4J-3

CoreSymphony P605のフロントエンドの実装

坂口　嘉一　†　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

1つのチップに複数のコアを集積するチップマルチプロセッサ（CMP）が主流になっている．CMPは複数のスレッドを同時に実行することで，高い性能を達成する．しかし，アムダールの法則に示されるように，CMPにおいてもシングルスレッド性能は重要である．そこでCMPにおいて，我々は高いシングルスレッド性能を得るため，CoreSymphony[1][2] アーキテクチャを提案している．

我々はCoreSymphonyアーキテクチャを適用したプロセッサ（CoreSymphony P605）のFPGAに実装を目指している．FPGAに実装することで，ハードウェア量の評価に加え，高速なシミュレーション環境として利用できる．本項では，FPGA向け実装のフロントエンド部分についての評価を行う．

2　CoreSymphonyのフロントエンド構成

CoreSymphony アーキテクチャ（以下 CoreSymphony）では，複数個のコアが協調し，仮想的な発行幅の広い1つのコアとしてプログラムを実行可能である．

CoreSymphonyにおけるフロントエンドの特徴として，次に3つが挙げられる．(1) 各コアが独立した命令の協調フェッチ．(2) 命令を複数のコアに分散する命令ステアリング．(3) コアをまたぐ依存を考慮したレジスタリネーミング．

図1にCoreSymphonyのフロントエンドの概略を示す．フロントエンドは，命令をフェッチする命令フェッチユニット（IFU），命令をデコードする命令デコードユニット（IDU），命令ステアリングを行うステアリングユニット（STU），レジスタリネーミングを行うレジスタリネーミングユニット（RRU）の4つのモジュールから構成される．

先に挙げた特徴の内，(1) はIFUが，(2) はSTUが，(3) についてはRRUが担っている．以下では，この3つのモジュールについて述べる．

2.1　命令フェッチユニット (IFU)

命令の協調フェッチを実現するため，IFUは通常の命令キャッシュ(conv.I-cache)に加え，ローカル命令キャッシュ（L-cache）を備えている．L-cacheはトレースキャッシュの一種であり，フェッチブロック（FB）と呼ばれる命令トレース単位でラインを割り当てる．1つのラインには，FBに含まれる命令の内，自コアにステアリングされたものとFBの制御情報が格納される．

CoreSymphonyの命令フェッチは，L-cacheのエントリ構築フェーズとL-cacheからフェッチする協調フェッチフェーズに分けられる．

フェッチ時にL-cacheにミスした場合，エントリ構築フェーズに遷移する．conv.I-cacheからFB全体をフェッチし，デコードとステアリングを行う．その後，自コアにステアリングされた命令とFBの制御情報をL-cacheに格納する．

Implementation of CoreSymphony P605 frontend
Yoshito Sakaguchi†, Tomoyuki Nagatsuka†, Kenji Kise†
†Tokyo Institute of Technology Graduate School of Information Science and Engineering

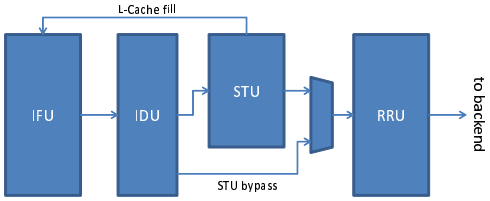

図1: CoreSymphonyアーキテクチャのフロントエンドの概略．4つのモジュールからなる

L-cacheにヒットした場合，協調フェッチが行われる．L-cacheに格納されている命令はステアリング済であるため，命令はデコード後，STUをバイパスしRRUに送られる．協調フェッチの場合，自コアにステアリングされた命令のみをフェッチするため，フロントエンドの実質的なスループットが増大する．

2.2　ステアリングユニット (STU)

図2に，STUの概略を示す．CoreSymphonyにおける命令ステアリングとは，命令を実行するコアを決定する操作である．ステアリングはFB単位で行う．ステアリングアルゴリズムにはリーフノードステアリング [1] を用いる．

また，STUではステアリングと並行してFBの制御情報の生成も行う．制御情報には，FBが結果を生成する論理レジスタの番号や，トレース中の分岐に関する情報などが含まれる．

リーフノードステアリングはFB内の命令を逆順に解析するため，FBに含まれる全命令がSTUに到達するまで，ステアリング結果が得られない．そのため，STUは結果が得られるまで命令を蓄えるバッファが設けられている．ステアリング結果が得られた後，自コアにステアリングされた命令をバッファから取り出し，次のステージに送る．

IFUでL-cacheから読みだされた命令はSTUをバイパスする．そのため，STUでステアリングが行われている間にL-cacheにヒットすると，命令の追い越しが発生する可能性がある．この状況を防ぐため，STUでステアリングが行われている間に，L-cacheから読みだされた命令がIDUに到達した場合，IFUとIDUをストールさせる．

2.3　レジスタリネームユニット (RRU)

図3にRRUの概略を示す．RRUでは，命令の依存関係を解決する．CoreSymphonyにおいてオペランドの供給元には，物理レジスタファイル (PRF)，論理レジスタファイル (LRF)，他のコア (remote) の3種類が存在する．RRUにおいて供給元を求め，ソースレジスタをリネームするタグを得る．

複数のコアで分散してプログラムを実行するため，コアを跨いだ依存が生じ得る．このような依存関係を追跡しつつ，ハードウェア規模を削減するため，CoreSymphonyでは2wayリネーミング [1] と呼ばれる手法で，レジスタをリネームする．

2wayリネーミングでは，コア内の依存を表すltagと

Copyright ©2012 Information Processing Society of Japan.
All Rights Reserved.

