卒業論文

題目

キャッシュ領域の割り当てアルゴリズムの改良と評価

指導教員

佐々木 敬泰 助教

2017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

吉田 康平 (412863)

内容梗概

近年，パソコンやスマートフォンの普及に伴い，それらに用いられているプロセッサでは，高性能化が求められている．高性能化の手法の一つとして，複数のコアを用意し並列に処理を行うマルチコア化が挙げられる．しかし，マルチコア環境では複数のアクセスが同時に発生し，シングルコア環境よりデータの競合が発生しやすくなる．そのため，ミス率が高くなり実効性能がかえって悪化する恐れがある．メモリアクセスの遅さは性能のボトルネックの一つであるため，これを回避するためにキャッシュの性能向上が必要である．キャッシュの高性能化手法には，キャッシュ・パーティショニング [1] という手法があるが，この手法は必ずしも最適なキャッシュ領域を割り当てているとは言えない．この問題を解決する手法として当研究グループでは，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．しかし，セル・アロケーションキャッシュは，キャッシュミス時に主記憶からキャッシュへ書き込む際に，使用頻度が高いデータを書き換えてしまう場合があり，通常キャッシュを用いた場合よりもキャッシュミスが増加する場合がある．そこで本研究では，2 種類の置換方式を状況に応じて切り換えることにより，キャッシュミス率の低減を目指す．置換方式には，従来から使用している，使用されてから最も長い時間が経過したデータを最初に捨てる LRU(Least Recently Used) 方式と，最後に使用されてから経過した時間と使用された頻度の両方を考慮する ARC(Adaptive Replacement Cache) 方式を用いる．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，ミス率を最大 4.9%，平均 3.2%低減できた．

Abstract

Multi-core processor is widely used to achieve high-performance. Multi-core processor can execute parallel process by mounting multiple cores in a processor. However, memory access frequency of multi-core processor is larger than that of single-core processor. Therefore, cache misses for multi-core processor may increase and degrade system performance. As a study to improve cache, Cell-Allocation Cache that is fine-grain dynamic assigning of cache region on each core is proposed. However, Cell-Allocation Cache has a problem that cache misses increase than normal cache. This study improves the algorithm of Cell-Allocation Cache to control cache spaces dynamically. Compared with a conventional Cell-Allocation Cache, the improved method reduces 3.2% of cache miss rate on average.

目 次

1 はじめに **1**
1.1 研究背景 1
1.2 研究目的 2

2 従来研究とその問題点 **4**
2.1 セットアソシアティブキャッシュ 4
2.2 キャッシュ・パーティショニング 5
2.3 従来研究の問題点 7

3 セル・アロケーションキャッシュ **10**
3.1 セル・アロケーションキャッシュの概要 10
3.2 セル・アロケーションキャッシュの問題点 11

4 キャッシュ領域割り当てアルゴリズムの改良 **13**
4.1 提案手法の概要 13
4.2 提案手法のアルゴリズム 14
4.2.1 アクセス時間と使用頻度の記録 14
4.2.2 割り当て方式の最適化 15

5 性能評価 **16**
5.1 評価方法 16
5.2 評価結果 17
5.3 考察 19

6 おわりに **20**

謝辞 **22**

参考文献 **23**

図 目 次

2.1 セットアソシアティブキャッシュの概要 4
2.2 同負荷における割り当て 6
2.3 異なる負荷における割り当て 6
2.4 ウェイ割り当てによる割り当て図 8
2.5 理想的な割り当て図 8
3.6 セル・アロケーションキャッシュの概要 10
4.7 提案手法の概要 14
4.8 改良アルゴリズムでの割り当て 16
5.9 通常キャッシュとのミス率（2 コア） 18
5.10 通常キャッシュとのミス率（4 コア） 18

