2A-04

Gem5 を用いた高効率な分割領域動的管理キャッシュの実環境下における性能評価

鬼頭　優人†　　佐々木　敬泰†　　深澤　祐樹†　　近藤　利夫†

1. まえがき

近年，スマートフォンやパソコンなどのプロセッサでは，高性能化と低電力化の両立が求められており，様々な高速化・低消費電力化手法が提案されている．そのうち，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，結果としてキャッシュミス率が高くなるという問題がある．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，高性能化を達成するためには主記憶への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，著者ら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[1]を提案しているが，トレースドリブン・シミュレータでしか評価していなかった．一般に並列処理ではほかのスレッドやプロセスとの依存関係によりプログラムの振る舞いや性能が変わるため，より現実的な評価を行うためにはアーキテクチャレベルのシミュレーションを行うことが望ましい．そこで，本研究では，セル・アロケーションキャッシュをアーキテクチャシミュレータ Gem5 上に実装することで，より現実的な環境下での評価を行った．マルチスレッドベンチマークで評価した結果，通常キャッシュや従来の固定的なキャッシュ・パーティショニングと比較して，ヒット率がそれぞれ最大 44.3 %，5.2 %向上した．

2. セル・アロケーションキャッシュ

2.1 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はキャッシュ・パーティショニング[2]を応用した手法で，より細かい粒度でキャッシュメモリを管理できる．一般にマルチコアプロセッサでは各コアが共有キャッシュメモリを奪い合って性能が落ちることがある．そこで，キャッシュ・パーティショニングではコアにウェイを割当て，コア間での競合を解消している．しかしながら，プログラムには時間的局所性，空間的局所性があり，アクセスするメモリ領域は偏っている．そのためコアにウェイを丸ごと割り付けるキャッシュ・パーティショニングでは，使用頻度の低い領域ができてしまい，キャッシュを有効活用できない危険性がある．

そこで，セル・アロケーションキャッシュでは 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当てる．セル・アロケーションキャッシュの概念図を図 1 に示す．セルは複数のエントリをまとめたグループである．また，図 1 の破線のように，ウェイ方向に並んだセルのグループをブロックとする．図 2 にセルの詳細を示す．セルはエントリごとにブロック番号を格納するビット列を持つ．また，データ共有を考慮し，各セルをデータ共有率とミス率に応じて，共有セルと固有セルに分ける．共有セルは通常キャッシュと同様にアクセスできるセルである．一方，固有セルは,キャッシュミス時のリプレース対象先を，割当てられたコアに限定するセルである．この 2 種類のセルを使い分けることで，データを共有しつつ，キャッシュメモリの競合を減らし，キャッシュの利用効率を高める．

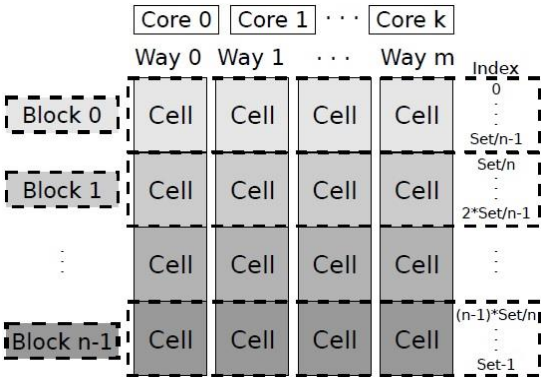

図 1. セル・アロケーションキャッシュの概念図

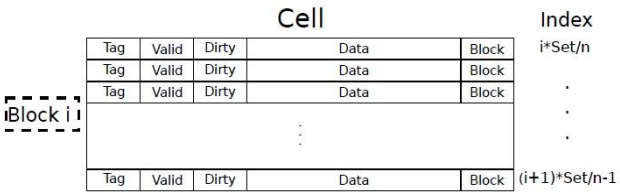

図 2. セルの詳細図

2.2 セル・アロケーションキャッシュの動作

セル・アロケーションキャッシュと通常キャッシュとの違いは，キャッシュミス時のデータの格納先を決める場合に，同じコアに属する領域へ優先的に書き込む点である．

共有セルと固有セルのキャッシュミス時の動作を説明する．あるコアでキャッシュミスが発生した場合，メモリからデータを取得する．その際の書き込み先は，共有セルかそのコアの固有セルが優先される．共有セルとそのコアの固有セルの両方が存在する場合，その中から LRU 法でどのセルに書き込むか決定する．共有セルとそのコアの固有セルに書き込めない場合にのみ，他のコアの固有セルを共有セルにして書き込む．共有セルが無く，そのコアの固有セルが存在する場合，固有セルの中から LRU 法でどの固有セルに書き込むか決定する．そのコアの固有セルに書き込めない場合にのみ，他のコアの固有セルを共有セルにして書き込む．

†三重大学大学院工学研究科情報工学専攻

3. 提案評価手法

文献[1]では，トレースドリブンベースのシミュレーションにより性能評価を行っている．具体的には，事前にベンチマークプログラムを実行してメモリアクセスを記録し，オフラインのトレースドリブン・シミュレータを用いて提案手法の評価を行っていた．しかし，一般に並列プログラムでは，他のスレッドやプロセスの動きにより，プログラムの振る舞いや性能が大幅に変わる場合がある．例えば，マスタ・ワーカー・モデルのプログラムの場合，スレッド間の僅かなタイミング差で処理の割り当てや振る舞いが大幅に変わる可能性がある．また，トレースドリブンシミュレーションでは，命令間の依存関係は表現できないため，ロード命令の遅延が他の後続命令に与える影響を正確にシミュレーションできない．

