卒業論文

題目

細分化ウェイ領域の動的割り当てを用いた高性能キャッシュの提案と評価

指導教員

佐々木 敬泰 助教

2016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

角谷 達明(412819)

内容梗概

プロセッサの高性能化手法の一つとして，並列化手法であるマルチコアプロセッサが普及している．しかし，並列プロセスを実行することで，シングルコアプロセッサと比較しメモリアクセスが増加する．このメモリアクセスはキャッシュへのアクセスと比較して低速であるため，性能のボトルネックとなる可能性がある．これを回避するにはキャッシュの性能向上が必要である．そのため，キャッシュをより効率的に扱うことで性能向上を図る研究が行われている．この研究の一つとしてコアがキャッシュにアクセスする領域を割り当て，制限するキャッシュパーティショニングと呼ばれる手法が存在する．キャッシュパーティショニングはセットアソシアティブキャッシュにおけるメモリアクセス手法の一つであり，各コアにアクセス可能なウェイを割り当て，各コアが扱うデータの場所を制限することができる．また，各コアの負荷に応じて割り当てるウェイ数を動的に制御することで，タスクに対して適切なキャッシュ容量を割り当てることが可能となる．しかし，割り当て単位がウェイであるため，タスクのメモリ要求的に全てのコアが必要としないウェイが存在する場合がある．先行研究として，不要なウェイを未割り当ての状態にするウェイ・アロケーションが存在する．ウェイ・アロケーションは，未割り当てとなったウェイを，電力を必要としない不活性状態とすることで性能低下を抑えつつ，消費電力の削減を行っている．しかし，ウェイ・アロケーションとキャッシュパーティショニングは共有データを扱えないという問題を持つ．加えて，ウェイを単位とした割り当てのみであるため，各コアへの割り当てが最適でない場合が多く存在し得る．そこで本稿では，共有データを扱うことができ，かつウェイをさらに細かく分割した『セル』という単位での割り当てを行うことで，より細かい領域で管理することにより，高性化を図るセル・アロケーションキャッシュを提案する．提案手法を評価した結果，同一のキャッシュ容量である通常キャッシュと比較して，最大 26.4 %，平均 9.3 %のミス率の低減に成功した．

Abstract

Multi-core processor is common technique for achieving high computing performance. In many multi-core processor architectures, all cores share L2 and last level cache memory. Thus, a performance of an entire multi-core processor depends strongly on a performance of shared cache memory. In particular, miss ratio of shared cache memory is one of the most important factor because every processor needs to wait for 100 to 1000 clock cycles when an access-miss occurs on shared cache memory. In addition, multi-core processor spoils temporal and spatial locality on shared cache memory that is the most important concept of memory to reduce a number of access-miss because necessary data and its allocated locations on cache memory are different cores and programs on it. Hence, this study focuses on reducing a number of access-miss on shared cache memory in order to achieve high-performance multi-core processing. This study proposes Cell-Allocation Cache (CAC) that is fine-grain dynamic assigning of cache region on each core.As a prerequisite, CAC targets on set-associative shared cache memory and multi-core processor. Features of CAC are mainly following three points. Firstly, CAC uses cell that is the minimum unit corresponding with the way and some indexes (on the other words, 'cell' is composed of some cache lines on each cache way) for re-assigning of cache region. This is the suit unit to maintain a cache region assigning and serve optimum cache region. Secondly, cache region assigning of CAC is dynamically changed depending on workload of executing programs on each core. Therefore, CAC can adapt to changing of behavior of processor. Finally, CAC assigns dedicated cells on each core that allows to read from every core and not allows to write from non-assigned core. Owning to this, CAC can reduce a number of replacing of every cache entry with remaining readable region. For these reasons, CAC achieves to reduce a number of access-miss on shared cache memory. This study also evaluates a miss ratio of CAC on software simulator, Gem5 using Splash2 and Himeno benchmarks. Accoding to the results, CAC achieves to reduce cache miss ratio by 26.4% on maximum and 9.3% on average compared with conventional cache memory.

