マルチコア・プロセッサにおける 単純で合理的なコア・フュージョン機構の実装

若杉 祐太 † 吉瀬 謙二 ‡

東京工業大学 工学部情報工学科 † 東京工業大学大学院 情報理工学研究科 ‡

1 はじめに

半導体技術の持続的な進歩により，プロセッサの 1 チップ上に集積可能なトランジスタ数は年々増加し続けている．近年のプロセッサ・アーキテクチャでは，この増加するトランジスタを 1 チップ上に複数のコアを集積することに利用する動きが高まっている．これは，ゲート遅延に対する相対的な配線遅延の増加等の問題から，単一コアの性能向上が困難になったためである．今後は，より多く (数十〜数百) のコアを集積したメニーコア・プロセッサの時代へ向かうものと考えられる．

しかし，多くのコアを無駄なく使うだけのスレッドレベル並列性を維持することは難しく，シングルスレッド性能が重要になる場面も依然として多く存在する．

このような問題に対応すべく，我々はコア同士が融合 (fuse)/分離 (split) することでスレッドあたりの実行能力及び，実行可能スレッド数を動的に変化させる，コア・フュージョン機構の実現を目指す．具体的にはコア同士の融合により命令の発行幅を増加させる (図 1)．これにより，実行スレッド数が少ない場合にはコアが融合して各スレッドを高速に実行し，実行スレッド数が多い場合には分離して並列処理，という様にタスクの粒度に柔軟に対応することが可能となる．また，スレッド毎に負荷が異なる場合にも，高負荷のスレッドを多くのコアを使用して高速化する事ができる．

本稿では，この仕組みを実現する，単純で合理的なコア・フュージョン機構を提案する．そして，提案機構を C 言語のサイクルレベル・シミュレータのレベルで実装し，簡単なアプリケーションにより性能を評価する．

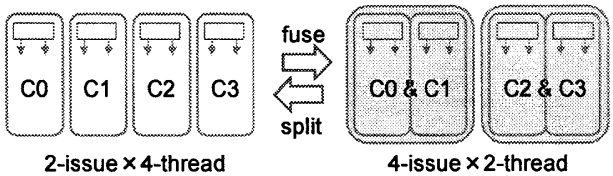

図 1: 融合による発行幅の増加および，分離による発行幅の減少．4 個の 2 命令同時発行のスーパスカラ (左) が 2 コアずつの融合により，あたかも 2 個の 4 命令発行スーパスカラ (右) であるかのように振舞う様子を示している．本実装では 4 コア全ての融合により 8 命令発行とすることも可能である．

Implementation of a simple and reasonable core-fusion mechanism for multi-core processors.
Yuhta WAKASUGI†, Kenji KISE‡
†Depart. of Computer Science, Tokyo Institute of Technology
‡Graduate School of Information Science and Engineering, Tokyo Institute of Technology

2 単純で合理的なコア・フュージョン機構の提案と実装

2.1 提案機構の設計方針と全体構成

各コアの協調により命令発行幅を動的に変化させるアーキテクチャとしては [1, 2] が提案されているが，本稿で示すコア・フュージョン機構とコンセプトを異にする部分も多い．以下に，提案機構の設計方針を示す．

1. 既存のプロセッサからの少ない変更
   本機構は既存のアウトオブオーダ・スーパスカラに非常に近いデータパスとなっており，命令セットも従来のものを使用できる ([2] は独自 ISA)．これにより実装及び従来技術の転用が容易になると考えられる．
2. 一つ一つのコアの独立性が高い実装
   本機構はコア間にまたがる集中的な制御モジュールを必要としない ([1] は多くの集中的なモジュールを要する)．また，コア間の通信に必要なバスは本質的に，オペランドフォワーディングに用いる 1 本のみである．これにより，柔軟な融合が可能となる (フォワーディングのパスさえあればどのコアとも融合可能)．

2-issue アウトオブオーダ・スーパスカラに本機構を取り入れると，パイプラインの全体像は図 2(a) のようになる．従来のプロセッサからの変更点はステアリング部 (命令をどのコアに割り振るかを決定する) と命令フェッチ部，及びコア間通信を含む結果書き込み部のみと少ない．

以降，従来のプロセッサからの変更点を中心に提案するアーキテクチャについて述べる．

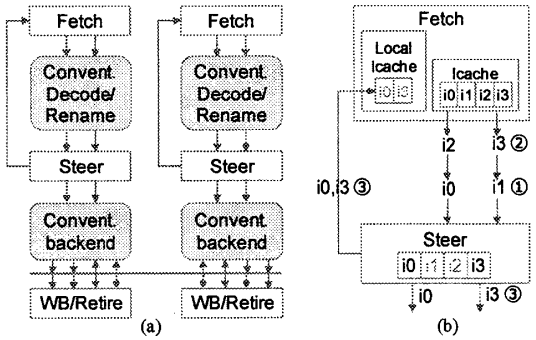

図 2: (a) 提案機構を取り入れたパイプライン概観 (2 コアの融合時)．(b) 変更されたフロントエンド．フェッチ・ブロック (i0,i1,i2,i3) 中の，自身にステアリングされた (i0,i3) をローカル命令キャッシュに書き戻す様子を示す．