そこで，本研究では提案手法のより正確な評価，及び動作解析を目指して，アーキテクチャシミュレータである Gem5 に提案手法を組み込むことを提案する．

4. 性能評価

4.1 評価方法

セル・アロケーションキャッシュをアーキテクチャシミュレータ Gem5 の L2 共有キャッシュに実装し，評価した．比較対象は通常キャッシュとキャッシュ・パーティショニング，評価項目はキャッシュヒット率とする．評価環境はスレッド数 4，コア数 4，ウェイ数 8，セル数 16 とする．また，ベンチマークとして姫野ベンチマーク[3]を使用する．モードを切り替えるインターバルはロードストア命令数が 5，10，20，25，30[$\times 10^4$]の倍数の時とし，共有データかどうか判定する閾値は 0.00 から 0.50 までを 0.05 刻み，0.50 から 1.00 までを 0.1 刻みで変化させ，評価した．なお，本研究では 2 つ以上のスレッドから一度でもアクセスのあったものを共有変数と考え，セル内のデータのうち閾値を超える割合の共有変数があれば共有セルに遷移させる．

4.2 評価結果

図 3 に改造した Gem5 を用いて評価したセル・アロケーションキャッシュのヒット率を示す．閾値が 0.00 の時は通常キャッシュと，1.00 の時は 1 コア当たり 2 ウェイを割り当てた理想的なキャッシュ・パーティショニングと同等となっている．通常キャッシュ，従来のキャッシュ・パーティショニングと比較して，ヒット率はそれぞれ最大 44.3 %，5.2 %向上した．また，閾値 0.30，インターバル 20[$\times 10^4$]の時，1 番ヒット率が高く 38.03 %になった．閾値 0.3 の時が平均して 1 番ヒット率が高く，それ以降はヒット率が伸びない傾向になり，インターバルが違ってもヒット率はほぼ同じくらいになった．最終的に閾値 1.0 の時には全てヒット率が同じになった．また，閾値が 0.00 から 0.25 まではインターバル 30[$\times 10^4$]でヒット率が最大になったが，0.30 から 0.90 までは 20[$\times 10^4$]で最大になった．

4.3 考察

紙面の都合上，詳細なデータは省略するが，従来のトレースドリブンレベルのシミュレーションでは，共有セルと固有セルの導入だけでは性能向上があまり得られなかった．しかし，アーキテクチャレベルシミュレーションを行った結果，通常キャッシュやキャッシュ・パーティショニングと比較して，閾値によって全体的にヒット率が向上したため，セル・アロケーションキャッシュによる効果があることを確認できた．

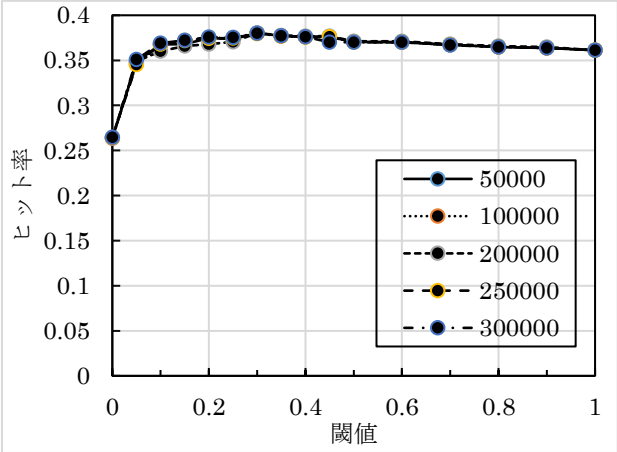

図 3. キャッシュヒット率

閾値は 0.30 の時，共有セルと固有セルの使い分けが効率良く行われたため，ヒット率が一番良くなったと考えられる．それ以降の閾値では，共有率が超えられなくなったため，固有セルが多い状態となりキャッシュ・パーティショニングと近い状態になったため，ヒット率は最大時に比べて低下したと考えられる．

5. まとめ

本稿では，プロセッサの高性能化手法として，ウェイより細かい領域で管理する，セル・アロケーションキャッシュをアーキテクチャシミュレータ Gem5 に実装することで，現実的な環境下における評価をした．通常キャッシュと比較した結果，ヒット率が最大 44.3 %向上した．また，固定的なキャッシュ・パーティショニングと比較した結果，ヒット率が最大 5.2 %向上した．

今後の展望としては，セル・アロケーションキャッシュでは動的な固有セルの割り当て数の変更や疑似的なウェイ拡張等も行っているが，今回は反映していないので，そこを実装する．また，今回の実装では，共有セルに一度なったセルは固有セルに戻らないので，戻るようにする．さらに，ベンチマークは今回 1 つのみを使用したが，他のベンチマークや複数のベンチマークの組合せでも評価する．また，今回は時間の都合上，ヒット率のみの評価になっているが，サイクルアキュレートなシミュレーションを行い，実行サイクル数，実行時間，実行速度での評価をとることが挙げられる．

参考文献

[1]. 刀根舞歌,他，“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 (2016).

[2]. 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

[3]. 姫野龍太郎，“ 姫野ベンチマーク ”，http://accc.riken.jp/supercom/himenobmt/，参照 July 2016.