Vol.2014-ARC-208 No.15
2014/1/23

スーパースカラ・プロセッサ「雷上動」の設計と実装

藤田 晃史[1]　中島 潤[1]　早水 光[1]　塩谷 亮太[2]

概要：
本稿ではスーパースカラ・プロセッサ「雷上動」の設計と実装について述べる．雷上動は 32bit ARM 命令セットのサブセットを実行するプロセッサである．雷上動では最大 64 インフライト命令を動的にスケジューリングして実行し，最大 5 命令を同時発行可能である．雷上動ではスーパースカラ・プロセッサや FPGA に関する様々な研究成果を取り入れることにより，高面積効率な実装を実現している．FPGA 向けに合成した結果， Xilinx Spartan-6 上で 18000 LUT 程度の回路規模となり，60 MHz で動作した．

1. はじめに

雷上動は 32bit ARM 命令セットのサブセットを実行可能な out-of-order スーパースカラ・プロセッサである．The 1st IPSJ SIG-ARC High-Performance Processor Design Contest に提出した構成では最大 64 インフライト命令を動的にスケジューリングし，最大 5 命令を同時発行可能である．雷上動ではスーパースカラ・プロセッサや FPGA に関する様々な研究成果を取り入れることにより，高面積効率な実装を実現している．

本稿では，この雷上動の設計と実装について述べる．本稿の以降の構成は以下のとおりである．2 節では雷上動で実行できる命令セットについて説明する．3 節ではマイクロアーキテクチャの概要について説明し，4 節と 5 節ではフロントエンドとバックエンドの詳細なマイクロアーキテクチャについて説明する．6 節では開発環境について述べる．7 節では各ベンチマーク向けに行ったソフトウェア最適化について説明し，8 節でまとめる．

2. 命令セット・アーキテクチャ

雷上動では 32bit ARM 命令セット（ARMv5te）のサブセットを実行可能である．実行可能な命令は，主な整数演算命令やロード/ストア命令などであり，浮動小数点命令や OS の制御に関わる命令は実装していない．

あとで詳しく述べるように，これらの命令はプロセッサ内部で複数のマイクロ命令に分解されて実行される．一部の複数ロード/ストア命令などのマイクロ命令への分解が困難な命令についてはサポートしておらず，これらはアセンブリ・プログラムの段階であらかじめ複数の命令に分解しておくことにより対応している．

[1] 東京大学 大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
[2] 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University

3. マイクロアーキテクチャの概要

本節では雷上動のマイクロアーキテクチャの概要について説明する．雷上動は Alpha 21264 [1] や IBM Power 7 [2]，Intel Haswell [3] などと同様の物理レジスタ・ベースのスーパースカラ・プロセッサである．**図 1** に雷上動のブロック図を，**図 2** に命令パイプラインの構成をそれぞれ示す．また，各部のパラメータを**表 1** に示す．

雷上動の命令パイプラインは大きくフロントエンドとバックエンドに分けられる．フロントエンドでは命令フェッチや命令デコード，レジスタ・リネーミングを行う．また各種キューやバッファのエントリを確保する．これらの処理を終えた命令はバックエンドにディスパッチされ，命令は発行キューから各実行ユニットに out-of-order に発行されて実行される．

Name	Parameter
fetch width	2 inst.
issue width	5 micro-ops.
commit width	4 micro-ops.
issue queue	16 entries, unified
exec. units	2 int / 1 complex int
	1 load / 1 store
active list	64 entries
ld./st. queue	16/16 entries
branch pred.	2-bit saturating counter,
	8K entries PHT, 1K entries BTB
L1C(I)	8 KB, 2 way, 16 B/line
L1C(D)	16 KB, direct-mapped, 16 B/line
physical register file	64 entries for general registers
	32 entries for flag registers

