計算機アーキテクチャ 139−10
（2000. 8. 4）

大規模データパス・アーキテクチャの実行機構

安島 雄一郎, 辻 秀典, 坂井 修一, 田中 英彦
東京大学大学院 工学系研究科

我々は大規模データパス (VLDP)・アーキテクチャを提案している。VLDP アーキテクチャでは複数パス実行と大規模投機的実行で実効 IPC 8 を目指している。本論文では、多数の実行ユニットと分散レジスタにより、複数の実行パスから得られる大量の命令を同時並列に実行可能な実行機構を提案する。本機構では分岐予測ミスが起きた場合も部分的な命令の無効化だけで処理を継続でき、制御依存問題を改善できる。また、レジスタ値転送、実行ユニット割り当て、無効化などの手順も示し、実行部の具体的な動作を説明する。

Execution Mechanism for the Very Large Data Path Architecture

Yuichiro Ajima, Hidenori Tsuji, Shuichi Sakai, Hidehiko Tanaka
University of Tokyo, Graduate School of Engineering

We propose the Very Large Data Path (VLDP) Processor Architecture which achieves actual IPC 8 by using Multi-Path Execution and Massive Speculative Execution. This paper presents execution mechanism for VLDP; the Multiple Execution Unit and Distributed Register can issue many instructions in a cycle and continue processing with partial instruction invalidation when a branch is mispredicted. We also discuss process of register data transfer, execution unit assignment and instruction invalidation.

1 はじめに

大規模データパス (Very Large Data Path, 以降 VLDP) アーキテクチャは、スーパースカラ, VLIW などの従来のアーキテクチャの延長ではない新しいアーキテクチャである [1][7]。VLDP では、大規模にハードウェア資源を利用することでマイクロプロセッサの性能を大幅に向上させ、性能目標として実効 IPC 8 の達成を目指している。

VLDP アーキテクチャの主な特徴は、複数パス実行による大規模な投機的実行 [4]、32 命令幅の命令ブロック (Instruction Block, 以降 IB) を実行単位とする高スループット処理 [5][6]、多数の機能ユニットによる命令の並列実行 [7]、レジスタを介さないデータアクセスの明示的処理 [5] である。本稿ではこれらを実現する実行機構を提案する。

2 VLDP アーキテクチャ

2.1 概要

VLDP アーキテクチャではサイクル当たりの高い処理スループットを得るために、最大 32 命令をまとめる IB を処理単位として各種処理を高速化する。その一方で 1 サイクルにフェッチする IB は最大 1 とし、各種処理を簡単にする。そして IB の実行を行なう実行ユニット (Execution Unit, 以降 EU) を複数用意し、複数の IB を並列に実行することで、全体として多数の命令の並列処理を行う [7]。

大規模な投機的実行では十分な演算資源確保のために大きなレジスタファイルが必要となるが、レジスタファイルへの負荷集中を避けるために VLDP では分散したレジスタファイル構成 [3] をとる。各 EU に分散レジ

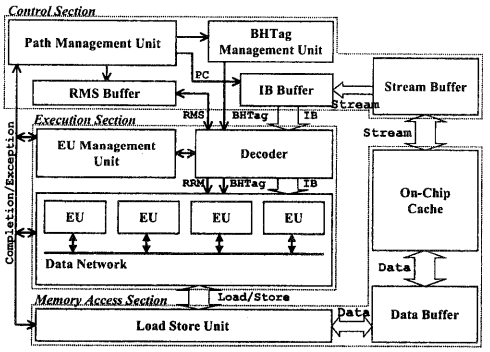

図 1: VLDP block diagram

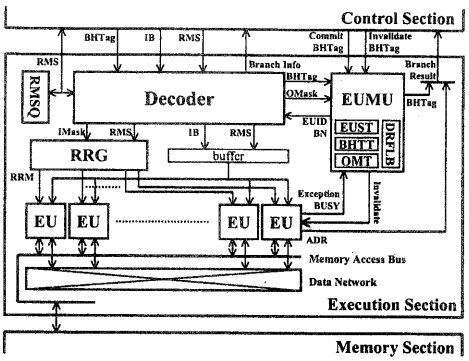

図 2: VLDP Execution Section

スタファイル (Distributed Register File, 以降 DRF) を置き、EU の演算結果はその EU の DRF にのみ保存する。また IB 内部で一時的に使われる値は、値を生産する命令を明示的に指定することで DRF を介さずアクセスされる。論理レジスタ番号と値が格納されている DRF の位置を結び付ける Register Map Set(以降 RMS) を利用することにより、各 EU は分散レジスタから値を得る。なお、初期型 VLDP の論理レジスタは 64 本と定められている。