表 目 次

5.1 評価環境 17

1　はじめに

1.1　研究背景

近年，パソコンやスマートフォンなどの普及に伴い，それらに用いられているプロセッサでは高性能化が求められており，様々な高性能化手法が提案されている．高性能化の手法の一つとして，複数のコアを用意し並列に処理を行うマルチコア化が挙げられる．しかし，マルチコア環境では複数のメモリアクセスが同時に発生することで，シングルコア環境よりデータの競合が発生しやすくなり，ミス率が高くなってしまう．メモリアクセスは性能のボトルネックの一つであるため，これを回避するためにキャッシュの性能向上が必要である．キャッシュの高性能化手法には，キャッシュ・パーティショニング [1] という手法がある．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割り当て，性能向上をはかる手法である．しかし，ウェイ単位で割り当てを行うため分配粒度が粗く，あるコアのアクセスがウェイの特定の部分に集中すると，ウェイの一部分しか使用していないにもかかわらず，そのウェイを独占するため，他のコアが未使用の領域を使用できない．そのため，必ずしも最適なキャッシュ領域を割り当てているとは言えない．この問題を解決する手法として，当研究グループではキャッシュをより細かい単位である「セ

ル」に分割し，コアに割り当て効率的に扱うセル・アロケーションキャッシュ[2] を提案している．

1.2 研究目的

セル・アロケーションキャッシュは，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手法である．これにより，キャッシュ・パーティショニングの問題点である，ウェイ単位での割り当ての分配粒度の粗さを解決できる．しかし，セル・アロケーションキャッシュでは，通常キャッシュを用いた場合よりもキャッシュミスが増加する場合がある．これはキャッシュミスが発生した場合に，主記憶からデータを取得し，キャッシュへ書き込む際に，使用頻度が高いデータを書き換えてしまう場合があるからである．したがって，キャッシュに格納できるデータ量が減少し，キャッシュミスがデータ競合の削減回数よりも増加すると考えられる．そこで本研究では，2 種類の置換方式を状況に応じて切り換えることにより，キャッシュミス率の低減を目指す．置換方式には，従来から使用している，使用されてから最も長い時間が経過したデータを最初に捨てる LRU(Least Recently Used) 方式と，最後に使用されてから経過した時間と使用された頻度の両方を考慮する ARC(Adaptive

Replacement Cache) 方式を用いる．2 種類の置換方式を切り換えるように改良したセル・アロケーションキャッシュを，従来のセル・アロケーションキャッシュと比較した結果，ミス率を 2 コアで最大 4.9%，平均 3.2%，4 コアで最大 3.7%，平均 2.1%低減できた．

2　従来研究とその問題点

2.1　セットアソシアティブキャッシュ

従来より，シングルコア用キャッシュシステムにおいても，データの競合を避けるためセットアソシアティブキャッシュが用いられている．セットアソシアティブキャッシュとは，メインメモリとキャッシュの間でデータの格納位置の対応付けを行う方式の一つである．図 2.1 にセットアソシアティブキャッシュの概要を示す．

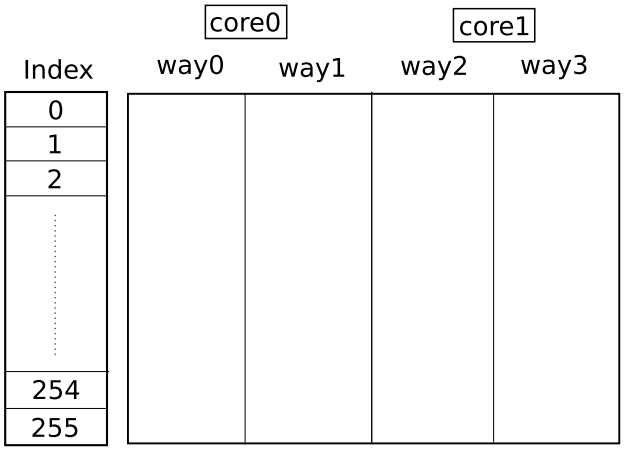

図 2.1: セットアソシアティブキャッシュの概要

セットアソシアティブキャッシュでは，図 2.1 のように複数個のメモリを並列に配置し，ウェイ数を増して連想度を上げる．単純なキャッシュでは，メインメモリのアドレスに剰余のような一定の計算を行ってインデックスを算出し，その位置にあるいくつかのウェイのいずれかにデータを