目 次

図 目 次

表 目 次

1　はじめに

1.1　研究背景

プロセッサの高性能化手法として並列処理が存在する．その一つにマルチコアプロセッサが普及しているが，並列プロセスを実行することで，シングルコアと比較してメモリアクセスが増加する問題がある．メモリアクセスはキャッシュへのアクセスと比較して低速であるため，メモリアクセスの増加は大幅な性能低下に繋がる危険性がある．すなわち，このボトルネックを解決するには，キャッシュに求めるデータが存在しないことでメモリアクセスを引き起こす原因となる，キャッシュミスの削減による主記憶へのメモリアクセスの低減が重要である．そこで，キャッシュをより効率的に扱うことでキャッシュミスを削減し，性能向上を図る研究が行われている．

1.2　研究目的

キャッシュミスの削減手法の研究として，各コアがキャッシュにアクセス可能な領域を割り当て，制限するキャッシュパーティショニング [1] が存在する．キャッシュパーティショニングはマルチコア環境において，各コアで実行されているタスクが異なるため，必要なキャッシュ容量が違う

ことを利用する．各コアの負荷に応じて，アクセス可能なキャッシュ領域を動的に各コアに割り当てる．これにより各コアで必要なキャッシュ容量を割り当てることが可能となるため，メモリアクセスが削減できる．しかし，キャッシュパーティショニングによる割り当てでは，実際のタスクが求めるキャッシュ容量に対し，動的に割り当てるキャッシュ容量の最小単位が大きいことから，キャッシュを最大限利用できない．そこで，本研究では共有キャッシュの領域をより小さな単位である『セル』に分割し，管理するセル・アロケーションキャッシュを提案・評価する．その結果，通常キャッシュと比較して，ミス率を最大26.4 %，平均9.3 %低減した．

2　従来研究とその問題点

2.1　キャッシュパーティショニング

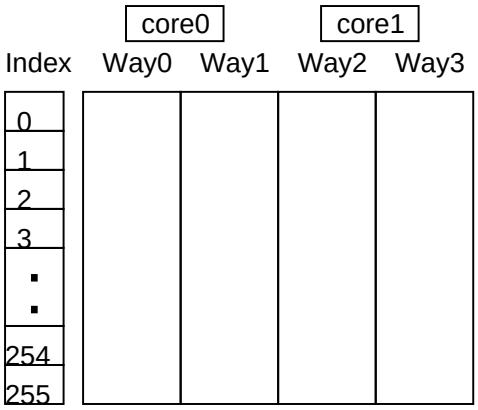

図 2.1: セットアソシアティブキャッシュ概要

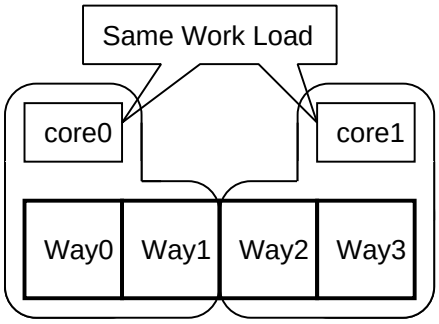

図 2.2: 同負荷におけるキャッシュパーティショニング割り当て

キャッシュパーティショニングはマルチコア環境におけるメモリアクセス手法の一つであり，セットアソシアティブキャッシュのウェイを利用する．そこで，図 2.1 にウェイと呼ばれる領域を複数持つキャッシュ構造で

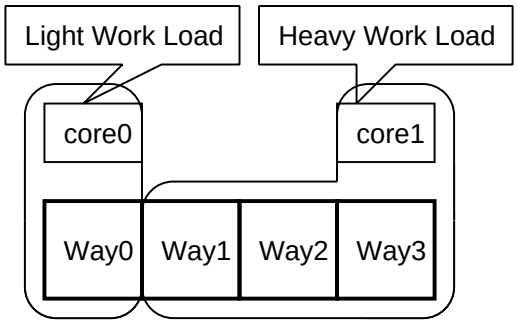

図 2.3: 異なる負荷におけるキャッシュパーティショニング割り当て

あるセットアソシアティブキャッシュを示す．アドレスを A，キャッシュメモリが N ブロックとすると，データを配置するブロック，すなわちインデックスは $A \bmod N$ という式で計算する．実際のキャッシュではインデックスは 4k〜16k エントリ程度あるため，より膨大なエントリ数を持つ．ここで，データアクセス時のアドレスが N の倍数に偏っていた場合，同一ブロックに割り当てが集中するため著しく性能が低下する．そこで，通常はキャッシュメモリを複数ウェイ分用意することにより，同一インデックスのデータを同時に複数格納可能としている．各コアではそれぞれ異なるタスクを実行するため，要求されるメモリ要求が異なる．そこで，キャッシュパーティショニングは各コアにアクセス可能なキャッシュ領域を動的に割り当て，各コアが扱うデータの場所を制限する．これにより，適切なキャッシュ容量を割り当てることが可能となる．このとき，

