CMP の逐次性能向上を目指す CoreSymphony アーキテクチャ

若杉 祐太 †　藤枝 直輝 †　坂口 嘉一 †　三好 健文 †,‡　吉瀬 謙二 †

東京工業大学 †　独立行政法人 科学技術振興機構 ‡

1 はじめに

半導体技術の持続的な進歩により，CMP(Chip Multi-Processor) における 1 チップあたりのコア数が増加し続けている．これに伴い，近い将来に次の問題が顕在化すると考えられている．それは，プログラム中に存在する並列化不可能な処理 (逐次処理) が CMP の性能を制限するという問題 [1] である．現状では，並列プログラム中の逐次処理をなくすことは難しい．CMP においても逐次処理の高速化が依然重要な課題であるといえる．

CMP の逐次性能を向上するアプローチとして，複数コアの融合により，逐次実行能力の高い仮想コアを作り出すアーキテクチャ技術 [2] が存在する．我々が提案する CoreSymphony アーキテクチャ [3] もそのひとつである．CoreSymphony は 2 命令発行のアウトオブオーダをベースとし，最大で 4 コアの協調動作をハードウェアでサポートする．4 コアの協調時には 8 命令発行の仮想コアを形成する．仮想コア上では従来のプログラムがネイティブで動作し，並列プログラム中の逐次処理の高速化を強力に支援する．図 1(左) に CoreSymphony を実装した CMP の構成を示す．協調動作のためのハードウェアを付与した 4 つのコアと共有 L2 キャッシュを持つ．協調動作のためのネットワークによりコア同士を接続する．図 1(右) は 2 コアの協調動作時の様子である．

本稿では，本研究の最も挑戦的な部分である協調可能フロントエンドの実装について述べ，重要な要素技術である 2-way リネーミングを提案する．

2 CoreSymphony アーキテクチャの実装

図 2 に CoreSymphony のコアのブロック図を示す．ベースのアーキテクチャはバックエンドにてレジスタ読み出しを行う方式のアウトオブオーダである．協調動作の実現のために我々が追加/大きな変更を加えたハードウェアを影付きで示している．次節以降では，フロントエンドの各ステージごとに CoreSymphony の動作を要点に絞って述べる．

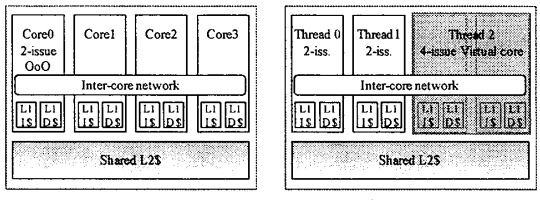

図 1: CoreSymphony アーキテクチャを実装した CMP．4 スレッド実行可能な構成（左）．協調動作により，あるスレッドに多くの演算資源を割り当てる場合の構成例の一つ（右）．

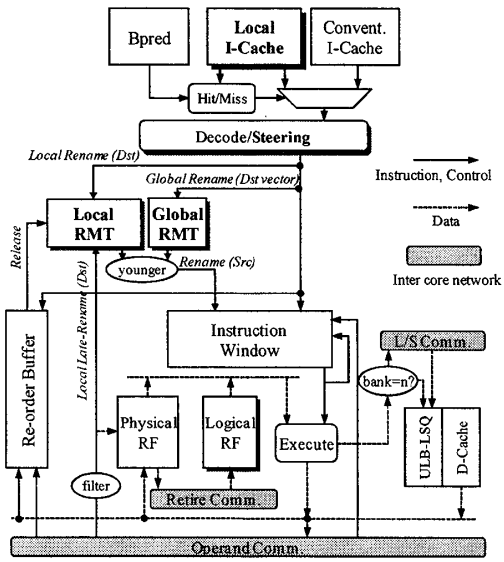

図 2: CoreSymphony アーキテクチャのコアのブロック図．

2.1 命令フェッチ

CoreSymphony では，協調動作中のコア数× 4 命令を最大長とする命令トレースを 1 単位として，フェッチ/ステアリングをおこなう．この命令トレースをフェッチブロック (FB) と呼ぶ．命令フェッチの分散はローカル命令キャッシュ [3] によって実現する．ローカル命令キャッシュは FB 中の自コアにステアリングされた命令および，FB 間の依存関係の解決に用いる制御情報を格納するトレースキャッシュである．命令本体は各コアのローカル命令キャッシュに分散して格納されるため，このキャッシュにヒットしている間は，協調動作中のコア数× 2 のフェッチ幅が得られる．制御情報には **Destination Vector**(以下 Dst.vec) と呼ぶ，FB 中の命令が結果を格納する論理レジスタの位置を記した，論理レジスタ数と同長のビット列が含まれる．

2.2 命令ステアリング

ローカル命令キャッシュにミスした場合は，各コアは FB 中の全命令を従来型命令キャッシュから得る．そして，次ステージで命令のステアリングをおこなう．命令ステアリングにおける鍵は，依存関係にある命令を同じコアに割り当てること，各コアの負荷を均等にすることである．前者は Wakeup レイテンシに，後者は Select レイテンシに直結するため，ステアリングアルゴリズムは

CoreSymphony: Boosting Sequential Performance in CMPs
Yuhta Wakasugi†, Naoki Fujieda†, Yoshito Sakaguchi†, Takefumi Miyoshi†,‡, and Kenji Kise‡
†Tokyo Institute of Technology
‡CREST, Japan Science and Technology Agency

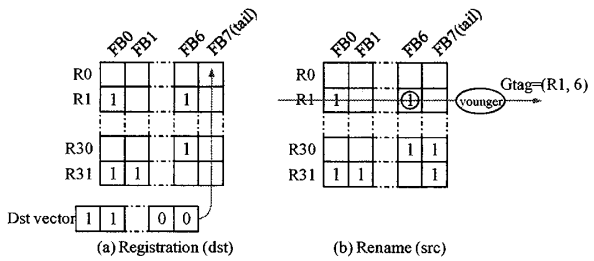

図 3: GRMT の動作.

