5K-7

効率的なCoreSymphonyアーキテクチャの実装

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シングを目的とするアーキテクチャ技術である．CoreSymphony は最大 4 つの 2 命令発行のアウトオブオーダコアを協調動作させ，8 命令発行の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．図 1 に CoreSymphony の概念図を示す．

CoreSymphony を実現するためには，ベースとなる 2 命令発行のアウトオブオーダコアに変更を加える必要がある．これまで CoreSymphony の評価はソフトウェアシミュレータを用いて行われていため，ハードウェア量や動作周波数といった項目について正確な評価が行われていなかった．本稿では，CoreSymphony を Verilog HDL で実装し，ハードウェア量と性能を評価する．

2　CoreSymphony の構成

2.1　命令フェッチユニット

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 × 4 命令であり，FB は分岐命令を最大 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である．

CoreSymphony の命令フェッチは，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に分けられる．ローカル命令キャッシュミス時にはエントリ構築フェーズに遷移する．各コアは従来型命令キャッシュから FB をフェッチし，デコード・ステアリングを行う．自コアにステアリングされた命令と各種制御情報をローカル命令キャッシュへ書き込む．ローカル命令キャッシュヒット時には協調フェッチフェーズに遷移する．各コアは自コアにステアリングされた命令のみをフェッチし，実行する．

2.2　ステアリングユニット

命令ステアリングとは，各コアでどの命令を実行するかを決定する操作である．CoreSymphony はステアリングのためのハードウェアを持ち，実行時にステアリングを行う．ステアリングアルゴリズムとして，リーフノードステアリング [2] を採用する．

2.3　レジスタマップテーブル

レジスタリネーミングに使用されるレジスタマップテーブル（RMT）のポート数は，命令のフェッチ幅に依存して増加する．CoreSymphony では 4 コア協調時には 1 サイクル当たり最大 8 命令をフェッチするため，RMT に非常に多くのポートを必要としてしまう．

CoreSymphony では，2-way リネーミング [1] という手法を用いることでこの問題を解決する．コア内の依存関係とコア間の依存関係を別のテーブルを用いて管理することで，RMT のハードウェア量を削減する．コア内の依存を管理するテーブルを Local RMT（LRMT），コア間の依存を管理するテーブルを Global RMT（GRMT）と呼ぶ．

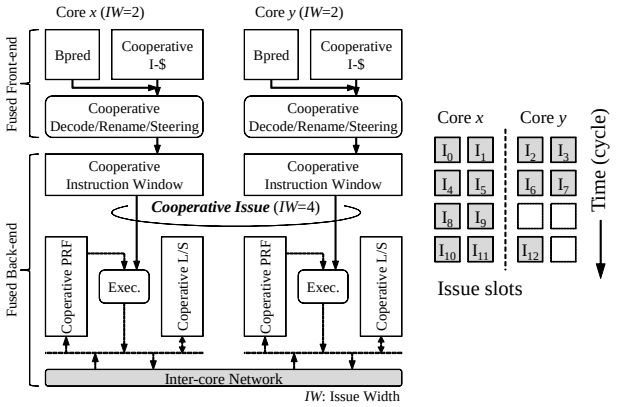

図 1: CoreSymphony の概念図（2 コア協調時）．

2.4　物理レジスタファイル・命令ウィンドウ

ディスパッチされた命令は命令ウィンドウに格納される．オペランドが揃った命令から順次発行され，物理レジスタファイル（PRF）からオペランドをフェッチし，実行する．一般的な 2 オペランド方式の RISC では，1 サイクルに最大 2 命令発行するためには，PRF に 4 つの読み込みポートを必要とする．また，4 コア協調時に実行結果を全コアで重複して保存するには，4 つのコアがそれぞれ 2 つずつ結果を生成するため，PRF に 8 つの書き込みポートを必要とする．結果，4R8W の PRF を必要とする．また，協調により in-flight な命令が増加するため，物理レジスタ枯渇によるパイプラインストールが発生して，性能に悪影響を与える．そのため，協調により物理レジスタの実効エントリ数を増加させられることが望ましい．