格納する．具体的には，アドレスを A，キャッシュを N ブロックとすると，データの格納先の番地，すなわちインデックスは $A \bmod N$ という式で求められる．ただし，アドレスによってはインデックスが同一になる．その場合，古いデータを追い出し，新しいデータを格納する．もしデータ格納時のアドレスが N の倍数に偏っていた場合，同一のブロックに割り当てが集中し，性能が著しく低下する．そこで，セットアソシアティブキャッシュはウェイを複数用意して，同一のインデックスにデータを同時に複数格納できるようにする．これにより，データの競合が減少し，メモリアクセスの削減につながる．しかし，セットアソシアティブキャッシュでは，マルチコア環境において各コアが異なるアドレス空間にアクセスするため，メモリアクセスが増加し，データの競合が発生しやすいという問題点がある．

2.2 キャッシュ・パーティショニング

セットアソシアティブキャッシュの問題点を解決する手法の一つとして，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ことで競合を削減する

手法である．図 2.2，図 2.3 にキャッシュ・パーティショニングを用いた場合における，各コアへのキャッシュ容量の割り当て図を示す．

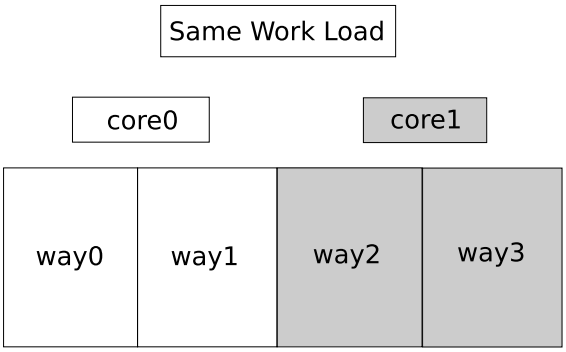

図 2.2: 同負荷における割り当て

例として，各コアの負荷が同程度の場合は，同程度のキャッシュの容量を求めていると考えられるため，図 2.2 のように同程度のウェイ数が割り当てられる．

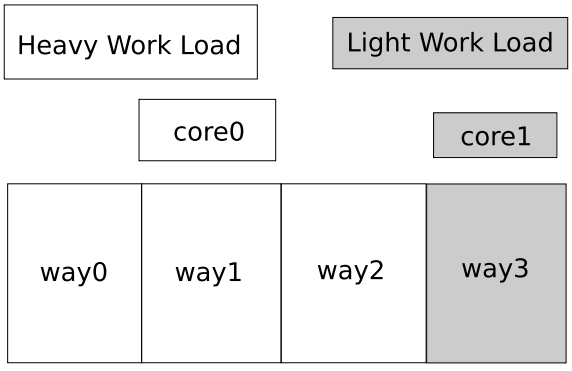

図 2.3: 異なる負荷における割り当て

一方，各コアの負荷が大きく異なる場合，例えばコア 0 の負荷が大き

く，コア 1 の負荷が小さい場合には，コア 0 がコア 1 よりキャッシュの容量を多く求めていると考えられるため，図 2.3 のようにコア 0 の方へウェイ数を多く割り当てる．このように，各コアに適切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える．これにより，キャッシュの未使用領域を削減し，全体のキャッシュミスを削減できる．