2.2 フロントエンドの協調

集中化した制御モジュールを用いずに，命令キャッシュに代表されるフロントエンドの資源を融合によって線形に増加させる事は挑戦的な課題である [2]．本研究ではこ

れを実現するために，ローカル命令キャッシュと呼ぶ，自身にステアリングされた命令のみを保存するキャッシュを提案する．これを用いたフロントエンドを図 2(b) に示す．基本方針は次の通りである．まず，命令は融合コア数 ×2 命令毎のフェッチ・ブロックに区切って実行する．あるフェッチ・ブロックを初めて実行する際には，小容量の従来型命令キャッシュからブロック内の命令を 2 サイクル (2 コア融合の場合) かけて，全てフェッチ/デコードする．その後，ブロック中の自身にステアリングされた 2 命令のみを，ローカル命令キャッシュに保存する．以降同様のブロックをフェッチする際にはこちらからフェッチを行う．ローカル命令キャッシュをメインに考え，従来型命令キャッシュを小さく作ることで，融合により命令キャッシュ全体の実効エントリ数をほぼ線形に増加させることができる．

しかしながら，自身に割り当てられた命令のみでは依存関係の解決ができない．このため，ローカル命令キャッシュには他コアに割り当てられた命令のデスティネーション・レジスタを同時に保存する．つまり図 2 の例では (i0,i3,rd1,rd2) を保存する．これにより，全てのコアで共通のリネーミングが可能になる．

2.3 バックエンドの協調

現在の実装では，あるコアの実行ステージで得られた結果は，結果通信バスを用いて融合中の全てのコアにブロードキャストされる．そして，コア間通信に要するサイクル数を待ち合わせた後，融合中の他コアから得られた結果と共に全コアで一斉にライトバックを行う．つまり，任意のタイミングで全コアは同じアーキテクチャ・ステートを持つという保守的な実装となっている．このため，ライトバック，リタイアステージは 1 コア当たり最大で $2 \times 4 = 8$ 命令幅を必要とする．

コア間のオペランド・フォワーディングは次のように行われる．例として，ある命令 i1 が命令 i0 に依存していたとする．依存先の i0 が i1 と同じコアにある場合，通常のプロセッサと同様，i0 → i1 と実行することができる．しかし，i0 が別のコアにある場合，先述した結果通信バスを用いて i0 の結果がブロードキャストされてくるのを待たなければならない．これにより実行パスは，

i0 → (コア間レイテンシ L) →ウェイクアップ→ i1

となり，コア内フォワーディングに比べ $L+1$ サイクルを余分に必要とする．このコア間フォワーディングが，通常のスーパスカラと比べ，本質的に性能を制限する要因となる．そのため，クリティカルパス上でのコア間フォワーディングの発生を抑えることが重要となる．

3 シミュレータによる性能評価

MIPS 命令セットの独自に開発したサイクルレベル・シミュレータにより提案機構を実装し，性能を評価する．表 1 に本シミュレータが模倣するプロセッサの仕様を示す．

現段階では，ロードストア命令の実装を見送っている．これは，本機構のコンセプトである，独立性と簡潔性を保持しながら，データキャッシュを分散させるのはフロントエンド同様，非常に挑戦的な課題であり，検討すべき点が多く残されているためである．

Param.	Value	Param.	Value
#BTB ent.	64/core	#FU	2Int/core
#ROB ent.	40/core	#RS ent.	6/FU
pipeline stage	IF,ID,STeer,OF,RS,EX,WB,RT		
branch predictor	Bimodal		
inter core latency	1 cycle		
steering algorithm	modulo-2, dep-based		

表 1: 想定するプロセッサ・モデル．比較対象のアウトオブオーダ・スーパスカラも基本的にこれを踏襲する．

評価にはアセンブリ言語で記述したレジスタ・バブルソート (bsort) とユークリッドの互除法 (euclid) を用いる．また，典型的なスーパスカラと性能を比較する．

シミュレータによる IPC の測定結果を図 3 に示す．euclid では 4 コアの融合により，単一コア時に比べ 133% の IPC 向上を達成する．また，ベースのアウトオブオーダと比べても IPC の低下は 9%に留まる．しかし bsort では融合による IPC の向上は 9%と小さい．これは bsort は基本ブロック内の依存が強く，コア間フォワーディングが頻発したためと考えられる．

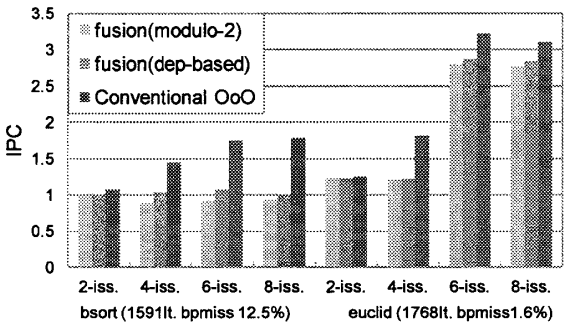

図 3: 発行幅 vs. IPC

4 まとめ

本稿では，マルチコア・プロセッサ向けの技術であるコア・フュージョン機構の第一歩の実装について述べた．また，簡単なアプリケーションで評価を行い，コアの融合により IPC が最大で 133%向上することを確認した．

今後の課題としては，合理的なロード/ストアの実装方法の検討，よりマルチコア向けの実装方法の検討，コストも含めた，より詳細な分析が挙げられる．

謝辞

本研究の一部は，科学研究費補助金若手研究 (B) 課題番号 18700042「投機技術を積極的に利用するチップマルチプロセッサに関する研究」の援助による．

参考文献

[1] Engin ipek, et al. Core Fusion: Accommodating Software Diversity in Chip Multiprocessors. In Proc. ISCA-34, 2007.

[2] Changkyu Kim, et al. Composable Lightweight Processors. In Proc. MICRO-40, 2007.