CoreSymphony では，PRF の分散管理 [1] を行うことでこの問題を解決する．これにより PRF を 6R3W で実現することが可能となる．また，各コアの PRF が自コアで使用される結果のみを格納することになるため，協調により物理レジスタの実効エントリ数を増加させることができる．その一方で命令ウィンドウに CAM を追加する必要があるため，命令ウィンドウのハードウェア量が増加する．

2.5　ロードストアユニット

CoreSymphony では協調によりロードストアキュー（LSQ）やデータキャッシュの容量を増加させることを目指す．この課題に対し各コアの LSQ・データキャッシュを実効アドレスで分けられたバンクとみなすアプローチが考えられるが，ロードストア命令がアクセスするメモリアドレスが決定されるのは命令実行後である．そのため，ステアリング時にロードストア命令をディスパッチするコアを決定することは不可能である．

CoreSymphony では Unordered Late-Binding Load/Store Queue（ULB-LSQ）[3] を用いることで，この問題を解決する．ULB-LSQ ではアドレス計算後にエントリを割り当てる．リモートコアへのディスパッチを行うためのコア間ネットワークを用意し，アドレス計算後に該当するコアへディスパッチを行う．

Efficient Implementation of CoreSymphony Architecture
Tomoyuki NAGATSUKA†, Kenji KISE†
†Graduate School of Information Science and Engineering, Tokyo Institute of Technology

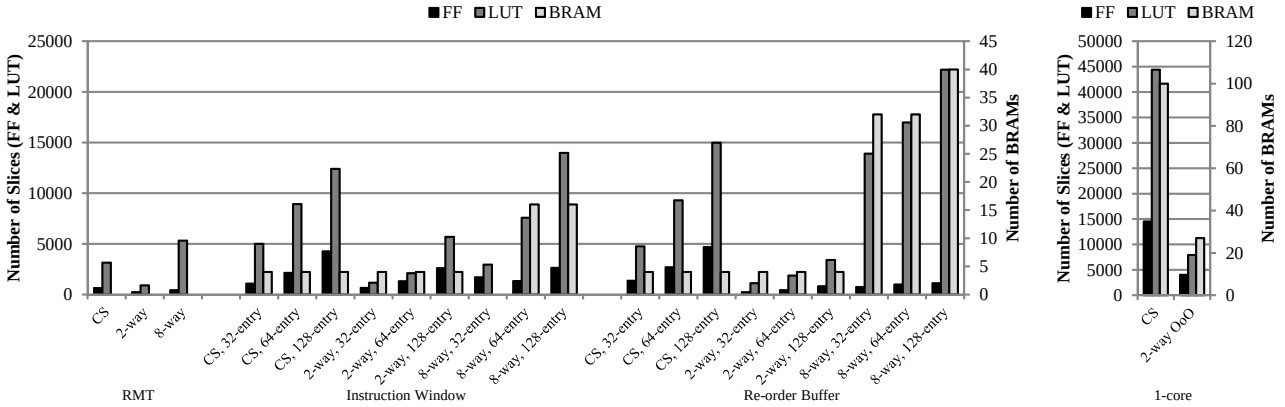

図 2: ハードウェア量の比較.

2.6 リオーダバッファ

リオーダバッファ（ROB）は全ての in-flight な命令の実行を管理し，プログラム順でのプロセッサ状態の更新と正確な例外を実現する．一般的に，プロセッサ中の in-flight な命令を増加させ，より多くの命令レベル並列性を抽出しようとすればするほど，大容量の ROB を必要とする．また，エントリの割り当てや命令の完了には，それぞれ命令のフェッチ幅・発行幅に応じたポート数を必要とする．CoreSymphony は協調により in-flight 命令数・フェッチ幅・発行幅が増加するため，4 コア協調時に十分な性能を引き出すには大容量・多ポートの ROB を必要としてしまう．

