

CoreSymphony における命令キャッシュの効率化

永塚 智之[†]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

CoreSymphony アーキテクチャ[1] は, CMP において逐次処理と並列処理のバランスを目的とするアーキテクチャ技術である. CoreSymphony は複数の発行幅の狭いコアを協調動作させ, 単一の発行幅の広い仮想コアを形成する. この仮想コアは単一のスーパースカラのように動作し, 逐次プログラムを高速に実行できる.

従来の CoreSymphony では, 各コアはローカル命令キャッシュ・従来型命令キャッシュと呼ばれる, 2 種類の命令キャッシュを持つ. これらの容量は設計時に決定され, 後で変更することはできない. アプリケーションにより, 2 つの命令キャッシュの内どちらがより多く使用されるかは異なる. そのため, 2 種類の命令キャッシュの容量比をどのように設定するかは重要な課題である.

本稿では, ローカル命令キャッシュと従来型命令キャッシュの共有化を行うことで, この課題の解決を目指す. 共有化により, 2 種類の命令キャッシュの容量比を実行時に動的に変更することが可能となる. シミュレータにより提案する命令キャッシュの性能を評価する.

2 従来の CoreSymphony の命令キャッシュ構成

CoreSymphony はフェッチブロック (FB) と呼ばれる命令トレース単位でフェッチ・ステアリングを行う. FB の最大長は 協調動作中のコア数 $\times$ 4 命令であり, FB は基本ブロックを最大 2 個まで含むことができる. ローカル命令キャッシュは, FB 中の自身にステアリングされた命令とその FB に関する各種制御情報を格納する, トレースキャッシュ[2] である.

図 1(1) に, 従来の CoreSymphony における命令フェッチの動作概略を示す. 各コアが自身にステアリングされた命令を独立してフェッチするためには, 2 つのフェーズを経る必要がある.

図 1(1-a) は, ローカル命令キャッシュのエントリ構築フェーズである. これは, ローカル命令キャッシュにミスした場合に行われる. 各コアは従来型命令キャッシュから FB 中の全命令を読み出し, デコード・ステアリングを行う. そして, ローカル命令キャッシュの当該エントリに, 自身にステアリングされた命令とその FB に関する各種制御情報を書き戻す.

図 1(1-b) は, 協調フェッチフェーズである. これは, ローカル命令キャッシュにヒットした場合に行われる. 各コアはローカル命令キャッシュから自身にステアリングされた命令のみをフェッチする. フェッチされた命令は各コアで独立にデコード・リネームが行われ, ディスパッチされる.

ローカル命令キャッシュは自コアにステアリングされた命令のみを格納するため, 命令キャッシュの実容量を協調により増加させることができる. 一方で, 従来型命令キャッシュは協調中の全コアでキャッシュの内容を共通とするため, 協調により実容量を増加させることができない. そのため, ローカル命令キャッシュの容量を従来型命令キャッシュと比べて大きくすることが望ましい. しかし, 4 章で示すように, 従来型命令キャッシュを大き

くした方が, より高い性能を得られる場合が存在する. そのため, これらの命令キャッシュの容量比をどのように設定するかは重要な課題である.

3 提案する CoreSymphony の命令キャッシュ構成

ローカル命令キャッシュと従来型命令キャッシュの容量比の課題を解決するため, 共有型ローカル命令キャッシュを提案する. これは, 2 種類の命令キャッシュのラインを単一の命令キャッシュ中に混在して格納する. 従来のローカル命令キャッシュのラインサイズは 250bit 程度である. そこで, このラインを 8 命令を含む従来型命令キャッシュのラインとしても利用できるようにする.

図 1(2) に, 提案する共有型ローカル命令キャッシュによる命令フェッチの動作概略を示す. 基本的な動作は従来のものと同様であるが, ローカル命令キャッシュのラインと従来型命令キャッシュのラインが, 同一のキャッシュから供給される. 両者を区別するために, 各ラインに 1 ビットのフィールド (line type) を追加する. フェッチ時にはタグとの比較に加えて, このビットを見ることで, キャッシュのヒット/ミス判定する.

ローカル命令キャッシュの共有化は, キャッシュのハードウェア複雑性に影響を与えない. なぜならば, キャッシュは従来と同様に, インデックスにより指定されたラインを供給すればよいからである. 変更する必要がある箇所は, キャッシュのヒット/ミス判定時に line type フィールドを加味する部分と, フェッチされたラインをどちらの命令キャッシュのラインとして解釈するかを決定する部分のみである.