割り当ては各コアの負荷に応じて必要なキャッシュ領域の割合を決定し，各コアにウェイを単位として割り当てる．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．例として，各コアの負荷が同程度の場合は図 2.2 のように同程度のウェイ数が割り当てられる．一方，各コアの負荷が大きく異なる場合，例えばコア 0 の負荷が小さく，コア 1 の負荷が大きい場合には，図 2.3 のようにコア 1 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容量を動的に割り当てることで，キャッシュの領域を効率的に扱うことが可能となるため，キャッシュの未使用領域を削減することに繋がり，全体のキャッシュミスを削減できる．

2.2 ウェイ・アロケーション

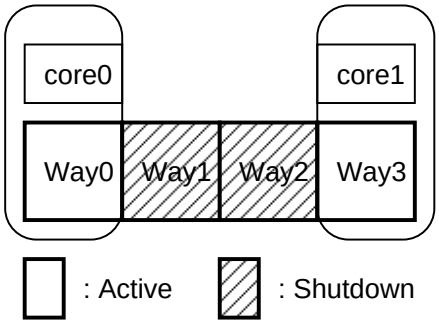

図 2.4: ウェイ・アロケーションによる割り当て

ウェイ・アロケーション [2] はキャッシュパーティショニングを応用した

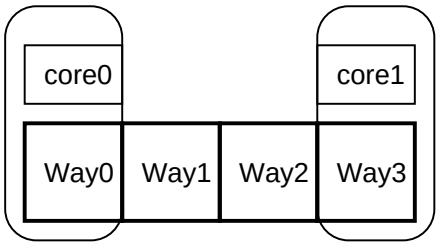

図 2.5: 求める割り当て領域

手法である．図 2.4 にウェイ・アロケーションを用いたキャッシュ容量の割り当てを示す．図 2.4 の斜線部は電力を消費しないシャットダウン状態である．ウェイ・アロケーションのアクセス方法は，キャッシュパーティショニングと同様であるが，ウェイの割り当てにシャットダウン状態が存在する．例として，全てのコアの負荷が小さい場合を考える．図 2.5 はこの時に必要とされるキャッシュ容量，および割り当てである．しかし，第 2.1 節で示したキャッシュパーティショニングの手法の場合，図 2.2 のようにウェイを均等に割り当てる状態になることが考えられる．すなわち，図 2.2 の割り当てでは，キャッシュ容量が過剰である．そこで，ウェイ・アロケーションは割り当てを図 2.5 のように最低限に抑え，かつ未割り当ての領域を，図 2.4 のようにシャットダウン状態にする．これにより，性能低下を抑えつつ消費電力を削減することができる．

2.3 従来研究の問題点

2.3.1 共有データ非対応

第 2.1, 2.2 節で示した従来研究は各コアがアクセスできる領域を制限することで，キャッシュをより効率的に使用しているが，共有データを考慮していない問題がある．そのため，共有データを使用するプログラムを実行する場合に問題が発生する．図 2.6 に従来研究におけるコア 0 のデータアクセスの制限を示す．従来研究は図 2.6 における矢印の指す部分にのみアクセスが可能となっている．そのため，図 2.6 から，異なるコアへ割り当てられているキャッシュ領域にアクセスが不可能である．そこで例として，2 コアプロセッサにおいてキャッシュパーティショニングを用いた構成を考える．図 2.3 のようにキャッシュ容量を割り当てた場合，コア 0 はコア 1 へ割り当てられた領域へアクセスすることができないため，図 2.6 のようにウェイ 1 からウェイ 3 へのアクセスはできない．また，図 2.6 のように共有データがウェイ 2 に存在する場合，コア 0 はアクセスが不可能であるため，共有データがキャッシュに格納されていても使用できない．共有データを使用できないという問題は，近年のマルチコアやマルチスレッドのようにデータ共有が発生しうる並列処理において大きな問題である．

