3F−7

動的VLIW変換アーキテクチャの提案

高瀬 亮　近山 隆
{takase, chikayama}@logos.t.u-tokyo.ac.jp
東京大学 工学系研究科*

1 はじめに

半導体技術の進歩にともない, 単一プロセッサ内に複数の機能ユニットを実装して, 命令レベルで並列動作をさせる高速化が一般的になっている. この方式にはVLIW方式, スーパースカラ方式があるが, コンパイラの力を借りて静的依存性解析したコードを使用するVLIWは, 命令セットの非互換性からプロセッサ方式の主流となり得えていない現実がある. 一方のスーパースカラは動的な解析を行なうのにコストがかかり, 速度の点でVLIWより不利であるといわれている.

本稿では, コードの実行時に動的に命令依存性解析をしてVLIWへの変換を行なうことにより, ループ中において同一命令列を実行する際の実行性能を向上させることを目的とした, 動的VLIW変換アーキテクチャについて述べる.

2 スーパースカラとVLIW

プロセッサにおける高速化手法を命令レベル並列処理方式の点から見てみると, スーパースカラ方式とVLIW (Very Long Instruction Word) 方式の二つがその主なものになる.

スーパースカラは, 動的な命令依存性解析を行なうことにより, 命令セットアーキテクチャ互換で複数命令の同時実行が可能であるという利点がある. しかしながら, 実行時に依存性解析を行なうコストがかかる欠点もある. また, スーパースカラ方式では, 実行するコード中の繰り返し (ループ) 部分でも, ループ中の命令の依存性解析を各イタレーションで毎回行なっている. コストがかかる解析を何度も繰り返して行なっているわけであり, ループの2巡目以降の解析は無駄であるといえる.

この問題を解決する方法の一つにVLIW方式があるという見方ができる. VLIW方式は, 命令の依存性解析を静的に行ない複数の命令がパックされた長命令 (以下VLIW命令とする) を使用するので, スーパースカラのような実行時の解析コストがかからない. しかしVLIW命令を使用するため, 命令セットアーキテクチャは従来のものとは異なってしまう. 従来のバイナリコードが使えないという, 互換性の点での不利がある.

3 動的VLIW変換アーキテクチャ

3.1 動的VLIW変換

動的VLIW変換方式は, 命令セットの互換性を保ちつつ, 主にループを実行する際にVLIW的な動作を行なうものである. 2節で述べたスーパースカラ方式とVLIW方式の両者の長所を活かすことを目標とした方式であるといえる. 基本的な動作は,

- スーパースカラのように動的依存性解析を行なって, 命令レベルの並列度を取り出す. 命令セットアーキテクチャは従来と同じものとすることができる.
- 命令依存性解析結果を後に利用するため, テーブルに保存しておく.
- ループ (の2巡目以降のイタレーション) では, 前イタレーションにおける解析結果をテーブルから引いて用いることにより, 依存性解析を省略してVLIW的な実行を行なう.

のようになる. 従来の方式と動的VLIW変換方式の比較を表1にまとめておく. 同様な研究にはハイパースカラ・アーキテクチャ [1] があるが, 本研究では命令セットアーキテクチャを保存する点で異なっている.

	依存性解析	命令セット
スーパースカラ	動的	互換
VLIW	静的	非互換
動的VLIW変換	動的 (ループ時) 静的	互換

表1: 従来方式と動的VLIW変換方式

3.2 アーキテクチャ

図1に動的VLIW変換アーキテクチャの基本構成を示す. 基本的な構成はスーパースカラアーキテクチャ

* "*Proposal of a Dynamic VLIW Translation Architecture*"
Ryo TAKASE and Takashi CHIKAYAMA
School of Engineering, the University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113, Japan

