4A-4

CMP向け分散キャッシュの可視化ツールを用いた効率化

藤枝 直輝 †　　吉瀬 謙二 †
東京工業大学 大学院情報理工学研究科 †

1 はじめに

プロセッサのシングルスレッド性能向上の限界により，プロセッサに搭載されるコアの数は増加傾向にある．多くのコアを搭載するプロセッサのキャッシュにおいては，コアごとの要求に応じて，いかに効率よくその容量を利用するかが重要になっている．

本稿では，CMP(Chip MultiProcessor) 向け分散キャッシュの研究ツールとして，その利用状況を可視化するツールである **SimMccc Viewer** について述べる．これを用いてキャッシュに対する要求により柔軟に応え，キャッシュの利用効率を高め，プロセッサの性能向上を目指す．本稿では，SimMccc Viewer の機能・設計・実装について述べ，実例を用いてその利用について概説する．

2 対象アーキテクチャ

本稿では，Elastic Cooperative Caching(ElasticCC)[1] のような，占有キャッシュをベースとしつつ，あるコアを追い出されたラインを別のコアへと移動させることで，共有キャッシュの柔軟性を追加する構成を対象とする．

図1に，ElasticCC の構成を示す．ElasticCC はディレクトリキャッシュを保持するいくつかの Distributed Coherence Engine(DCE) をもち，要求されたラインを持つコアの特定や，キャッシュのコヒーレンス制御を行う．また，各コアの L2 キャッシュは仮想的に占有 (Private) と共有 (Shared) に分割される．あるコアが追い出されたラインを別のコアで格納する時は，その格納先を共有領域のみとする．これにより占有領域を保護し，利用頻度の高いラインが誤って追い出されることを防ぐ．

このような構成において重要なことは，コアごとに異なるキャッシュへの要求を正しく把握すること，また，その情報に基づいて適切なラインの移動を行うことである．これらを満たす方式を検討するため，そして，実際にそのような方式の有効性を確認するためには，キャッシュが持つ情報をまとめ，わかりやすい形で提供する可視化ツールが大いに助けになる．

3 SimMccc Viewer

我々は，メニーコアシミュレータ SimMc[2] の拡張である SimMccc (SimMc for cooperative caching) のログを利用し，分散キャッシュの利用状況の可視化ツール SimMccc Viewer を構築した．以下にその機能と設計・実装について述べる．

SimMccc には，コア・キャッシュ・DCE の各モジュールがそれぞれ一定期間ごとに利用状況を取得し，それらをログとして出力する機能が備わっている．SimMccc Viewer は，モジュールごとに分割されたログを用途別に集計して XML 形式で出力するコンバータ (Ruby により実装) と，コンバータが出力した XML ファイルを読み込み，適切なグラフとして表示するビューア本体 (Java により実装) から構成される．

Improving Efficiency of CMP Cooperative Caching with a Visualization Tool
Naoki FUJIEDA† and Kenji KISE†
†Graduate School of Information Science and Engineering, Tokyo Institute of Technology

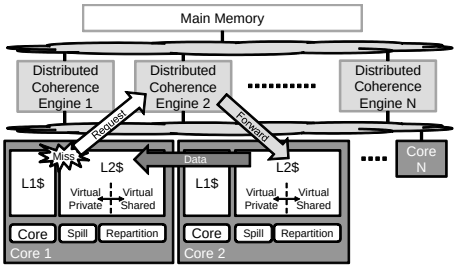

図 1: Elastic Cooperative Caching の構成.