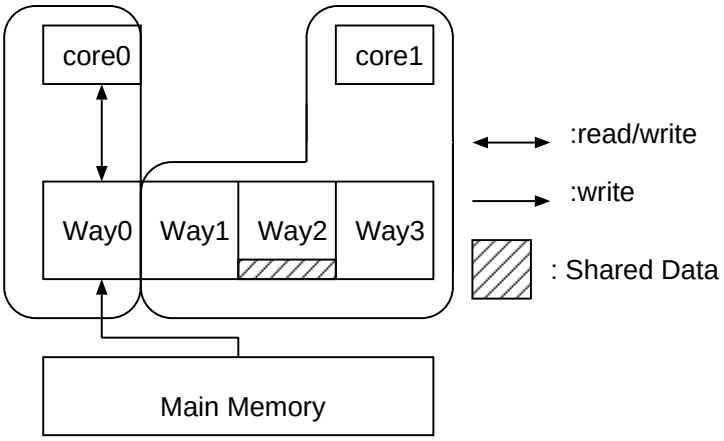

図 2.6: 共有データにおける問題

2.3.2 不適切な割り当て方式

従来研究は図 2.1 のような，セットアソシアティブキャッシュを用い，キャッシュ領域の割り当てのためにウェイ単位での割り当て変更を行う．しかし，プログラムごとに扱うデータは異なることから，インデックスに偏りが生じる可能性が十分にある．そのため，ウェイ単位では最適な割り当てが不可能である．ここで，図 2.7 はキャッシュパーティショニングによる割り当てであり，図 2.8 は理想的な割り当てである．例として，図 2.7 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前半に集中しており，コア 1 はそれ以外の領域でアクセスが集中していると，各コアの割り当ては図 2.8 が望ましいが，キャッシュパーティショ

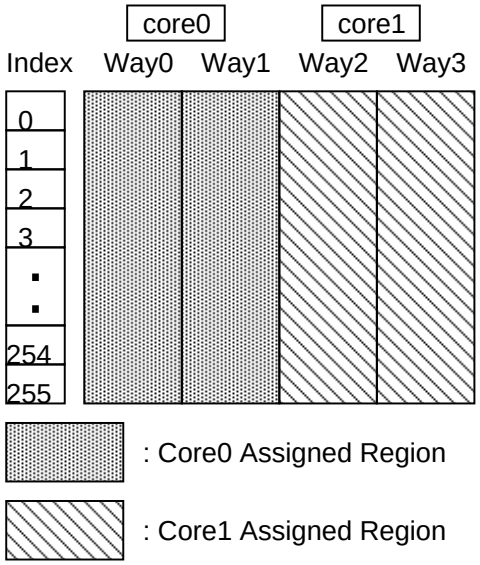

図 2.7: ウェイ割り当てによる割り当て図

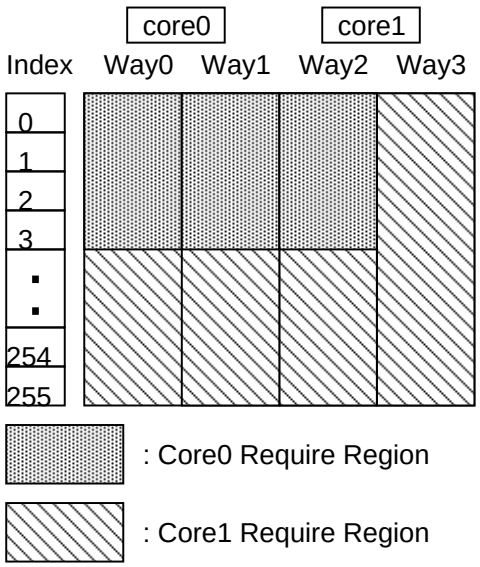

図 2.8: 理想的な割り当て図

ニングはウェイ単位での割り当てを行っているため，全体の負荷が同程度の場合，実際に可能な割り当ては図2.7である．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の提案