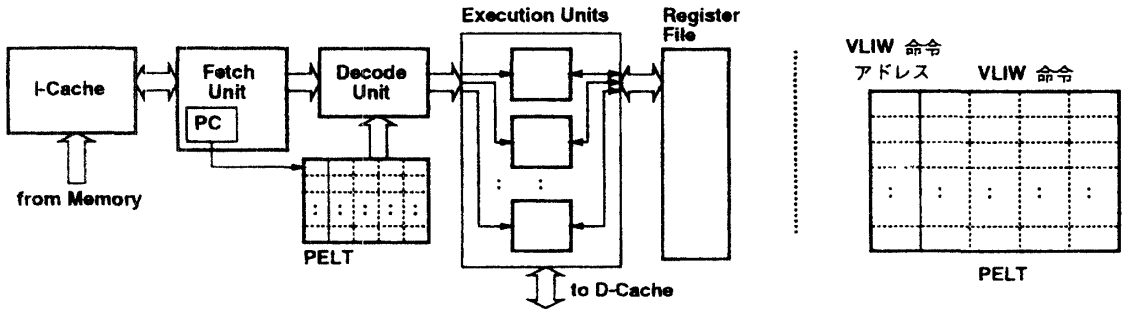

図 1: 動的 VLIW 変換アーキテクチャ

に類似したものとなる．図１の構成の例だと，フェッチユニットで命令をフェッチした後，デコードユニットで命令をデコードする．命令デコード時には，命令間の依存関係を解析して，命令レベルの並列度を取り出す．複数の実行ユニットでは複数の命令を並列に実行して，実行した結果をレジスタファイルないしメモリに書き出す．

動的 VLIW 変換アーキテクチャの特徴は，命令間の依存性解析結果を保存しておく PELT (Parallel Execution Look-aside Table) を設けたところにある．PELT は，VLIW 命令アドレスと，複数個の命令を一まとめにした VLIW 命令とを対応付けた表である．VLIW 命令アドレス一つと VLIW 命令一つの組を一ラインとして，複数のラインが存在する．

動的 VLIW 変換アーキテクチャの動作は次のようになる．まず，プログラムカウンタ (PC) を使って命令をフェッチする．フェッチの際はまず PELT を参照し，PC と PELT のアドレスが一致する (PELT にヒットする) かどうかによってその後の動作は二通りに分けられる．

PELT にヒットしない場合

PELT にヒットしない場合は，命令間依存解析を行ない，この結果から VLIW 命令を生成する (動的に VLIW に変換する)．生成された VLIW 命令は実行ユニットに送出されるとともに，VLIW 命令アドレス (VLIW にパックされた複数命令のアドレスのうち，最も若いアドレス) とともに PELT に書き込まれる．

PELT にヒットした場合

一方，PC と PELT のアドレスが一致した場合は，実行しようとしているコードの命令依存性解析結果は，既にそれより以前に行なわれている．このような状況は多くの場合，ループの中を繰り返し実行する時に現れるであろう．この場合は，命令キャッシュからの命令のフェッチを行なわず，PELT から VLIW 命令をデコードユニットに直接読み込み，これを直ちに実行ユニットに送って実行する．

本アーキテクチャでは，PELT にヒットした時の命令間依存性解析を省略して高速化を目指す．しかしながら，PELT にヒットしなかった際に VLIW の生成を待って，常に VLIW 命令を実行するというアプローチでは命令間依存解析がボトルネックになる可能性がある．この問題は，VLIW 命令の生成と，それよりはナイーブな解析による実行とを同時並行的に行なうことにより解決できると考えられる．

4　おわりに

本稿では，主に繰り返し実行の際の高速化と，命令セットアーキテクチャの保存を目標とした，動的 VLIW 変換アーキテクチャの概要について述べた．このアーキテクチャで高速化を目指すためには，どれだけの範囲の命令依存性解析を行ない動的 VLIW 変換するかという問題と，VLIW 実行時の速度を落とさないこととのバランスを取ることが必要である．また，PELT の容量の検討や管理手法の工夫によるヒット率の向上なども鍵になる．今後は，ループの内部における命令レベル並列度について調査した上で具体的なアーキテクチャを仮定し，シミュレーションを行なっていく予定である．

参考文献

[1] 弘中，斎藤，宮嶋，村上，“ハイパースカラ・プロセッサ・アーキテクチャ—プロトタイプの設計および性能評価—”，並列処理シンポジウム JSPP'94 論文集，pp. 9–16, May 1994.