2.3 従来研究の問題点

キャッシュ・パーティショニングでは，キャッシュ領域をウェイ単位で割り当てるが，ウェイ単位での割り当てでは分配粒度が粗いという問題がある．例えば，あるコアのアクセスがウェイの特定の部分に集中すると，ウェイの一部分しか使用していないにもかかわらず，そのウェイを独占するため他のコアが未使用の領域を使用できない．そのため，必ずしも最適なキャッシュ領域の割り当てができているとは言えない．図 2.4 にキャッシュ・パーティショニングによる割り当てを，図 2.5 に理想的な割り当てを示す．

例として，図 2.4 のキャッシュ構成を考える．この場合に，コア 0 がキャッシュメモリの上部に集中しており，コア 1 はそれ以外の領域でアクセスが集中していると仮定する．各コアの割り当ては図 2.5 が望ましいが，キャッ

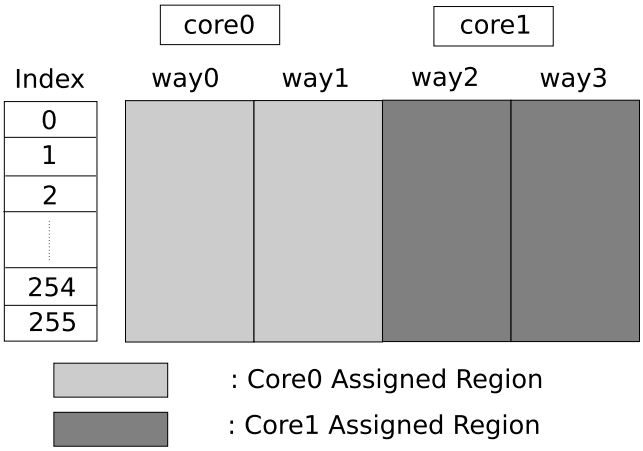

図 2.4: ウェイ割り当てによる割り当て図

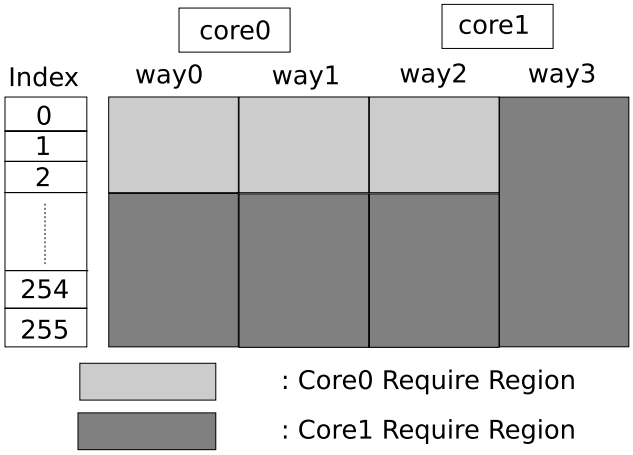

図 2.5: 理想的な割り当て図

シュ・パーティショニングはウェイ単位での割り当てを行うため，全体の負荷が同程度の場合，実際に可能な割り当ては図 2.4 となる．各コアがアクセスの集中するインデックスを最大限使えないため，これは最適な割り当てではない．

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

前章で述べた問題点を解決するため，当研究グループではセル・アロケーションキャッシュという手法を提案している．セル・アロケーションキャッシュは，キャッシュ・パーティショニングを応用した手法で，1 個のウェイを複数の細分化した領域である「セル」を単位としてコアに割り当てる手法である．図 3.6 にセル・アロケーションキャッシュの概要図を示す．