3.1　セル・アロケーションキャッシュの概要

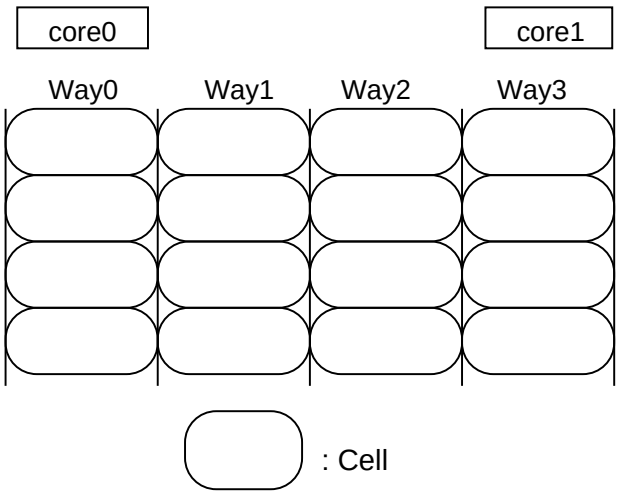

図 3.9: セル・アロケーション割り当て図

前章のキャッシュパーティショニングとウェイ・アロケーションにおける問題に対して，共有データを扱うことが可能であり，かつより細かな領域を割り当てられるセル・アロケーションキャッシュを提案する．セル・アロケーションキャッシュは，メモリからの書き込みの場合に限り，各コアに割り当てられたキャッシュ領域に対してのみ書き込みを行うように制限する．しかし，コアからの読み込みや書き込みのアクセスは制限することなく通常のキャッシュと同様にアクセスを行うことで共有データを扱うことを可能にしている．また，ウェイ単位による割り当てを行うため，

求めるキャッシュ容量を適切に割り当てることができない問題に対して，割り当て領域をウェイ単位よりさらに細分化することでより適切な割り当てを行う．図 3.9 にセル・アロケーションを用いたキャッシュ構成を示す．図 3.9 の 1 つの領域は複数のインデックスをまとめたものであり，回路の肥大化や遅延などを考慮し，全体で 4k や 16k 存在するインデックスを実験的に 4 等分している．この細分化した領域をセルとし，図 3.9 の様に各コアで必要とする領域を，セルを単位として割り当て, インデックスを考慮した割り当てが可能とすることでパフォーマンスの向上を図る．

3.2 セル・アロケーションキャッシュのアルゴリズム

3.2.1 共有データへの対応

セル・アロケーションは全ての読み込み・書き込み命令に制限を掛けるキャッシュパーティショニングとは異なり，各コアからの読み込み・書き込み命令のアクセスはキャッシュパーティショニングの割り当てを無視し，キャッシュ全体にアクセスできる．図 3.10 にセル・アロケーションにおけるアクセス制限を示す．図 3.10 のようにウェイ 2 に共有データが存在する場合，キャッシュパーティショニングの場合はコア 0 からアクセスができない．それに対して，セル・アロケーションのアクセス方法であればウェイ 2 に存在する共有データを扱うことが可能となる．ただし，メ

モリ側からキャッシュへ書き込む場合のみ，コアごとのアクセス制限を設ける．この変更により，キャッシュパーティショニングでは非対応であった共有データを扱うことができる．

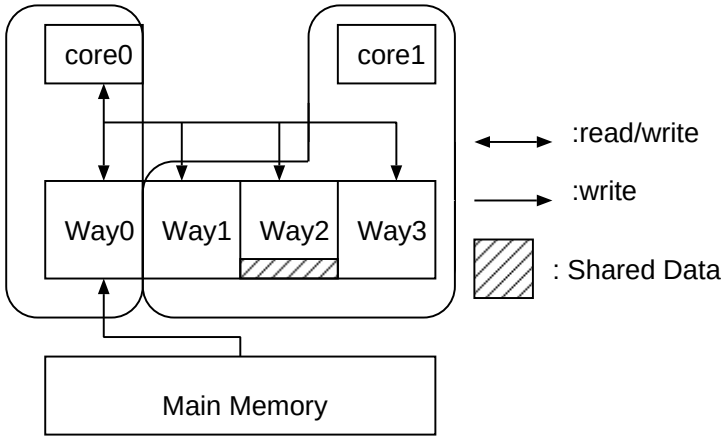

図 3.10: セル・アロケーションデータアクセス概要

3.2.2 割り当て方式の最適化

セル・アロケーションの割り当て領域は図 2.7 のようなウェイ単位での割り当てと異なり，図 3.9 のようにウェイ単位からさらに細分化した単位での割り当てを行う．また，同一のインデックスを持つセル，すなわち図 3.9 における行方向に存在する 4 つのセルで 1 つのセル群と呼ぶ．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 0 からコア 1 に割り当てを変更することを決定する．割り当て対象を決定後，最もミス率の高いコアの