表 1　雷上動の構成

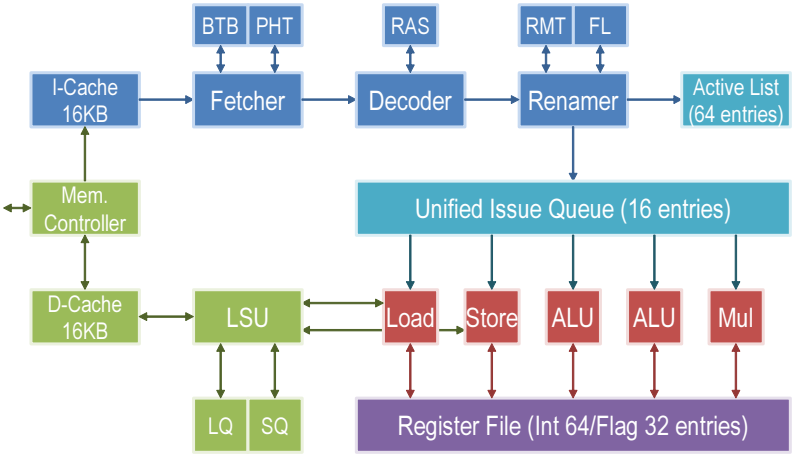

図 1 雷上動のブロック図

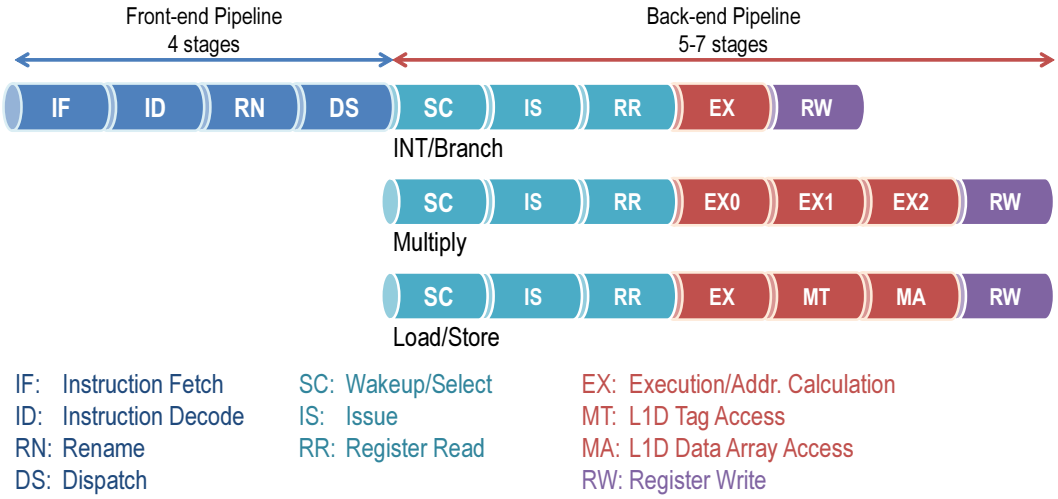

図 2 雷上動の命令パイプライン

次節からは，これらフロントエンドとバックエンドの命令パイプラインの動作について詳しく説明する．

4. フロントエンド・パイプライン

本節では雷上動のフロントエンド・パイプラインについて説明する．

フェッチ

フェッチ・ステージでは分岐予測器が予測した命令アドレスに基いて命令がフェッチされる．分岐予測器は 2 ビット・カウンタを用いた方向分岐予測器と BTB からなる．命令キャッシュや BTB の構成については表 1 に示すとおりである．

デコード

デコード・ステージではフェッチされた ARM 命令をデコードし，最大 3 命令までのマイクロ命令に分解する．ARM 命令が複数のマイクロ命令に分解された場合，それらは複数サイクルかけて次段のステージに送られる．このように 1 つの ARM 命令を複数のマイクロ命令に分解するのは，ARM 命令セットでは 1 つの命令が非常にたくさんのオペランドを持つ場合があるためである．また，ARM 命令セットでは PC が汎用レジスタにマッピングされているため，多くの命令がレジスタ間接分岐と同様の機能を持つ．雷上動ではこれらの命令を演算命令やロード/ストア命令とレジスタ間接分岐命令に分解する．

リネーム

リネーム・ステージではレジスタ・リネーミングと各種資源の確保を行う．リネームを行うリネーム・ロジックは主に RMT (Register Map Table) とフリー・リスト，レジスタ番号の比較器からなる．これらを用いたレジスタ・リネーミングは以下のように行われる．

(1) ソース・レジスタの論理レジスタ番号を使用して RMT を参照し，物理レジスタ番号に変換する．
(2) デスティネーション・レジスタの論理レジスタ番号に現在割り当てられている物理レジスタ番号を得るため，RMT を参照する．この時得られた物理レジスタ番号は物理レジスタ解放の際に使用される [4]．
(3) フリー・リストから物理レジスタ番号を取り出し，デスティネーション・レジスタに割り当てる．また，割

り当てられた物理レジスタ番号を RMT に書き込み，更新する．

なお，32bit ARM 命令セットにはフラグ・レジスタも存在するため，これについても別途レジスタ・リネーミングを行う．RMT とフリー・リストは，汎用レジスタとフラグ・レジスタ用にそれぞれ用意される．