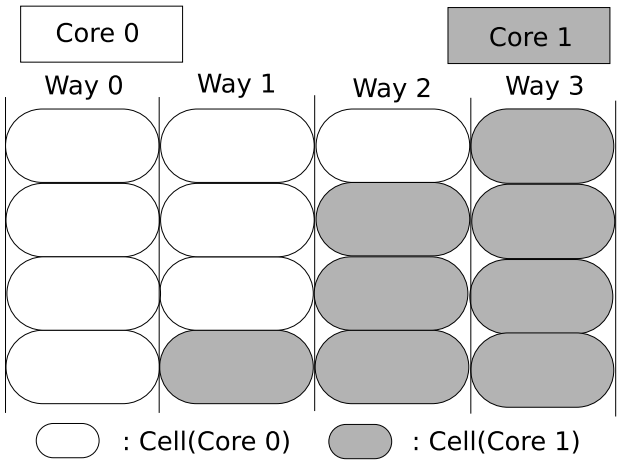

図 3.6: セル・アロケーションキャッシュの概要

セル・アロケーションキャッシュの割り当て領域は，図 3.6 のように 4 個のウェイを更に複数の細分化して割り当てを行う．この細分化したキャッシュ領域を「セル」と呼ぶ．また，図 3.6 における水平方向に存在する 4

個のセルで 1 個のブロックと呼ぶ．セルの割り当て変更を行う際，最初に各コアのミス率を比較する．コア 1 のミス率が最も高く，コア 0 が最も低い場合，コア 1 がキャッシュ領域をより必要としていると判断し，コア 0 のセルをコア 1 のセルへ割り当て直す．次に，最もミス率の高いコアの中でミス数が最も多いブロックを探す．ミス数が最大のブロックの中にミス率の低いコアのセルがあるか探索し，存在するならばその中で最もアクセスの古いセルの割り当て先をコア 1 に変更する．このような手法による割り当て方式によって，図 3.6 のように，ブロックごとに異なる負荷に応じた最適な割り当てができる．

3.2 セル・アロケーションキャッシュの問題点

従来のセル・アロケーションキャッシュには，キャッシュミス率が悪化してしまう場合があるという問題点がある．これはキャッシュミスが発生した場合に，主記憶からキャッシュへ書き込む際，セルへの書き込み制限により，使用頻度が高いデータを書き換えてしまう場合があるからである．これにより，キャッシュに格納できるデータ量が減少し，キャッシュミスが競合データの削減量よりも増加したと考えられる．使用頻度が高いデータを書き換えてしまう原因として，従来のセル・アロケーション

キャッシュの置換方式である，LRU(Least Recently Used) 方式が考えられる．LRU 方式は，最後に使用されてから最も長い時間が経ったデータを最初に捨てる置換方式である．しかし，使用頻度は考慮しないため，使用されてから経過した時間を優先し，キャッシュミスが増加する場合がある．LRU 方式は，最近使用されたデータはまた使用される可能性が高く，長時間未使用のページは今後も使われる可能性が低いという，時間的局所性の概念に基づいてデータ置換を行う．そのため，理論上は最適な置換方式に近い性能を出すことができるはずだが，完璧に使用頻度を考慮したデータの置換は行うことはできない．

4　キャッシュ領域割り当てアルゴリズムの改良

4.1　提案手法の概要

前章で述べたセル・アロケーションキャッシュの問題点を解決するため，本研究ではセル・アロケーションキャッシュの置換方式の変更というアプローチをとる．置換方式の候補の一つに ARC(Adaptive Replacement Cache) 方式 [3] という手法がある．ARC 方式とは使用されてから最も長い時間が経ったデータを最初に捨てる LRU 方式と，使用頻度が最も少ないデータを最初に捨てる LFU(Least Frequently Used) 方式の両方の長所を兼ね揃えており，LRU 方式よりもキャッシュミス率が低いという利点がある．しかし，ARC 方式は使用頻度を必ず考慮するので，例えば，過去に使用された頻度は高いが最後に使用されてから時間が経過しているようなデータを置換せずに残しておいてしまう．シングルコアの場合はこのような手法でも問題ないが，マルチコアの場合は当該セルを他のコアで利用できるように解放した方が効率が良くなると考えられるため，本論文ではこのようなセルは LRU 方式で置換する．そのため，LRU 方式と ARC 方式の 2 つの方式を切り換えて，状況に応じて最適な置換を行えるようにする．