前章で述べた目的を満たすため特に求められる機能は2つある．1つは，L2 アクセスを要求されたラインが前回利用されてからの時間と，アクセス結果 (自コアへのヒット，他コアへのヒット，またはミス) との関係を確認できることである．これは，コアのキャッシュへの要求を確認するのに有効である．もう1つは，占有領域・共有領域間のパーティションの変化と，それに応じて実際のライン数がどのように増減したかを確認できることである．これは，適切なパーティショニングやラインの移動ができているかを確認するのに有効である．これらを実現するために，SimMccc のシミュレーションにもいくつか追加の機能が必要となる．

第1の機能を実現するためには，ラインの物理アドレスごとに最後にコアからアクセスされた時刻を記録しておき，L2 アクセスの際にその時刻を参照する機能を備える必要がある．これを実現するために，全てのコアの L1・L2 キャッシュ，および全ての DCE から記録・参照可能なグローバルなテーブルを追加する．テーブルへの時刻の記録は L1 キャッシュが行う．自コアへのヒットは L2 キャッシュが，他コアへのヒットおよびミスは DCE が検出できるので，時刻の参照はこれらの双方で行えるようにし，アクセス間隔ごとのヒット回数やミス回数は，それを検出した L2 キャッシュと DCE とが独立して行う．これら分割されたログの集計は，前述のとおり SimMccc Viewer のコンバータが行う．

第2の機能を実現するためには，各ラインをプロセス ID とキャッシュの状態ごとに分類してライン数を数え上げ，パーティションの状態とともにロギングする機能をキャッシュに備える必要がある．これは，SimMccc の既存のログ出力機能にプロセス ID による場合分けを追加することで実現できる．

以上の機能を実現するために SimMccc に追加したコード量 (空行やコメント行を含む) は約 300 行である．また，現バージョンの SimMccc Viewer におけるコード行数は，コンバータが 207 行，ビューワ本体が 1,263 行である．

4 利用例

本章では SimMccc Viewer の利用例として，従来方式の ElasticCC と，そのパーティショニング方式を改良した HFC パーティショニング[3] との挙動の違いについて，これらを可視化することで明らかにする．評価パラメータは，論文 [3] と同様とする．ベンチマークには，Equation Solver Kernel[4] とクイックソートを同時に実行させたものを用いるが，クイックソートについてはデータセット