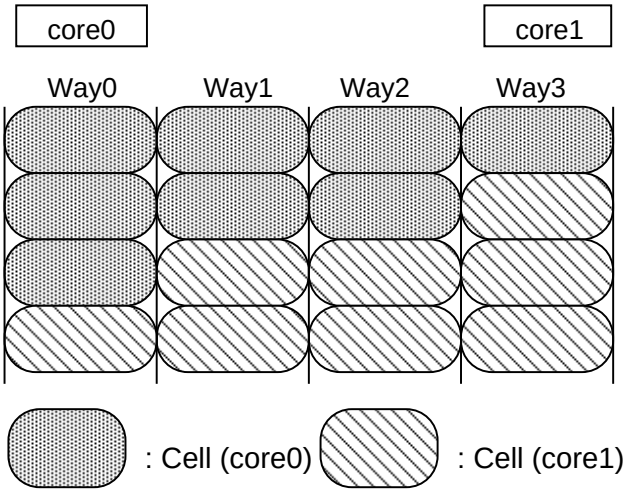

図 3.11: セル・アロケーション概要

中でミス数が最も多いセル群を探す．ミス数最大のセル群を発見後，同じ領域に存在するセル群で，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．このような提案手法による割り当て方式によって，図 3.11 のように，セル群ごとに異なる負荷に応じた最適な割り当てを実現できる．

4 性能評価

4.1 評価方法・項目

表 4.1: 評価環境

コア数	2 コア/4 コア
スレッド数	2 スレッド/4 スレッド
L1 キャッシュ	32kB
L2 キャッシュ	256kB × 4WAY
セルの割り当て変更を行う間隔	8192cycle

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を改造してセル・アロケーションを導入，評価を行う．このトレースドリブン型の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，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[3] 上でベンチマークを実行，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う．評価用のベンチマークとして，共有メモリ型マシンのベンチマークプログラムとして広く用いられている splash2[4] 及び姫野ベンチマーク [5] を使用する．比較対象は通常キャッシュとし，評価環境は表 4.1 として比較を行う．なお既存研究は共有データを扱えない

ため対象外とする．また，評価対象は性能面での評価を行うため，ミス率における性能比較を行う．

4.2 評価結果

図 4.12，図 4.13 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．縦軸が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り，横軸が提案手法を導入した状態で実行した各ベンチマーク名である．縦軸が通常キャッシュと比較したミス率であるため，提案手法のミス率が小さいほど良い結果である．図 4.12 と図 4.13 より，提案手法は全体的にミス率を低減できており，最大で 26.4 %，平均で 9.3 %低減できた．

