C-001

コア融合アーキテクチャにおける最適コア割り当ての研究

A Study of Optimum Core Allocation for Uniting Core Architecture

山田裕介† 小林良太郎† 若杉佑太‡ 吉瀬謙二‡
Yusuke Yamada Ryotaro Kobayashi Yuhta Wakasugi Kenji Kise

1. はじめに

近年のプロセッサアーキテクチャの主流として、1 チップ上に複数のコアを搭載するマルチコアがあり、2010 年以降、数十、数百のコアが 1 チップ上に搭載されるようになると言われている。

これはゲート遅延に対する配線遅延の増加などの理由から、単一コアに投入されるハードウェアコストに対して見合うだけの性能向上が得られなくなったことと、プロセッサの処理能力を単一スレッドの処理にだけでなく、複数のスレッドを同時に処理することに焦点を当てて設計するようになったためである。

しかし一方で、数十ものコアを無駄なく使うだけのスレッドレベル並列性(TLP)を維持することは難しく、加えて、単一スレッドの処理能力もいまだに重要であることに変わりはない。このことから、搭載される多くのコアを効果的に使用するためには、実行されるタスクの粒度に対して柔軟に対応する必要がある[1][2]。

コア融合アーキテクチャはこれに対する解決策の一つとして提案されている[3][4]。コア融合アーキテクチャでは、マルチコア上の複数のコアが協調動作してあたかも一つの大きなコアのように振舞う。これによりチップ全体での TLP 能力は下がるが、実行される単一スレッドの処理を早めることができる。

しかし、コアの融合におけるスレッドの処理性能向上の度合いは命令レベル並列性(ILP)等のスレッドのもつ特性に左右され、場合によってはスレッドに投入したコアに対して十分な処理能力の向上が得られないことも考えられる。そこで本研究ではスレッドに対するコアの投入において、その投入されるコア数の最適化手法を考える。これによりコアの融合において無駄なコアの割り当てを省き、チップ全体での効率的なプログラム実行を目指す。

本論文において、第 2 章では本研究で利用するコア融合技術 CoreSymphony[4]アーキテクチャの概略を簡単に述べる。次に第 3 章で提案手法について説明する。そして第 4 章で提案手法の評価を行い、第 5 章で本稿をまとめる。

2. CoreSymphony の概略

本研究は若杉らが提案するコア融合アーキテクチャ CoreSymphony を基にして行う。

† 豊橋技術科学大学大学院工学研究科
Graduate School of Engineering, Toyohashi University of Technology

‡ 東京工業大学大学院情報理工学研究科

CoreSymphony アーキテクチャでは 2 命令発行のアウトオブオーダプロセッサコアを多数有するメニーコアプロセッサを想定している。各コアは一つずつルータを所持しており、各コアはルータ同士がメッシュ状に四方のコアの持つルータと結合してネットワークを構成している。

コアの融合では最大 4 つのコアが融合して 8 命令発行可能なコアを仮想的に構築することが可能である。

CoreSymphony は実現する上で、高性能、低消費電力、高ディペンダリティを目指し、その上で、次の 3 つの基本方針を示している。

2.1 基本方針

CoreSymphony はシンプルな結合網により接続された均質なコアを多数持つプロセッサへの適用を考えている。この時、融合対象のコアや融合の形態にはいくらかの柔軟性を有することに配慮する。そのためコア間通信を必要とするモジュールや、各コアの情報を収集し集中的に制御を行うモジュールを数多く持つことはできる限り避け、制御を各コアに分散し少ない通信による協調動作を目指す。これにより、制御情報などを各コアで多重化する必要が生じてしまい実行効率および面積効率でいくらかの不利を被ると考えられるが、これらの不利を現実的な範囲に収めながら各コアのステータス制御が独立に行われることを重要な課題としている。

またコアの融合を実現するためには、従来のプロセッサコアに変更を加える必要があるが、CoreSymphony ではこの変更をなるべく小さいものに留めている。これにより従来のアーキテクチャ技術の流用が可能となり、実装が容易になる。またコアの複雑化に伴う配線遅延の増加の抑制をしている。

CoreSymphony では専用の命令セットを用いず従来型の RISC 命令セットを採用し、融合の実現を目指している。これにより既存のコンパイラ最適化技術の流用やバイナリの連続性を維持することを可能とする。

2.2 アーキテクチャ

CoreSymphony では融合時に各コアの分岐予測器、分岐先アドレス、リオーダバッファ、各レジスタなどはコア間通信の簡略化のため、多重化して使用されている。また、CoreSymphony の特徴の一つとして各コアは従来の命令キャッシュとは別に、ローカル命令キャッシュという特別なキャッシュを持つ。ローカル命令キャッシュはコアが融合した際に、各コアに命令を割り振った結果を格納する。格納される情報はそのコアで実行される命令とこれと並列に他コアで実行される命令のデスティネーションレジスタである。これによって、融合コア数が増えるほどその融合したコア全体で格納される命令量は増大する利点がある。