4.2 提案手法のアルゴリズム

4.2.1 アクセス時間と使用頻度の記録

提案手法では図 4.7 のように，セルの使用回数を記録する機構を追加する．

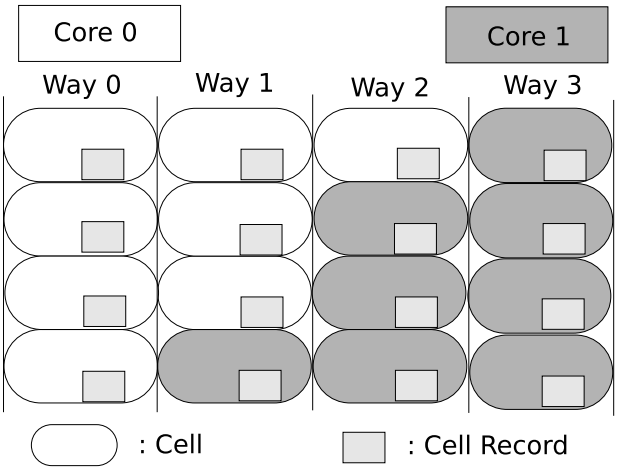

図 4.7: 提案手法の概要

各セルごとに最後にアクセスされた時間を変数に記録し，再度アクセスされた際に変数に記録されているアクセス時間を新たなアクセス時間で更新する。使用回数は最後にアクセスされた時間と同時に記録されて，各セルが使用された際に使用回数を記録するための変数を 1 インクリメントする．このような方法で最後にアクセスされた時間と使用頻度の両方を記録する．

4.2.2 割り当て方式の最適化

提案手法では，一定サイクル毎にセル割り当ての最適化を行う．具体的には割り当て変更を行う際，最初に各コアのミス率を比較する．コア 1 のミス率が最も高く，コア 0 が最も低い場合，コア 1 がより多くのキャッシュ領域を必要としているため，コア 0 のセルをコア 1 のセルへ割り当て直す．割り当て対象を決定後，最もミス率の高いコアの中でミス数が最も多いブロックを探す．次に，最もミス数の多いブロックの中から，使用されてから最も時間が経ったセルを探す．次に，発見したセルの使用頻度をチェックし，当該ブロック内の他の割り当て候補のセルの使用頻度と比較する．他の割り当て候補のセルの中に，発見したセルより使用頻度が高いものがあれば，そのまま発見したセルをミス率の高いコアに割り当てる．発見したセルの使用頻度が最も高ければ，ARC 方式に切り換えて割り当て直す．このような割り当て方式により，図 3.6 では最適に割り当てできなかった領域が，図 4.8 のように，ミス率が低いコア 0 のセルをミス率が高いコア 1 に割り当てられる．

図 4.8 の左上のセルは元々コア 0 に割り当てられていたセルである．このように割り当て直すことで，ミス率の高いコアで格納できるデータ量が増えてミス率を低減できる．