VLDP は複数パスを実行するため、実行中の全命令の処理前後における RMS は RMS Buffer に格納され、フェッチした IB にあわせて読み出される。IB と RMS を併せて実行機構に投入することにより、VLDP では毎サイクル、複数の実行パスから任意のフェッチ先を選んで実行できる。また、実行中の多数の命令から特定のパスの命令だけをまとめて削除するため、IB には分岐履歴タグ (Branch History Tag, 以降 BHTag) が添付される。なお分岐制御を簡単にするため、IB を 8 命令ずつの 4 つのフィールドに分割し、各フィールドの末尾にのみ分岐命令を許す。また、IB には RMS を高速に更新するための更新値テーブル Output Register Map(以降 OMap), Output Register Mask(以降 OMask)、及びレジスタ・リクエストを高速に生成するために要求レジスタ・マップ Input Register Mask(以降 IMask) が静的に付加されている。

VLDP のブロック図を図 1に示す。VLDP アーキテクチャは大きく分けて Control, Execution, Memory Access (制御部, 実行部, メモリ部) の 3 つのセクションに分かれる。制御部は分岐予測によるフェッチ制御、RMS の管理、BHTag の管理を行なう。実行部は複数の EU による IB の実行、RMS の更新、DRF アクセス制御を行なう。メモリ部は実行中の命令によるメモリアクセスを仮想化し、ストア命令による書き込みを命令が commit されるまで遅らせる。

2.2 実行部の機能

ここでは実行部に要求される機能を述べる。まず、制御部はフェッチした命令に RMS と BHTag を添付し、実行部に渡す。実行部ではこれを解釈し、IB の各抜け出し点 1〜4 箇所における RMS の生成、分岐命令種別の判定を行ない制御部に通知する。同時に既に投入された IB を並列に実行し、メモリアクセスや分岐結果の feedback も行なう。制御部は分岐結果を受けて。特定パスの無効化や Correct Path が確定した IB の commit を指示する。実行部は指示に従い、演算資源の解放・再利用、メモリアクセスの無効化・確定、例外発生の通知を行なう。

3 VLDP 実行機構の提案

本節では VLDP の要求を満たす実行部の機構を提案する。

3.1 実行部の概要

図 2に提案する実行機構のブロック図を示し、各ユニットの機能を述べる。

デコーダは制御部から IB, RMS, BHTag を受け取り、実行部内の各ユニットに情報を

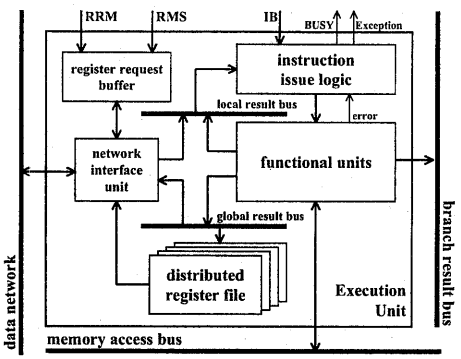

図 3: Execution Unit

分配する。また、受け取った IB の抜け出し点の RMS を生成し、制御部に返すとともに RMS Queue (以降 RMSQ) に格納する。デコーダは制御部の指示により、RMSQ から RMS を得る場合がある。

EU 管理ユニット (EU Management Unit, 以降 EUMU) は EU, DRF の使用状況を管理し、EU, DRF の割り当て、無効化の制御を行う。内部には EU 状態テーブル (EU Status Table, 以降 EUST), BHTag テーブル (以降 BHTT), OMask テーブル (以降 OMT) 及び分散レジスタファイル・ロック・バッファ (DRF Lock Buffer, 以降 DRFLB) を持つ。

レジスタ・リクエスト生成器 (Register Request Generator, 以降 RRG) は、各 EU が他の EU へ送るレジスタのマップ (Register Request Map, 以降 RRM) を生成する。RRM は EU 別に生成され、それぞれの EU に送られる。