ディスパッチ

ディスパッチ・ステージではリネーム済の命令を発行キューに書き込む．発行キューに書き込まれた命令は，wakeup/select の対象となる．

5. バックエンド

本節では雷上動のバックエンド・パイプラインについて説明する．

スケジューリング

スケジューリング・ステージでは発行キューにディスパッチされた命令をスケジューリング（wakeup/select）し，命令を out-of-order に発行する．雷上動では全ての種類の命令を単一の発行キューに格納する構成をとる．Wakeup logic には matrix scheduler を使用している [5], [6]．Matrix scheduler は依存行列を用いて命令の wakeup を行う手法であり，通常の CAM を用いる wakeup logic と比べて省資源かつ高速に実装可能である．Wakeup/selct はプロセッサ全体のクリティカル・パスとなっているが，雷上動では matrix scheduler により現実的な速度で 5 命令同時発行を可能としている．

発行

発行ステージでは，select された命令を発行キューから読み出して次段以降のステージに送り出す．

レジスタ読み出し

レジスタ読み出しステージでは物理レジスタ・ファイルからソース・オペランドを読み出す．物理レジスタ・ファイルは汎用レジスタとフラグ・レジスタ用にそれぞれ専用のものが用意されている．また，オペランド・バイパスのためのレジスタ番号の比較もこのステージで行われる．

整数演算の実行

整数演算パイプラインでは ALU/シフタ/ビット・カウンタの各演算器が存在し，それぞれ対応した命令を実行する．ビット・カウンタとは，ソース・オペランドの上位から連続しているビット 0 の数を数える演算器である．

乗算の実行

乗算パイプラインでは，3 段にパイプライン化された乗算器が乗算を実行する．

ロード/ストアの実行

ロード・パイプラインはアドレス計算，タグ・アクセス，アレイ・アクセスの 3 ステージからなる．タグ・アクセス・ステージではデータ・キャッシュのタグにアクセスし，ヒット・ミス判定を行う．また，このステージではロード・キュー/ストア・キューへのアクセスを行い，ストア-ロード・フォワーディングやアクセス順序違反の検出を行う．そして続くアレイ・アクセス・ステージでロード結果を得る．

ストア・パイプラインも，ロードと同じ 3 ステージからなる．ただし，アレイ・アクセス・ステージではキャッシュへの書き込みは行わない．ストアの実行結果はストア・キューに書き込まれ，後のコミット時にデータ・キャッシュに書き戻される．

ロード・ストア・ユニットは，データ・キャッシュやロード・キュー，ストア・キューからなる．データ・キャッシュはノンブロッキング・キャッシュとなっているため，キャッシュ・ミスが発生した場合もその他の命令はデータ・キャッシュにアクセスが可能である．キャッシュ・ミスは MSHR (Miss Status Handling Register) によって管理され，最大 2 つまでのキャッシュ・ミスを同時に扱うことができる．ロード・キューには，ロード命令の実行時にロードアドレスが格納される．ストア命令は実行時にロード・キューを検索し，同じアドレスの後続ロードを発見した場合は，メモリアクセス順序違反として検出する．違反を検出したストア命令より下流の命令は全てフラッシュされ，フェッチ・ステージから実行を再開する．ストア・キューには，ストア命令の実行時にストア・アドレスとストア・データが格納される．ロード命令は実行時にストア・キューを検索し，同じアドレスの先行ロードを発見した場合は，フォワーディングを行ってそのストア・データをロード結果とする．

レジスタ書き込み

レジスタ書き込みステージでは物理レジスタ・ファイルに演算結果を書き込む．また， active list に命令の実行が終了したことを記録する．

コミット

コミット・ステージでは命令をコミットする．また，物理レジスタの解放やストア・データのデータ・キャッシュへの書き戻しを行う．命令のコミットは active list の先頭を読み出して行われる．Active list には，コミットに必要な命令の実行状態やオペランドのレジスタ番号などが記録されており，これらを用いてコミットを行う．実行時に検出したアクセス順序違反や分岐予測ミスなどはこの際に処理され，後続する命令のフラッシュや状態の回復，再実行が行われる．雷上動ではこの active list に分離 ROB [7] を採用している．このため，投機ミスからの回復先アドレスを active list ではなく単一のレジスタに格納することができ，回路面積を大きく縮小している．

6. 開発環境

本節では雷上動の開発環境について説明する．開発に使用したソフトウェアを**表 2** に示す．雷上動の実装では

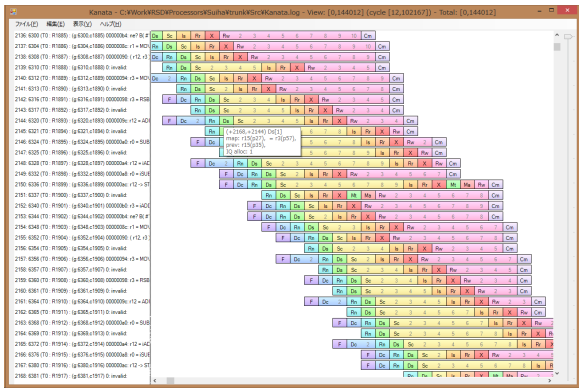

図 3 Kanata による雷上動の実行の様子

System Verilog を使用してハードウェアを記述した．System Verilog の持つ interface などの機能を積極的に使用することにより，記述量を大幅に削減しながら可読性の高い記述を実現している．シミュレーションや合成を行うソフトウェアには，System Verilog への対応が進んでいる点を重視し Mentor Graphics QuestaSim と Synopsis Synplify を使用している．