今回の実装では, キャッシュミス時のラインリプレースの際には, 両者のラインを区別せずに, LRU で置き換えるラインを選択する. そのため, リプレースメント方式を工夫することで, 性能が向上する可能性がある. 例えば, 従来型命令キャッシュのラインを優先的に置き換えることが考えられる. ローカル命令キャッシュとしてのアクセスでヒットした場合には, 従来型命令キャッシュとしてのアクセスは行われない. そのため, 従来型命令キャッシュのラインが存在していても, そのラインは使用されないからである. 一方で, 従来型命令キャッシュのラインを残す方がよいとも考えられる. ローカル命令キャッシュと従来型命令キャッシュの間には, L1-\$ と L2-\$ のような, ある種のキャッシュ階層が存在する. そのため, ローカル命令キャッシュのラインが置き換えられたときに, 該当する従来型命令キャッシュのラインが残っている場合には, キャッシュミス時のレイテンシを低減できるからである. これらのことを考慮したキャッシュラインのリプレースメント方式の検討は, 今後の課題とする.

4 性能評価

シミュレーションにより評価を行った. シミュレータとして, ソフトウェアシミュレータ SimMips[3] を CoreSymphony のシミュレーション用に拡張したものを用いた. ベンチマークには, SPEC2006 ベンチマークから INT5 種, FP5 種を用いた.

評価結果を図 2 に示す. 図中の 1-core, 2-core fusion, 4-core fusion は, それぞれ 1 コア時, 2 コア協調時, 4 コア協調時を表す. 評価では, 以下の 4 通りについて IPC を測定した.

Efficient Instruction Cache for CoreSymphony Architecture
Tomoyuki NAGATSUKA[†], Kenji KISE[†]

[†]Graduate School of Information Science and Engineering,
Tokyo Institute of Technology

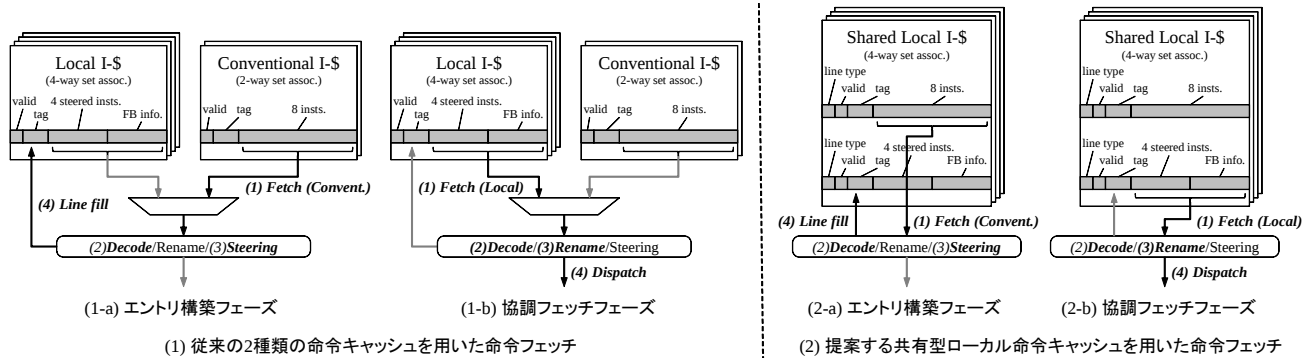


図 1: ローカル命令キャッシュのエントリ構築フェーズと協調フェッチフェーズ.

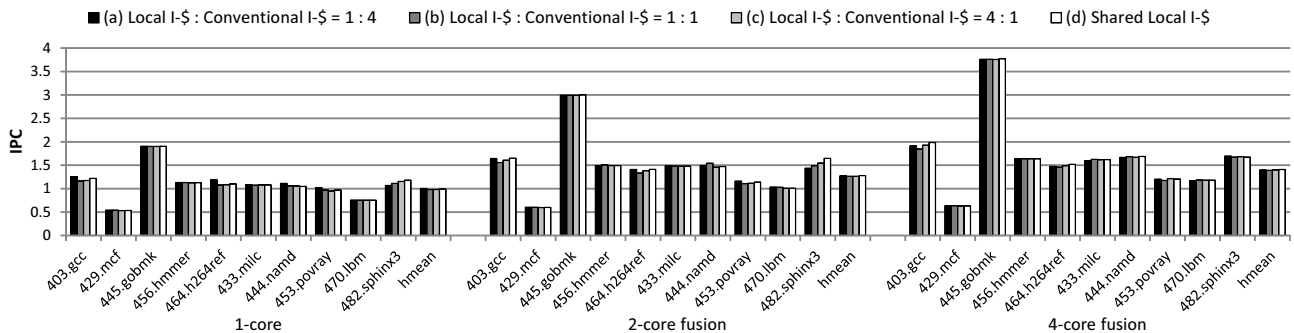


図 2: 評価結果.