EU は命令ブロックが割り当てられると演算を開始し、結果を内部の DRF に書き込んで待機状態に戻る。レジスタ・リクエストに基づくレジスタ値の送信は常に行なう。

buffer は命令デコード・ビット列や RMS の一部など、ビット幅の広いデータを各 EU へ送信するための中継器である。

なお、図 2ではバスの調停や EU へ IB を割り当てるためなどの制御線は省略してある。

3.2 実行ユニットの構成

図 3に EU のブロック図を示し、以下に各ユニットの機能と動作を述べる。

instruction issue logic はデコードされた IB を受け取り、オペランドをバッファする。各命令の発火条件を調べ、発火可能なものから functional units に送る。また、issue logic は EU での実行状態 (BUSY, Exception) を EUMU に通知する。functional units は各命令に対応した機能ユニットの集合体で、演算結果は global result bus または local result bus に出力する。演算内容によっては branch result bus への出力や memory access bus へのアクセスを行なう。DRF は global result buffer に出力された演算結果を保存し、issue logic は local result buffer に出力された演算結果を取り込む。DRF はバンク構成で、1 バンクのエントリ数は IB の最大命令数 32 となっている。

register request buffer(以降 RRB) はデコーダから RRM と RMS の一部を受け取り、送り先 EU 別にレジスタ・リクエストのリストを保存する。network interface unit(以降 NIU) は RRB を参照して DRF を読み出し、Data Network に送る。また、外部からレジスタ値を受け取った場合、これを local result buffer 経由で issue logic に渡す。

なお、初期型 VLDP では EU 16 ユニット、DRF 各 8 バンクとされており、DRF 総バンク数は 128 である。DRF の総バンク数は EU の総数と論理レジスタ数の和を越えていれば良いので 47 バンクが余剰である。この余剰は、使用済みバンクの偏りによる EU の飽和を緩和するために設けられている。

4 実行部の動作

本節では、前節で提案した実行機構の各部の動作と詳細を、実行部の機能動作に即して説明する。

4.1 RMS 生成

デコーダでは IB 先頭での RMS から各抜け出し点における RMS を生成し、制御部に返す。RMS 生成では、IB に含まれる OMap, OMask を用いて先頭 RMS の該当のエントリを一括置換する。この生成の仕組みを図 4に示す。

デコーダはまず EUMU より、IB を割り当てる EU 番号 (EUID) と分散レジスタ番号 (Bank Number, 以降 BN) を得る。これは RMS の EUID, BN フィールドの置換に

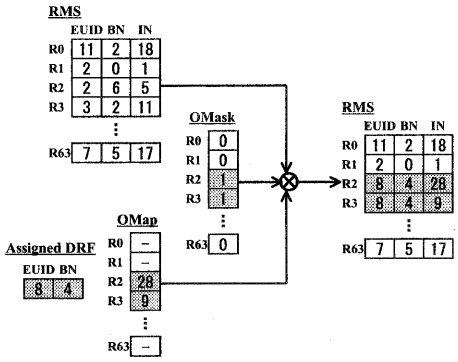

図 4: RMS generation

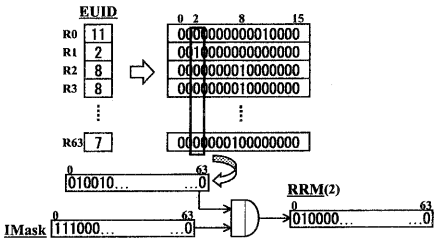

図 5: register request generation

使用される。命令番号 (Instruction Number, 以降 IN) の置換には OMap を使う。置換するエントリは OMap で示されており、図ではレジスタ 2, 3 が置換される。

4.2 RRM 生成

RRG では各 EU に送る RRM を生成する。RRM は、そのサイクルで新しく IB を割り当てられる EU に対してどの論理レジスタの値を送るかを示す。RRM の各ビットは、RMS で該当 EU に値があるとされ、かつ IMask により送信先 IB が要求しているレジスタのみが 1 、それ以外は 0 となる。RRM 生成方法を図 5に示す。

まず RMS の EUID フィールド全エントリをデコードし各 EU に対応するビットを取り出すことにより、該当 EU に値があるレジスタのマップを生成する。図では EU(2) のマップを取り出している。これと IMask との論理積を取ることにより、該当 EU の RRM が得られる。

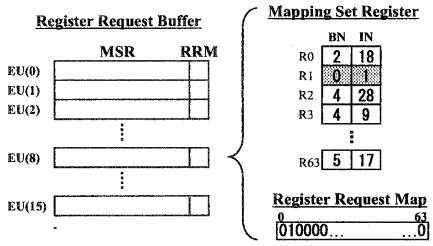

図 6: Register Request Buffer

4.3 レジスタ値転送

新しい IB がいずれかの EU に割り当てられる毎に、全ての EU にそれぞれの RRM と共通の RMS (BN, IN フィールドのみ) が送られる。これらの情報は EU 内の RRB に蓄積され、EU 内の NIU によって読み出される。RRB の構成を図 6に示す。