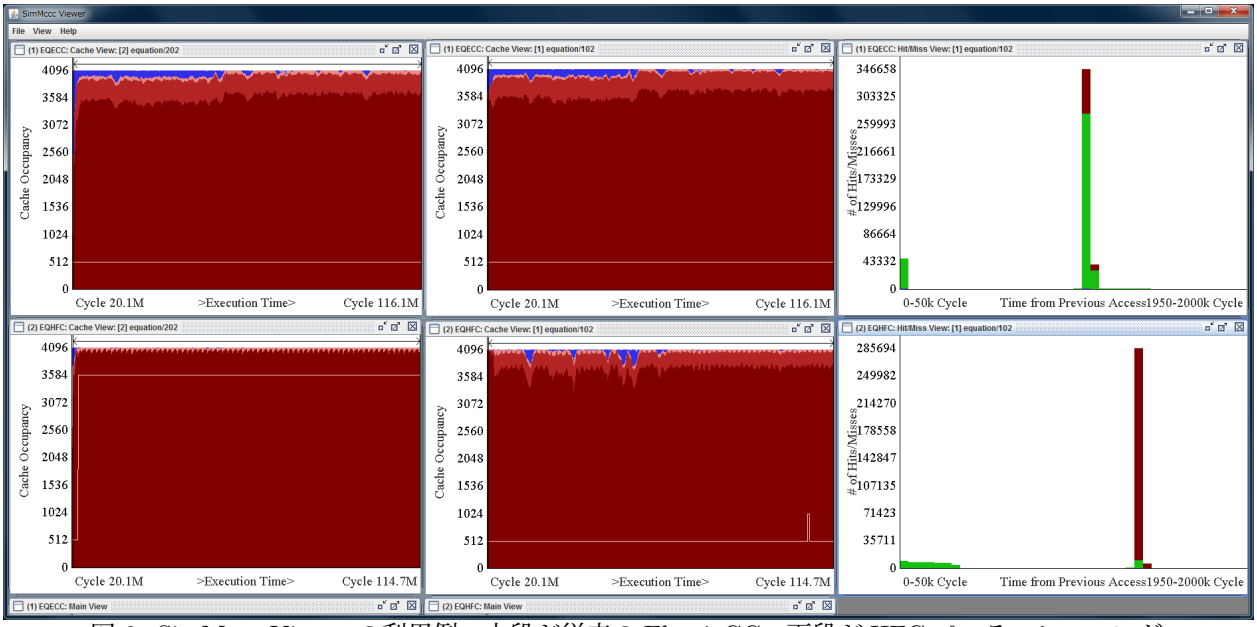

図 2: SimMccc Viewer の利用例．上段が従来の ElasticCC，下段が HFC パーティショニング．

を 160K 要素から 360K 要素に変更している．

これらの性能変化を測定したところ，HFC パーティショニングを用いたことでクイックソートの実行時間は 1.5%短縮されたものの，Equation Solver の実行時間が 24.8%も増加してしまった．その原因を SimMccc Viewer を用いて考察する．

図 2 にこの実行のログを SimMccc Viewer で可視化した時のスクリーンショットを示す．上段が従来方式の，下段が HFC パーティショニングを組み込んだ ElasticCC の可視化結果を表す．

左 2 つ，中央 2 つのグラフは Equation Solver のそれぞれ別のコアにおけるキャッシュの利用状況を表す．縦軸がライン数，横軸が実行時間である．グラフの大部分を占める濃い赤が自コアで利用したライン，中度の赤が他コアから受け入れた Equation Solver のライン，薄い赤が無効化されたラインであり，グラフ上部の青が他コアから受け入れたクイックソートのラインを表す．また，薄い線はその時点のパーティションの位置を意味する．

右 2 つのグラフは，中央 2 つのグラフと同じコアにおける，アクセス間隔とヒット・ミスの関係を表す．縦軸がアクセス回数，横軸が前回のアクセスからの経過時間である．薄い青が自コアでのヒット (このグラフではほとんど存在しない) を，中度の緑が他コアでのヒットを，濃い赤がミスを表す．

HFC パーティショニングの 1 つの利点は，右 2 つのグラフからわかるような周期的なキャッシュアクセスが支配的なベンチマークにおいても，正しくパーティションを設定できることである [3]．この実行においても Equation Solver の 4 コア中 3 コアは左下のグラフに示すように正しくパーティションを設定できており，他コアからのラインの移動を受け入れることによる悪影響を防いでいる．

しかしながら，この実行では中央下に示す 1 つのコアがその設定に失敗している．左上・中央上のグラフで示すように，従来手法では全てのコアが均等に悪影響を受けていたものが，HFC パーティショニングではこの 1 つのコアが必要とするラインだけが他コアからのラインによる悪影響を強く受けてしまい，キャッシュミスを大きく増加させてしまっている．このコアがボトルネックとなって，全体の性能が低下してしまったということが，図 2 から判断できる．

このように，SimMccc Viewer の可視化を利用することで性能変化の原因を素早く把握することができた．

5 まとめ

CMP 向け分散キャッシュにおいては，多様な要求に対して効率良く容量を利用することが重要である．本稿では，CMP 向け分散キャッシュ研究のツールとして可視化ツール SimMccc Viewer を提案し，その機能，設計，実装，利用例について述べた．今後は，より使いやすいツールを目指して SimMccc Viewer を改良するとともに，これを活用してより効率の良いキャッシュアーキテクチャを提案していく．

謝辞

本研究の一部は，科学技術振興機構・戦略的創造研究推進事業 (JST CREST) の「アーキテクチャと形式的検証の協調による超ディペンダブル VLSI」の支援による．

参考文献

1) Enric Herrero, et al. Elastic Cooperative Caching: An Autonomous Dynamically Adaptive Memory Hierarchy for Chip Multiprocessors. In *Proc. of the 37th ISCA*, pp. 419–428, 2010.

2) 植原昂ほか．メニーコアプロセッサの研究・教育を支援する実用的な基盤環境．電子情報通信学会システム開発論文特集号, pp. 2042–2057, 2010.

3) Naoki Fujieda, et al. A Partitioning Method of Cooperative Caching with Hit Frequency Counters for Many-Core Processors. In *Proc. of the 3rd UPDAS*, pp. 160–165, 2011.

4) D. E. Culler, A. Gupta, and J. P. Singh. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999.