの傾向が変化した際に提案手法を再度実行することで，改善できると考えられる．反対に，キャッシュの傾向が変化しない場合は，各コアのキャッシュミス率はほとんど変化しないため，提案手法で効果がでると考えられる．

6　おわりに

スマートフォンやパソコンなどのプロセッサでは高性能化のために様々な手法が提案されている．それらの高性能化手法の一つとして複数のコアを用いて並列に処理を行うマルチコア化が挙げられる．しかし，マルチコア化では複数のアクセスが同時に発生するため，データの競合が発生しやすくなり，キャッシュミス率が増大する問題点がある．メモリアクセスは性能のボトルネックであるため，高性能化を達成するためにはメモリアクセスの低減，すなわちキャッシュミス率の低減が重要である．そこで，当研究室では，キャッシュをウェイより細かい単位である「セル」に分割して動的にコアに割り当てるセル・アロケーションキャッシュが提案されている．しかし，キャッシュミス率が高いコアにキャッシュの容量を割り当て過ぎてしまい，その結果，全体のキャッシュミス率改善を妨げる場合があるという問題がある．そこで，本研究ではキャッシュのアクセス解析を用いることにより，キャッシュミス率が低いコアに必要なキャッシュの容量を推定し，キャッシュミス率が低いコアにも十分にキャッシュの容量を割り当てる手法の提案・評価した．その結果，従来のセル・アロケーションキャッシュと比較して，キャッシュミス率が平均 4.1 %，最大 7.5 %増加したため，その原因を調査し，更に改善案を提案した．今後の

展望として，キャッシュミス率だけでなくキャッシュのアクセス回数も考慮し，変化し続ける傾向に対応できる手法の提案が挙げられる．

謝辞

本研究に関して，ご指導，ご助言をいただいた近藤利夫教授，佐々木敬泰助教授，深澤祐樹研究員に感謝いたします．また，様々な局面でご助力頂いた修士 1 年の鬼頭優人先輩をはじめとするコンピュータアーキテクチャ研究室の皆様にも感謝いたします．

参考文献

[1] G. E. Sue, L. Rudolph, and S. Devadas, “Dynamic Partitioning of Shared Cache Memory,” Journal of Supercomputing, vol.28, No.1, pp.7-26, January 2004.

[2] 刀根 舞歌，“高効率な分割領域動的管理キャッシュの研究，”March，2017．

[3] 吉田 康平，“キャッシュ領域の割り当てアルゴリズムの改良と評価，”March， 2017．

[4] 姫野龍太郎，姫野ベンチマーク，<http://accc.riken.jp/supercom/himenobmt/> (2017 年 2 月 20 日アクセス)．

[5] The Modified SPLASH-2，<http://www.capsl.udel.edu/splash/>(2017年3月2日アクセス).