また CoreSymphony ではコミット処理時に正しく各コアが同じ制御フローを辿ることを保つための機構を従来の機構に対して付加している。

3. 提案手法

本論文では効率のよいコアの割り当てを実現するためにコアの融合によって得られる性能の向上を予測する手法を提案する。コアの融合によって得られるスレッドの処理性能の向上はスレッドの性質に依存するため、これは明示的に得ることはできない。そこで本提案手法では実行中のスレッドの命令間の依存性とその依存関係にある命令を実行するコア間のネットワーク上の距離に着目し予測を行う。これは融合によるスレッドの実行は複数のコアにまたがって行われるため、命令間に依存がある場合には通常のストールに加えて命令結果の送信レイテンシの発生によって、性能低下の重大な要因になると考えられるからである。

提案手法の実装において、命令間の依存の検出はリオーダバッファでコミットされた命令を対象にしている。そして検出した際にその命令依存によって生じるコア間のレイテンシをカウントすることで予測を行う。予測は実際に融合したコア数に関わらず、全ての融合コア数の場合において行われる。

また提案手法はコア内でモジュールとして実装されることを想定している。最適化の手法としてプログラムの実行の前段階でのコンパイラによる情報の埋め込み等のソフトウェアでの実装が考えられるが、この場合には実行されるソフトウェアを制限してしまうという問題や，最適コア数が扱うデータセットに依存する場合に対応できないという問題が発生する可能性がある。

4. 評価

サイクルレベルのシミュレータに５つの簡単なプログラムを実行させ、提案手法を用いて性能の向上を推定した。評価の際にシミュレーションしたコアのパラメータを表 1 に示す。次にプログラムについて説明する。BubbleSort は 20 本のレジスタの内容に対してバブルソートを行うプログラムで、Euclid は ユークリッド互除法による最大公約数の計算を行っている。Dataflow では命令間に依存の多い演算を行っている。VectorSum は 24 個のレジスタにそれぞれある固定値を加算する。最後に Brunch では VectorSum に 3/8 の割合で分岐ミスが生じるコードを挿入したプログラムである。

シミュレーション結果を図 1 に示す。各棒グラフはコアがスレッド処理に投入された場合のその性能向上比を示す。左側の棒グラフがシミュレータから得られた IPC による値で、右側の棒グラフが提案手法によって推定された値である。この二つの大きさが近いほど、推定の精度が高いと評価することができる。図 1 の結果より実際の値と推定値の差が 0.2 以上あるものはほとんどないが、一部では大きく差が出てしまっていることがわかる。

結果から ILP による制約とコア間のレイテンシによる性能低下が Core Symphony アーキテクチャにおけるコア融合における性能の向上に対して相関があると考えられる。しかし推定が失敗している部分もある。これは特に分岐ミスが多く生じる Brunch や BubbleSort において多い。これより分岐ミスによる強調動作の低下が推定値の精度に影響していると思われる。また VectorSum の 1 コアと 2 コアの場合においても推定値との間に大きな差ができてしまっており、この原因も解明する必要がある。

表 1：評価に用いたコアのパラメータ

Parameter	Value
Fetch issue width	2/core
Retire width	6
Functional Units	2×Integer/core
Reservation Station	8entry/ALU
Re-order Buffer	64 entry, 128 entry
Conventional I-cache	64 Instruction (256B), direct map
Local I-cache	64 Instruction (420B), 2way
Brunch prediction	Bimodal(64 entry)
BTB	64 entry
Steering strategy	Modulo-2
Network Latency	1 cycle/hop
Routing Algorithm	dimension-order routing

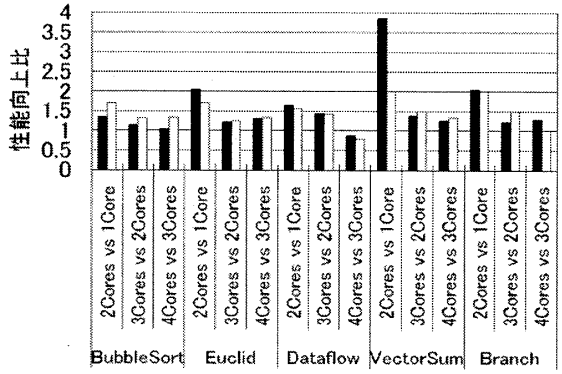

図 1：実際の IPC と推定値の比較

5. おわりに

本稿ではコア融合の概念とこれを取り扱うコア融合アーキテクチャ CoreSymphony について説明し、この技術を効率的に利用するための手法を提案した。そしてシミュレータを用いて提案手法の評価を行った。今後の課題として推定精度を高めるために分岐ミスによるペナルティを含めた手法等を提案する必要があると考えている。

謝辞

本研究の一部は文部科学省科学研究費補助金若手研究(B)課題番号 21700057、柏森情報科学振興財団研究助成、及び、中山隼雄科学技術文化財団研究助成の支援により行った。

参考文献

[1] 笹田耕一ほか，“メニーコアプロセッサ時代を拓くシステムソフトウェアへの挑戦，” 情報処理学会研究報告 2008-OS-108, pp.67-74, 2008.

[2] 小林良太郎ほか，“多機能メニーコアを実現するアーキテクチャ技術 Feature-Packing の構想，” 情報処理学会研究報告 2007-ARC-175, pp.11-15, 2007.

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

[4] 若杉祐太ほか，“メニーコアプロセッサに向けたシンプルで柔軟なコア融合機構 CoreSymphony，” 先進的計算基盤システムシンポジウム SACSIS2008, pp.421-430, 2008.