RRB のエントリは送り先 EUID に関連付けられており、MSR フィールドには RMS の BN, IN フィールド、RRM フィールドには RRM が格納される。NIU は RRM に 1 が立っているビットを探し、MSR を参照して DRF にアクセスする。もしくは global result bus を監視して値を得る。

NIU は得られた値を Data Network を通して送り先 EU に転送する。受信側の EU では NIU が値を一旦バッファし、local result bus を通して issue logic に値を渡す。

以上の手順より Data Network には、新しい IB が EU に割り当てられる度に他の EU から一斉にデータが送られるというトラフィックの特徴がある。Data Network 構造、IB 割り当て戦略は、この特徴を踏まえて設計する必要がある。

4.4 実行ユニット管理・完了処理

EUMU では EU, DRF の使用状況を管理し、割り当て、解放の処理を行なう。まず、EU で実行中の IB について、各フィールド末尾での OMask, BHTag をそれぞれ OMT(図 7), BHTT(図 8) を保存する。なお、末尾が抜け出し点でないフィールドの OMask は保存しなくてよい。OMask は完了による DRF の解放に使用し、BHTag は無効化による DRF などの演算資源の解放に使用する。

また、EU の状態は EUST(図 9) に保持す

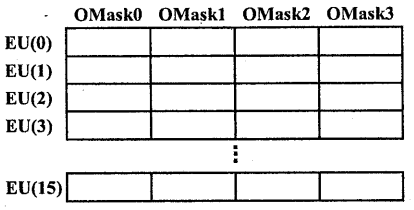

図 7: OMask Table

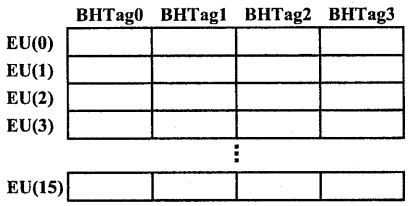

図 8: BHTag Table

る。active ビットは EUMU が既に IB を割り当てた EU で 1 にセットされる。bank valid ビットは EU 内の各 DRF の使用状況を表す。BN は active な EU がどの DRF バンクに書き込むかをバンク番号で格納する。これは割当時に bank valid を参照して決定される。commit ビットは、制御部から Correct Path であることが通知された IB を処理している EU のエントリがセットされる。制御部からは BHTag で Correct Path が通知されるため、IB の途中までが正しいパスで、途中から無効なパスである場合がある。EUMU は BHTT を参照し、どのフィールドまでが正しいパスであるかを判定して EUST の lelvel フィールドに書き込む。

EU が IB の処理を完了すると、BUSY 信号が 1 から 0 に落される。EUMU はこれを検出し、active な EU である場合は IB 実行が終了したとみなす。この時 Exception 信号が立ってなければ、EUMU は IB の完了処理を行なう。

まず、EUMU は EUST の commit ビットを読み出し、該当 IB が正しいパスに含まれているか検査する。commit ビットが立っていない時は、commit されるか IB が無効化されるまで待つ。IB 実行終了かつ commit されている場合、EUMU は EUST の active ビットをリセットし、EU を解放する。また、全 EU の RRB から解放する EU へ

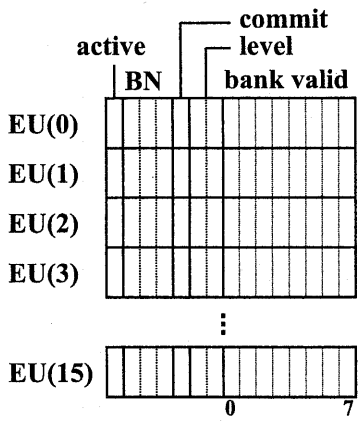

図 9: EU Status Table

の RRM をクリアする。同時に、active になっていた DRF バンクを DRFLB に登録し、ロックする。これは、演算が終了しても後続の命令が演算結果の格納された DRF にアクセスするためである。DRFLB にはロックする DRF の EUID, BN とともに正しいパスの抜け出し点における OMask が保存される。ここで保存すべき OMask は、EUST の level フィールドを使用して OMT を参照することで得られる。新しい DRF が DRFLB に追加される時、全エントリの OMask が、新しい DRF の OMask との論理積に更新される。これは、新しく追加した DRF に含まれるレジスタ値によって、古い DRF が参照される可能性がなくなるのをチェックするためである。この更新によって OMask のビットが全て 0 になると、そのエントリは DRFLB から取り除かれ、対応する EUST の bank valid ビットがリセットされる。これにより、DRF の再利用が可能になる。