CoreSymphony では ROB の分散化 [1] を行うことでこの問題を解決する．単一の複雑な ROB を，より限定された情報を管理する 2 種類の ROB に分割することでハードウェア量を削減する．自コアで実行された命令の情報を管理する ROB を Local ROB（LROB），分岐や例外などの制御上必要な情報のみを全コアで共有する ROB を Global ROB（GROB）と呼ぶ．

3 評価

論理合成には，Xilinx ISE 14.3 を使用する．実装は Xilinx 社の評価ボード VC707 を対象に行う．

評価では，CoreSymphony を実現する上で大きく変更が加えられた RMT・命令ウィンドウ・ROB についてハードウェア量を評価し．次に，コア全体のハードウェア量と動作周波数を評価する．図 2 に評価結果を示す．

3.1 レジスタマップテーブル

評価結果から，CoreSymphony のリネーミング機構は，ベースとなる 2 命令発行スーパースカラの RMT と比較して，FF・LUT 共に 3 倍程度必要とすることがわかった．また，8-way 相当の RMT と比較すると，LUT の使用量を 40%程削減できることがわかった．

2-way リネーミングを行わない場合，各コアに 8-way 相当の RMT を必要とするため，2-way リネーミングは有効な手段といえる．

3.2 命令ウィンドウ

CoreSymphony は 8-way よりも多くの FF と LUT を消費する．これは 2-way リネーミングにより命令ウィンドウのエントリが拡張されたことと，PRF の分散管理のために追加された CAM によって必要とされる比較器が増加したことが原因であると考えられる．

ベースとなる 2-way 相当の命令ウィンドウと比較してかなり多くのリソースを消費する．しかし，PRF のポート数を大幅に削減することができるため，効率的であると考えられる．

3.3 リオーダバッファ

2-way 相当の ROB と比較し，4〜5 倍ほどの FF・LUT を消費することがわかった．これは新たに追加された GROB による影響と考えられる．8-way 相当の ROB と比較すると，FF の数は増加するものの，LUT と BRAM を大幅に削減できることがわかった．これは，大容量で多ポートの ROB を 2 つの小規模な ROB に分割したことによる影響と考えられる．

ROB の分散化を行わない場合，各コアが 8-way 相当の ROB を必要とする．そのため，2-way 相当の ROB と比較して多くのリソースを消費するものの，ROB の分散化は有効な手段と考えられる．

3.4 コア全体

ベースとなる 2 命令発行アウトオブオーダコアと比較すると，FF は 3.6 倍，LUT は 5.6 倍，BRAM は 3.7 倍必要とする．これは，ローカル命令キャッシュ，STU，RMT などの CoreSymphony を実現するために追加・変更されたモジュールによる影響と考えられる．ハードウェア量の削減は今後の課題である．

CoreSymphony の動作周波数は，1 コア時・4 コア協調時共に約 100MHz となった．レジスタリネーミングがクリティカルパスとなっている．ベースラインコアの動作周波数は約 138MHz である．これより，コアの協調による動作周波数への影響はないが，追加・変更したモジュールによる影響があることがわかる．2-way リネーミングのロジックのチューニングは今後の課題である．

4 まとめと今後の課題

本稿では，CoreSymphony を Verilog HDL で実装し，ハードウェア量と性能を評価した．その結果から，CoreSymphony 実現のために提案されてきたいくつかの手法の有用性を示した．

今後の課題として，CoreSymphony のハードウェア量の削減，動作周波数の改善，CoreSymphony の実機での動作が挙げられる．

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] Tomoyuki Nagatsuka, et al. CoreSymphony Architecture. In *Proc. of the 9th CF*, pp. 249–252, 2012.

[2] 若杉祐太ほか. 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.

[3] Simha Sethumadhavan, et al. Late-Binding: Enabling Unordered Load-Store Queues. In *Proc. of the 34th ISCA*, pp. 347–357, 2007.