性能を大きく左右する．我々が提案するリーフノードステアリング[3] は，複数の命令の依存元となる命令を重複してステアリングすることで，これら 2 つの要求をバランスよく満たすことができる．

2.3 レジスタリネーミング:2-way リネーミングの提案

リネームステージの役割は，(1) 論理レジスタ番号から物理レジスタ番号への対応を与えること．(2) ある命令が依存する命令を表すタグを与えることの 2 つである．一般的なアウトオブオーダではタグ＝物理レジスタ番号であるため，これら 2 つは同義である．ここで，(2) のタグはその性質上グローバルな管理を必要とする．これにより，アウトオブオーダにおける RMT(Register Map Table) は集中化の必要が生じ，8-way のスーパースカラではポート数は 32 にもなる．しかし，(1) と (2) の役割は独立しているため，必ずしもタグ＝物理レジスタ番号でなくともよい．本稿では，この観察に基づく RMT の分散手法 **2-way** リネーミングを提案する．

2-way リネーミングの本質は，コア内の依存とコア間の依存を異なるタグで表現し，別々のテーブルで管理することにある．ここで，コア内の依存を管理するタグ/テーブルを Local tag(Ltag)/Local RMT(LRMT)，コア間の依存を管理するタグ/テーブルを Global tag(Gtag)/Global RMT(GRMT) と呼ぶ．LRMT は通常の RMT に相当するハードウェアである．よって，LRMT のポート数は 2-way アウトオブオーダの RMT に準じる．ただし，各エントリを { 物理レジスタ番号，生産者の FB 番号 } に拡張する．ここで，FB 番号とは in-flight な FB にサイクリックに割り当てられる 4 ビット程度のタグである．

一方，GRMT の構成は少々特殊である．ISA が規定する論理レジスタの本数を N，パイプライン中に保持可能な最大 FB 数を M とすると，GRMT は $N \times M$ のビットマトリックスとして構成される．x 行 y 列のビットは，FBy 中に論理レジスタ Rx をデスティネーションとする命令が存在するか否かを表す．

GRMT によるリネームは通常の RMT 同様，(a) デスティネーションの登録，(b) ソースのリネームの 2 ステップに分けられる．図 3 に GRMT の各ステップの動作を示す．(a) では各 FB に付随する Dst.vec を GRMT の tail 列に書き込む．Dst.vec は各コアにコピーが保存されているため，GRMT は全コアで同一に保たれる．(b) では LRMT と同様に論理レジスタ番号で GRMT の行を読みだす．GRMT の行 x は，論理レジスタ Rx に値を書き込む命令が存在する FB 番号を与える．ここから図 3 中 younger で示すロジックにより，最も若い FB 番号 y を求める．結果，Gtag=$\{x, y\}$ としてタグ付けが完了する．

Conv. I-$	8KB, 2way, 1cycle
Local I-$	16KB, 4way, 2cycle
D-$	16KB, 2way, 1cycle
L2-$	2MB, 4way, 10cycle
Memory	100 cycle
Bpred	Bimode, 1k entry
BTB	1k entry, 2way
Inst win	24 entry
ROB	Max 6FBs
LSQ	80 entry
PRF	INT 48 / FP 48
FU	2 INT, 1 FP

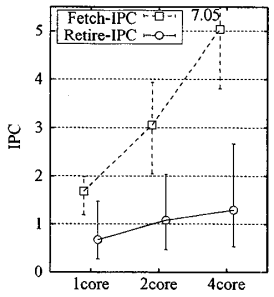

図 4: 評価のパラメタ (左) と評価結果 (右).

GRMT と LRMT の読み出しは同時におこなう．Ltag と Gtag の FB 番号を比較し，FB 番号が若い方を最終的にスケジューリングに用いるタグとして採用する．

3 CoreSymphony アーキテクチャの評価

実行駆動のサイクルレベルシミュレータにより，CoreSymphony の性能を評価する．評価のパラメタを図 4(左) に示す．ベンチマークは SPEC2006 より INT5 種類 (gcc,mcf,hmmer,libquantum,h264ref)，FP5 種類 (milc,namd,povray,lbm,sphinx3) を用いる．データセットは train，1G 命令スキップ後の 100M 命令を利用する．

協調動作するコア数を変化させた場合の，フェッチ IPC とリタイア IPC(実 IPC) の全ベンチマークにおける最大，最少，調和平均を図 4(右) に示す．2 章で示したフロントエンドアーキテクチャにより，命令フェッチは良いスケーリングが得られている．実 IPC についても 4 コアの協調により，1 コア時と比較して平均で 1.91 倍高い値が得られた．

4 まとめと今後の課題

本稿では，複数コアの協調により CMP の逐次性能を向上する CoreSymphony アーキテクチャを提案し，そのフロントエンドの実装について述べた．評価の結果，協調動作によりフロントエンドのスループットがスケールすることを確認し，性能が向上することを確認した．

今後はハードウェア規模を含めたより詳細な評価をおこなう予定である．

参考文献

[1] M. D. Hill, et al. Amdahl's Law in the Multicore Era. *IEEE Computer*, vol.41(7), pp.33-38 (2008).

[2] E. Ipek, et al. Core Fusion: Accommodating Software Diversity in Chip Multiprocessors. In *Proc. of ISCA-34*, pp.186-197 (2007).

[3] 若杉他. CoreSymphony アーキテクチャの高効率化. 情処研報 2009-ARC-184, pp.1-12 (2009).