4.5 命令無効化手順

制御部より命令の無効化が指定された場合、EUMU は指定 BHTag とその子孫に対応する実行中の IB を全て無効化し、EU から取り除く。無効化の操作は EUMU から各 EU を個別に制御し、無効化される EU では演算の停止とメモリアクセスの無効化を行ない、無効化されない EU では無効化される EU への RRM をクリアする。その後各 EU の無効化操作の終了時点で、無効化した EU に対応する EUST の active ビットがリセッ

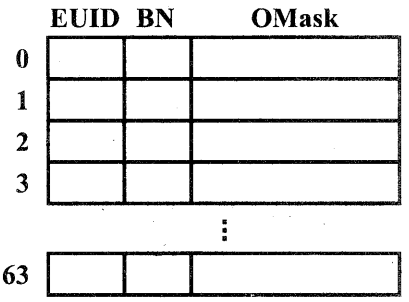

図 10: DRF Lock Buffer

トされ、EU が再利用可能になる。

また、IB の途中までが commit された場合、その IB の commit されなかった部分を無効化する。この場合、無効化部分の演算停止とメモリアクセスの無効化のみが行なわれ、後は commit 操作によって処理される。

4.6 分岐結果 feedback

EU での分岐命令処理の結果により、実行パスの選択が行なわれる。この結果を制御部に渡すことにより、正しいパスの実行が継続される。分岐命令の結果が出ると EU ではアドレスを、EUMU では BHTT を参照して該当の BHTag を出力する。これをあわせて制御部に転送する。単純には 1 サイクル 1 IB のスループットより、1 サイクルに 4 程度の分岐結果を返すことができればよいと予想される。

また、VLDP では分岐予測及び分岐確信度予測 [4] により投機的に実行を進めるため、分岐命令の存在、種別を早い段階で制御部に通知することによって効率的な命令フェッチを行なえる。このため、デコーダは IB デコード時に分岐命令の存在と種別を即座に制御部に返す。

5 まとめ

本論文では新しいプロセッサ・アーキテクチャ VLDP の実行機構を提案し、各ユニットの詳細と動作について説明した。

6 今後の課題

現在 VLDP プロジェクトではクロックレベルのシミュレータを作成中であり、これを用いて実行部各ユニットの改良、調整を行なう。また、VLDP を大規模化すると Data Network の階層構造化が要求されるため、IB 割り当て戦略などを含めた検討が必要になる。

本研究の一部は、文部省科学研究費補助金 (基盤研究 (B) 課題番号 11480066) および、(株) 半導体理工学研究センターとの共同研究によるものである。

参考文献

[1] 中村 友洋, 吉瀬 謙二, 辻 秀典, 安島 雄一郎, 田中 英彦: "大規模データパスプロセッサの構想", 情報処理学会研究報告 97-ARC-124, pp.13-18 (1997)

[2] 高峰 信, 辻 秀典, 吉瀬 謙二, 坂井 修一, 田中 英彦: "VLDP アーキテクチャにおけるデータアクセスの軽減手法", 情報処理学会研究報告 99-ARC-133, pp.31-36 (1999)

[3] 安島 雄一郎, 坂井 修一, 田中 英彦: "複数パス実行のためのレジスタセット管理方式", 情報処理学会研究報告 99-ARC-133, pp.37-42 (1999)

[4] 吉瀬 謙二: "高レベル投機技術を用いた複数パス実行プロセッサ", 博士論文, 東京大学大学院 工学系研究科 (2000)

[5] 田中 洋介: "VLDP プロセッサにおける命令ブロック構成の研究", 修士論文, 東京大学大学院 工学系研究科 (2000)

[6] 塚本 泰通: "次世代マイクロプロセッサにおける命令ブロックの構成とフェッチ方式に関する研究", 卒業論文, 東京大学 工学部 (2000)

[7] 辻 秀典, 安島 雄一郎, 坂井 修一, 田中 英彦: "大規模データパス・アーキテクチャの提案", *SWoPP2000* (2000 予定)