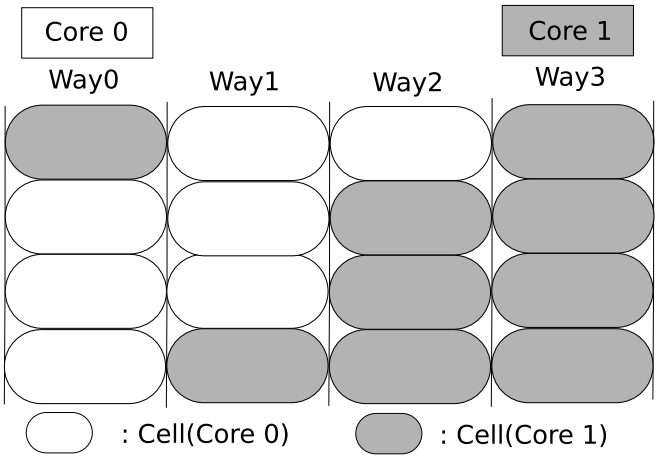

図 4.8: 改良アルゴリズムでの割り当て

5 性能評価

5.1 評価方法

提案手法を組み入れたセル・アロケーションキャッシュを，当研究グループで開発しているトレースドリブン型キャッシュシミュレータに実装し，評価した．このトレースドリブン型のキャッシュシミュレータは，実際に動作したデータの履歴をトレースすることでシミュレーションを行う．そのため，プロセッサシミュレータ上でベンチマークを実行して，そのトレースデータを作成する必要がある．そこで，マルチコア対応プロセッサシミュレータである Gem5[4] 上でベンチマークを実行し，キャッシュアクセスのトレースデータを作成し，シミュレーションを行った．評価用のベンチマークとして，Splash2[5] 及び姫野ベンチマーク [6] を使用

する．比較対象は通常キャッシュと，LRU 方式のみを用いている従来のセル・アロケーションキャッシュと，ARC 方式のみを用いているセル・アロケーションキャッシュとする．また，評価対象は性能面での評価を行うため，キャッシュミス率による性能比較を行う．その他の評価環境を表 5.1 に示す.

表 5.1: 評価環境

コア数	2 コア/4 コア
L1 キャッシュ	32KB
L2 キャッシュ	256KB × 8 ウェイ
セルの割り当て変更を行う間隔	8192 サイクル

5.2 評価結果

図 5.9，図 5.10 にそれぞれ 2 コア及び 4 コアプロセッサにおける評価結果を示す.

縦軸は通常キャッシュのミス率を 1 として正規化したものである.「LRU のみ」が従来のセル・アロケーションキャッシュ,「ARC のみ」が LRU の代わりに ARC を用いて実装したセル・アロケーションキャッシュ,「LRU と ARC」が提案手法である．縦軸が通常キャッシュと比較したミス率であるため，ミス率が小さいほど良い結果である．図 5.9 と図 5.10 より，従来の LRU 方式のみを採用したセル・アロケーションキャッシュと比較し