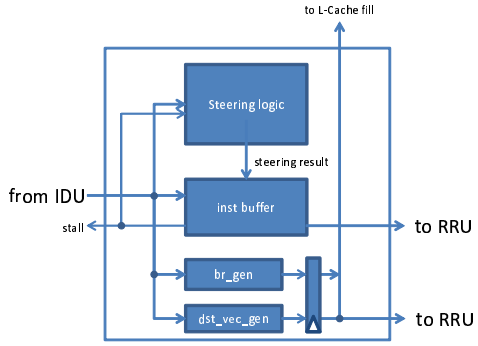

図 2: STeering Unit．ステアリングロジック，命令バッファ，FB 制御情報生成部に分かれる．

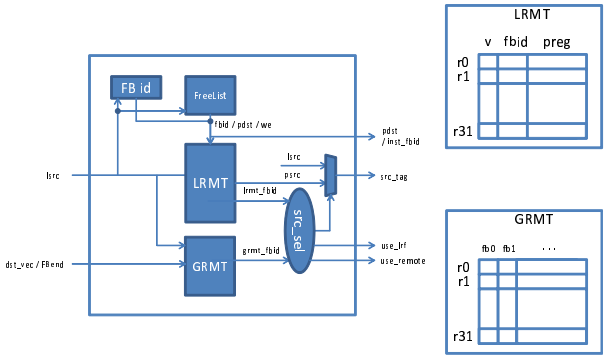

図 3: Register Renaming Unit と GRMT および LRMT

フェッチブロック間の依存を表す gtag の 2 種類のタグを用いて，依存関係を表現する．ltag は，自コアの物理レジスタ番号であり，gtag は論理レジスタ番号とその結果を生成する FB の番号を連結したものである．2 種類のタグはそれぞれ，LRMT と GRMT と呼ばれるテーブルで管理される．LRMT は ltag に加え，そのレジスタに書き込む命令の FB 番号も記録される．

ソースレジスタのリネーム時には，LRMT と GRMT 両方を読みだす．いずれかのエントリが有効な場合，それぞれの FB 番号を比較し，GRMT から読みだした FB 番号の方が若い場合には，gtag にリネームされ，他のコアから値の供給を受ける (use_remote) とマークされる．そうでない場合は ltag にリネームされる．

LRMT と GRMT 両方のエントリが無効な場合，ソースレジスタは，リネームされず（論理レジスタ番号をとする）LRF から値を読みだす (use_lrf) とマークされる．

バックエンドの命令ウィンドウでは，タグと 2 種のマークを使用し，命令発行をスケジュールする．

3 評価

ここでは，フロントエンドのハードウェア規模および動作周波数を評価する．実装は Xilinx 社の評価ボード ML605 に行う．論理合成には ISE 13.2 を使用する．

conv.I-cache はダイレクトマップで，容量は 4KB，ラインサイズは 2，L-cache はダイレクトマップで，256 エントリ（1024word 分）とした．

3.1 ハードウェア量

表 1 に論理合成の結果得られた，リソース占有量を示す．フロントエンド全体 (frontend) および，IFU, IDU, STU, RRU 毎のリソース占有量に加え，L-cache, conv.I-cache ステアリングロジック (leaf-node), STU 内の命令バッファ(inst-buf), GRMT，LRMT についても占有量を求めた．

表 1: モジュール別リソース占有量

モジュール	FF	LUT	BRAM
IFU	762	1366	33
conv.I-cache	71	100	3
IFU.L-cache	359	647	18
IDU	275	162	0
STU	1232	2669	8
leaf-node	855	2194	0
inst-buf	63	279	8
RRU	1244	4207	0
grmt	268	823	0
lrmt	382	1465	0
frontend	3827	8791	0
conventional	4877	9335	31

比較のため，2 命令発行のアウトオブオーダコア（conventional) のリソース使用量を示す．これはパイプライン全体に命令，データキャッシュ（それぞれ 4KB，ダイレクトマップ）を含めた値である．

表から，CoreSymphony のフロントエンドのみで，conventional に相当するリソースを消費していることがわかる．conventional には存在しない STU および，複雑化したリネーム機構による増分が大きい．このオーバーヘッドは許容できるものではなく，ハードウェア量の削減が必要といえる．

3.2 動作速度

論理合成の結果によると，モジュール内での最長遅延は 7.907ns（約 126MHz）であった．L-cache からのフェッチがクリティカルパスになっている．

conventional（コア全体）の動作周波数は約 95MHz であるので，フロントエンドのモジュールが，今後クリティカルパスになる可能性は低いと考えられる．

4 まとめ

CoreSymphony アーキテクチャのフロントエンド部分を論理合成し，ハードウェア規模を評価した．その結果，動作周波数に悪影響を与える可能性は低いと分かった．一方，ハードウェアの増加量が多く削減が必要といえる．

今後は，バックエンドの実装を進め，コア全体の動作を目指す予定である．

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] 若杉他．調可能スーパースカラ CoreSymphony 情報処理学会論文誌コンピューティングシステム, Vol.3, No.3, pp. 67-87 (September 2010)

[2] Nagatsuka et al. CoreSymphony: An Efficient Reconfigurable Multi-core Architecture international Workshop on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2011), pp. 29-34 (June 2011)

Copyright ©2012 Information Processing Society of Japan.
All Rights Reserved.