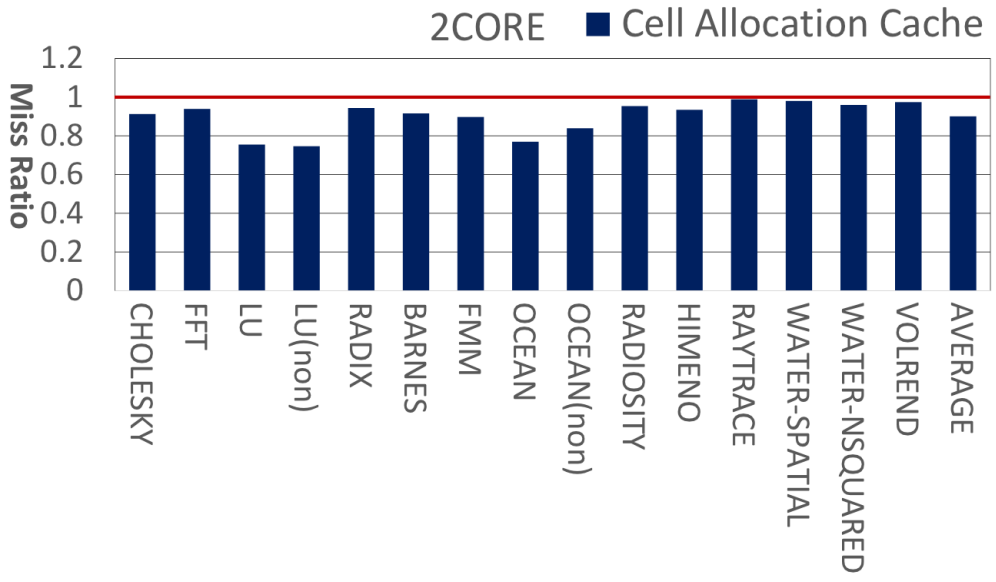

図 4.12: 通常キャッシュとのミス率（2 コア）

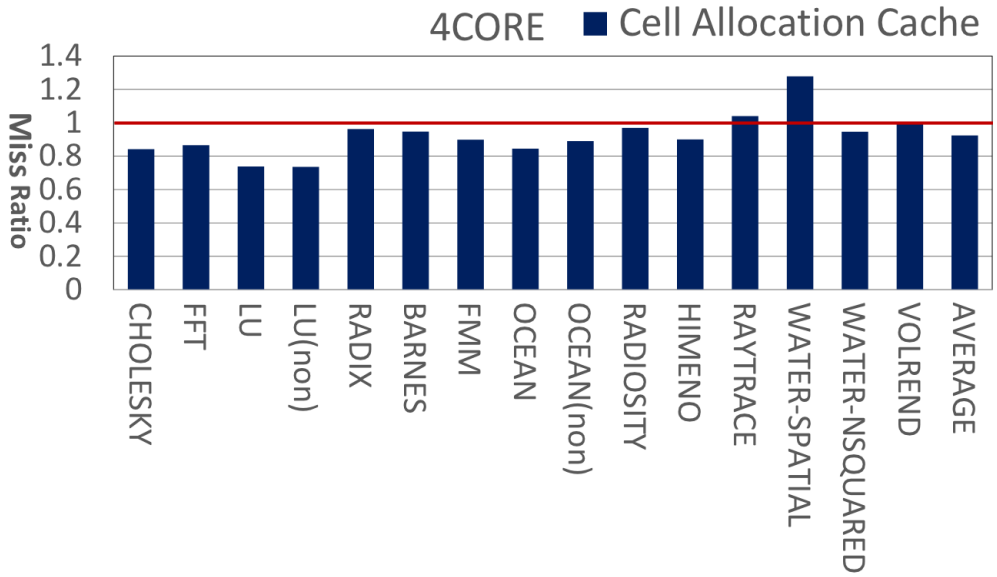

図 4.13: 通常キャッシュとのミス率（4 コア）

4.3 考察

全体的にミス率が低減していることから，割り当てのアルゴリズムは良好であることが判断できる．しかし，4 コアでの評価結果において，RAY-TRACE と WATER-SPATIAL の項目のみが大きく悪化しており，特に WATER-SPATIAL の項目に関しては 27.9%の悪化を示している．各項目とも 2 コアでのミス率の削減に成功していることから，2 コアから 4 コアへ条件を変更したことに原因があると予想できる．考えられる原因として，図 3.9 における最上段のセル群において，セル・アロケーションは実装上，各コアで最低でもセルを 1 つは保持する性質を持つ．そのため，

ウェイ数とコア数が同じ場合，最上段のセル群は割り付け変更が不可能となっていることが原因であると考えられることから，その改善を図ることで，さらにミス率を低減できると考えられる．

5　おわりに

キャッシュパーティショニングを元に共有データを扱うことができ，インデックスの偏りを考慮した割り当てができるセル・アロケーションキャッシュを提案・評価した．その結果，通常キャッシュと比較して最大26.4％，平均9.3％性能が向上した．今後の課題として，割り当て手法のさらなる改良，未使用領域のスリープ・シャットダウンによる電力削減や面積の見積もりとその改善などがあげられる．また，ソフトウェアシミュレーションでの評価であるため，ハードウェアでの実装を行い，今回行えていないサイクル数の正確な評価を行う必要がある．

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，深澤祐樹研究員，修士１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, p.1-8, 2009-07-28

[2] 小寺 功消, “ 費電力を考慮したウェイアロケーション型共有キャッシュ機構, ” 情報科学技術レターズ, 6 巻, p.55-58, 2007-08-22

[3] gem5, http://www.gem5.org/Main_Page/

[4] The Modified SPLASH-2, http://www.capsl.udel.edu/splash/

[5] 姫野ベンチマーク—理化学研究所情報基盤センター, http://accc.riken.jp/supercom/himenobmt/