- (a) ローカルと従来型の容量比が 1 : 4 の場合 .
 (b) ローカルと従来型の容量比が 1 : 1 の場合 .
 (c) ローカルと従来型の容量比が 4 : 1 の場合 .
 (d) 提案する共有型ローカル命令キャッシュを用いた場合 .
 全ての場合において、制御部を含む命令キャッシュの全体の容量が、コア当たり 30KB 前後となるように調整した .

結果を見ると、従来の命令キャッシュ構成では、ベンチマークにより最も高い性能を示す容量比が異なることがわかる . 例えば、1 コア時において、gcc、h264ref、namd、povray は (a) が最も高い性能を示すが、sphinx3 は (c) が最も高い性能を示す . これは、設計時に容量を固定した場合には、命令キャッシュ全体の容量を効率的に利用できていないためであると考えられる .

一方、提案手法は、特に gcc、sphinx3 において高い性能を示す . 従来の構成では、これらのベンチマークは命令キャッシュの容量比と性能において、逆の傾向を示していた . しかし、提案手法は両方のベンチマークで高い性能を達成した . これは、共有化によりローカル命令キャッシュと従来型命令キャッシュの容量比を実行時に動的に変更することが可能となり、命令キャッシュ全体の容量をより効率的に利用できたためであると考えられる .

また、提案手法は平均して高い性能を達成したことがわかる . 10 種のベンチマークの調和平均では、1 コア時には、(a) と比べると提案手法は 1.3% 低い IPC を示すものの、(b)、(c) と比較すると、それぞれ 0.82%、0.85% 高い IPC が得られた . 2 コア協調時には、(a)、(b)、(c) と比較して、それぞれ 0.43%、1.3%、1.3% 高い IPC が得られ、4 コア協調時には、それぞれ 0.77%、1.1%、0.46% 高い IPC が得られた .

以上から、提案する共有型ローカル命令キャッシュは、ローカル命令キャッシュと従来型命令キャッシュの容量比を実行時に動的に変更することを可能とし、様々なベンチマークで平均して高い性能を達成できることがわかる .

5 まとめと今後の課題

本稿では、CoreSymphony における 2 種類の命令キャッシュ (ローカル命令キャッシュ、従来型命令キャッシュ) を共有化した、共有型ローカル命令キャッシュを提案し、命令キャッシュの効率化を行った . シミュレーションによる評価の結果、提案する共有型ローカル命令キャッシュはベンチマークにより容量比を動的に変更し、高い性能を達成した .

今後の課題として、より効率的なリプレースメント方式の検討、従来のトレースキャッシュを用いたプロセッサへの提案手法の適用、などが挙げられる . 3 章で述べたように、リプレースメント方式を工夫することで、さらなる性能向上を達成できる可能性がある . また、従来のトレースキャッシュを用いたプロセッサは、CoreSymphony のように、2 種類の命令キャッシュ (トレースキャッシュ、従来型命令キャッシュ) を持つため、提案手法を適用して性能向上が得られる可能性がある .

謝辞

本研究の一部は科学研究費補助金 (課題番号:22700046 若手研究 (B)) による .

参考文献

- [1] 若杉祐太, 坂口嘉一, 吉瀬謙二. 協調可能スーパースカラ CoreSymphony. 情報処理学会論文誌 ACS, Vol.3, No.3, pp. 67–87, 2010.
- [2] Eric Rotenberg, Steve Bennett, and James E. Smith. Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching. In *Proceedings of the 29th International Symposium on Microarchitecture (MICRO-1996)*, pp. 24–35, 1996.
- [3] 藤枝直輝, 渡邊伸平, 吉瀬謙二. 教育・研究に有用な MIPS システムシミュレータ SimMips. 情報処理学会論文誌, Vol.50, No.11, pp. 2665–2676, 2009.