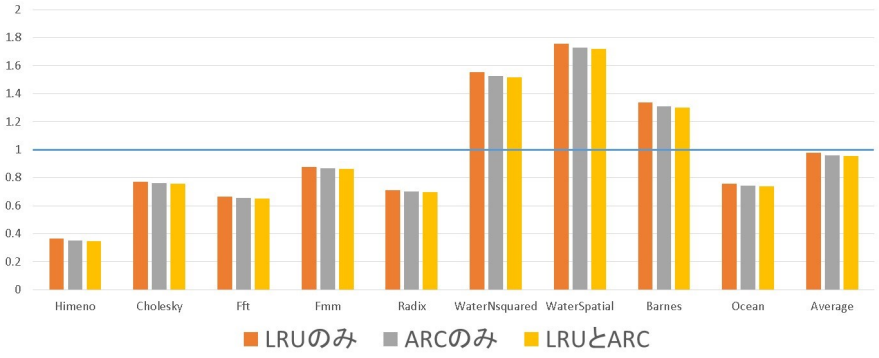

図 5.9: 通常キャッシュとのミス率（2 コア）

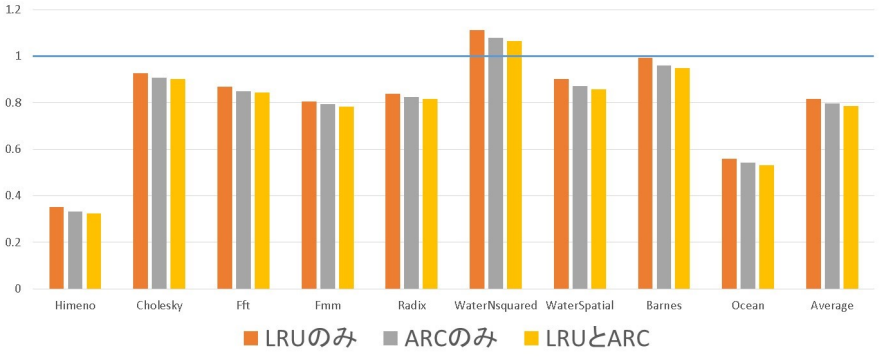

図 5.10: 通常キャッシュとのミス率（4 コア）

て，ミス率を 2 コアで最大 4.9%，平均 3.2%，4 コアで最大 3.7%，平均 2.1%低減できた．

5.3 考察

従来の LRU 方式のみを採用したセル・アロケーションキャッシュと比較して，ミス率を 2 コアで最大 4.9%，平均 3.2%，4 コアで最大 3.7%，平均 2.1%低減できた．しかし，2 コアで平均 3.2%，4 コアで平均 2.1%と大幅なミス率の低減には繋がらなかった．原因として，多くのデータが LRU 方式で正しく置換が行われており，ARC 方式を使用しなければ正しく置換されないデータが想定していたよりも少なかったためであると考えられる．更なるミス率の低減を目指すために，より多くのデータで評価を行い，アルゴリズムの妥当性を調べることが必要である．

6　おわりに

パソコンやスマートフォンの普及に伴い，高性能化が求められている．高性能化の手法の一つとして，複数のコアを用意し並列に処理を行うマルチコア化が挙げられる．しかし，マルチコア環境では複数のアクセスが同時に発生し，シングルコア環境よりデータの競合が発生しやすくなる．メモリアクセスの遅さは性能のボトルネックの一つであるため，キャッシュの改良により性能を向上させる研究が行われている．その内の一つとして，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，セル・アロケーションキャッシュは，使用頻度が高いデータを書き換えてしまう場合があり，かえってキャッシュミスが増加する場合がある．そこで本研究では，従来のセル・アロケーションキャッシュの問題点を解決するために，LRU 方式と ARC 方式の動的な切り換えを提案・評価した．その結果，従来の LRU 方式のみを採用したセル・アロケーションキャッシュと比較して，ミス率を 4 コアで最大 3.7%，平均 2.1%，2 コアで最大 4.9%，平均 3.2%低減できた．また，ARC 方式のみを採用したセル・アロケーションキャッシュと比較して，ミス率を 4 コアで最大 0.8%，平均 0.6%，2 コアで最大 1.4%，平均 1.0%低減できた．全てのベンチマー

クにおいてキャッシュミス率の低減に成功したが，大幅な低減には繋がらなかった．これは多くのデータがLRU方式で正しく置換が行われており，ARC方式を使用しなければ正しく置換できないデータが想定していたよりも少なかったためであると考えられる．今後の課題として実行時間の削減，置換方式の変更以外でのミス率低減方法の検討などが挙げられる．

謝辞

本研究の機会を与えて頂いた近藤利夫教授，並びにご指導，ご助言頂いた佐々木敬泰助教授，深澤祐樹研究員，修士 2 年の刀根舞歌先輩に深く感謝いたします．また，様々な局面でご助力頂いたコンピュータアーキテクチャ研究室の皆様にも心より感謝いたします．

参考文献

[1] 小川 周吾, “置換データの性質に着目した動的キャッシュパーティショニング,” 研究報告計算機アーキテクチャ（ARC）, 20 号, pp.1-8, July 2009.

[2] 刀根舞歌, 佐々木敬泰, 深澤祐樹, 近藤利夫, “キャッシュの分割領域の動的管理による高速化,” 信学技報, Vol.CPSY2016-19, pp.119–124, August 2016.

[3] Nimrod Megiddo and Dharmendra S. Modha, “ARC:a surf-tuning, low overhead replacement cache,” 03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies, pp.115-130, March 2003.

[4] gem5, <http://www.gem5.org/Main_Page/> (2017 年 3 月 2 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)

[6] 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)