雷上動ではプロセッサ・シミュレータ鬼斬 [8] 用に開発されたパイプライン可視化ツール Kanata を移植して使用することにより，検証の効率を大幅に向上させている．**図 3** に Kanata による雷上動の実行の様子を示す．Kanata では命令パイプラインの状態を可視化し，各種資源の確保や解放，リネーミングの結果などをオーバーレイして表示可能である．このような機能を使うことにより，雷上動の開発では複雑なスーパスカラ・プロセッサの振る舞いを容易に確認することを可能としており，検証の効率を大きく向上させている．

7. ベンチマーク・プログラムの最適化

本節では The 1st IPSJ SIG-ARC High-Performance Processor Design Contest の課題向けに行ったソフトウェアの最適化について説明する．

ソート

ソート・アルゴリズムには radix ソートをベースとしたものを使用した．また，ソフトウェア・プリフェッチ命令を挿入することにより，キャッシュ・ミス・レイテンシの隠蔽を行っている．

用途	ソフトウェア
HDL シミュレーション	Mentor Graphics QuestaSim 10.2c
論理合成	Synopsys Synplify Premier I-2013.09-1
配置配線	ISE Design Suite 14.6
パイプライン可視化	Kanata 1.24
コンパイラ	GNU GCC 4.8.2

表 2 開発環境

密行列積

データ・キャッシュのヒット率を向上させるため、ブロック化やループ順序の変更，ソフトウェア・プリフェッチ命令の挿入などを行った．

ステンシル計算

ステンシル計算で行う周囲 9 ピクセルの加算処理を縦方向と横方向に分割して処理している．これに加えてブロッキングを行うことにより，結果として加算とロード命令の実行回数を大幅に減らしている．

最短経路問題

アルゴリズムの最適化とソフトウェア・プリフェッチ命令の挿入により，性能を向上させている．

8. おわりに

本稿では雷上動の設計と実装について述べた．雷上動ではスーパースカラ・プロセッサや FPGA に関する様々な研究成果を取り入れることにより，高面積効率な実装を実現している．今後の課題としては クロック速度の最適化があげられる．本稿の投稿時では，雷上動はマイクロアーキテクチャの検討や実装が終わったばかりの段階であり，クロック速度を上げるための最適化をまだほとんど行っていない．

謝辞 本研究は一部，文部科学省科学研究費補助金 No. 23300013，および No. 24680005 による．

参考文献

[1] Kessler, R.: The Alpha 21264 microprocessor, *IEEE Micro*, Vol. 19, No. 2, pp. 24–36 (1999).
[2] Sinharoy, B., Kalla, R., Starke, W. J., Le, H. Q., Cargnoni, R., Van Norstrand, J. A., Ronchetti, B. J., Stuecheli, J., Leenstra, J., Guthrie, G. L., Nguyen, D. Q., Blaner, B., Marino, C. F., Retter, E. and Williams, P.: IBM POWER7 Multicore Server Processor, *IBM J. Res. Dev.*, Vol. 55, No. 3, pp. 191–219 (2011).
[3] Krewell, K.: Intel's Haswell Cuts Core Power, *Microprocessor Report 9/24/12*, pp. 1–5 (2012).
[4] Yeager, K.: The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, Vol. 16, No. 2, pp. 28–41 (1996).
[5] 五島正裕：Out-of-Order ILP プロセッサにおける命令スケジューリングの高速化の研究，博士論文，京都大学大学院情報学研究科 (2004).
[6] Sassone, P. G., Rupley, II, J., Brekelbaum, E., Loh, G. H. and Black, B.: Matrix Scheduler Reloaded, *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 335–346 (2007).
[7] 岩原佑磨，安藤秀樹：リオーダ・バッファのハードウェア量と消費エネルギーの削減，先進的計算基盤システムシンポジウム SACSIS 2010，pp. 37–44 (2010).
[8] 塩谷亮太，五島正裕，坂井修一：プロセッサ・シミュレータ「鬼斬弐」の設計と実装，先進的計算基盤システムシンポジウム SACSIS 2009，pp. 120–121 (2009).