4G－3

マルチスレッド処理をサポートする VLIWプロセッサ・アーキテクチャ

†國領 琢也　‡富田 眞治
†富士通(株)　‡京都大学工学部

1　はじめに

近年の高性能マイクロプロセッサは、動作周波数が飛躍的に高速になると共に、命令レベルの並列処理を行うことにより性能向上を図っている。プロセッサの高速化のためには、パイプラインピッチを短縮するためのスーパパイプライン方式に加えて、命令の並列実行のためにスーパスカラ方式・VLIW方式を導入することが必要となっている[1]。

しかしながら、命令間の依存関係や階層メモリのアクセスによりプロセッサの並列実行が阻害され、プロセッサの持つ潜在的な並列度を引き出すことができず、現状では大幅な性能向上を達成することができない[2][3]。

本研究では、従来の単一命令ストリームを実行するプロセッサに代わって、複数命令ストリームを共有のパイプラインにおいて実行可能とすることにより、並列度を引き出すプロセッサ・アーキテクチャを提案する。

2　プロセッサ・アーキテクチャ

2.1　特徴

本アーキテクチャの主な特徴は、以下の通りである。

1. 静的パイプライン・スケジューリング
   (ノンインタロック・パイプライン)
2. 静的命令発行(VLIW方式)
3. スーパパイプライン構造
4. 共有パイプラインによるマルチスレッド処理

1, 2の特徴は、命令レベルの並列処理とマルチスレッド処理において必要となるハードウェア量を小さく抑えるために考慮した。図1にプロセッサのハードウェア構成図を示す。命令フェッチユニットをはじめとするパイプライン処理ユニットは、複数の命令ストリームにより共有されており、ハードウェア・コストは、スレッドの数だけレジスタ・ファイルを用意する必要がある以外は、VLIWプロセッサと大差がない。

2.2　マルチスレッド処理

本アーキテクチャのマルチスレッド処理では、同一パイプライン中において複数の命令ストリーム(スレッド)の命令が実行される。パイプラインに投入するスレッドの選択は、パイプラインの最初のステージにて行われ、1サイクルごとにスレッドを切り替えることが可能である。スレッドの切り替えは、以下の要因に対して行われる。

1. データ依存による次命令の実行を抑止する場合

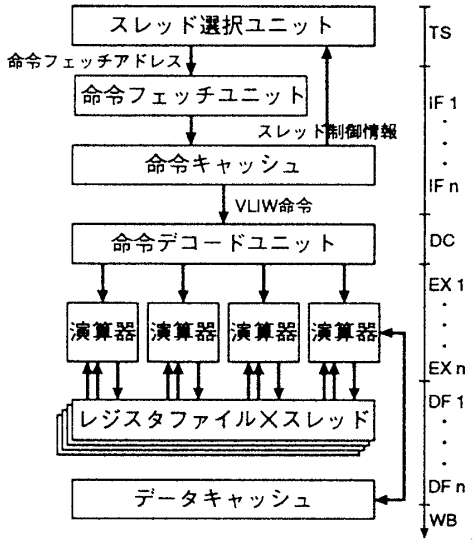

図1: プロセッサの構成図

2. 分岐命令の遅延スロットを埋められない場合
3. 内蔵キャッシュのミスヒットが発生した場合

1, 2については、コンパイル時に検出可能な要因であり、VLIW命令中の特定フィールド(後述)によってスレッド切り替えを指示する。3についてはプロセッサが動的に検出し、スレッドを切り替える。

2.3　命令パイプライン

命令パイプラインは、大きく分類すると、スレッド選択 → 命令フェッチ → 実行という経路により処理する。単一命令ストリーム内における命令間の依存関係の解消は、最初のパイプライン・ステージであるスレッド選択ユニットにて行う。ここでは、コンパイル時に生成した依存解析情報を用いて、命令フェッチユニットに対して命令フェッチ要求を出力する。依存解析情報は、VLIW命令中のフィールド(4bit程度)に埋め込まれており、次に実行するVLIW命令に対する遅延サイクル数を指定する。フェッチした遅延サイクル数は、サイクルごとにカウントダウンし、その期間は、別のスレッドの命令フェッチ要求を出力するため、命令パイプラインはインタロックすることなく処理される。

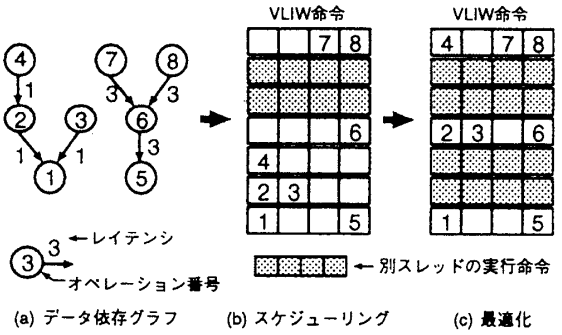

図 2: VLIW 命令のスケジューリング例

このように、VLIW 命令の遅延サイクル数フィールドの指定により、単一命令ストリームのパイプライン処理タイミング、およびマルチスレッド処理のスレッド切り替えタイミングを規定している。

2.4 VLIW 命令スケジューリング

基本ブロックの実行サイクル数は、そのコードのクリティカルパスに従う。このため、単一命令ストリームの命令スケジューリングは、クリティカルパスを延長しない限りは自由に VLIW 命令の組み替えを行っても性能に影響しない。したがって、クリティカルパスを延長させることなく、マルチスレッド処理の効果が高まるようにスケジューリングを行う。

命令のスケジューリング例を図 2に示す。(a) は、基本ブロック内の 2 本のデータ依存グラフであり、レイテンシの小さい命令から成るグラフとレイテンシの大きい命令から成るグラフを示している。(b),(c) は、命令をスケジューリングした結果であり、共にクリティカルパス長は同一であるが、VLIW コード密度の高い (c) の方がマルチスレッド処理時における性能の高いことが推測できる。

3 シミュレーション結果

図 3にソフトウェア・シミュレーションによる評価結果を示す。プログラムには、浮動小数点演算としてベクトルの乗算プログラムと整数演算として複数要素のバイナリサーチ・プログラムを使用した。パイプライン化された機能ユニットの構成は、表 1に示す。

それぞれ、スレッド数が増えるにしたがってパイプライン処理の空きが埋められ、プログラムの処理速度が向上している。ベクトルの乗算プログラムでは、ループ・アンローリングを行うことにより 2 スレッドの時にほぼ最高性能に達している。アンローリングを行わない場合は、性能を達成するために 4 スレッドを必要とする。同様に、バイナリサーチ・プログラムでも、オペランド・バイパスを行うことにより 2 スレッドの時にほぼ最高性能に達している。バイパスを行わない場合は、性能を達成するために 4 スレッドを必要とする。

このように、ループアンローリングやオペランド・バイパスを行うことにより、2 スレッドの並列処理により

機能ユニット	個数	実行／反復 サイクル数
整数演算	4	1 ／ 1
ロード/ストア	2	4 ／ 1
浮動小数点演算	4	4 ／ 1

表 1: 機能ユニットの構成

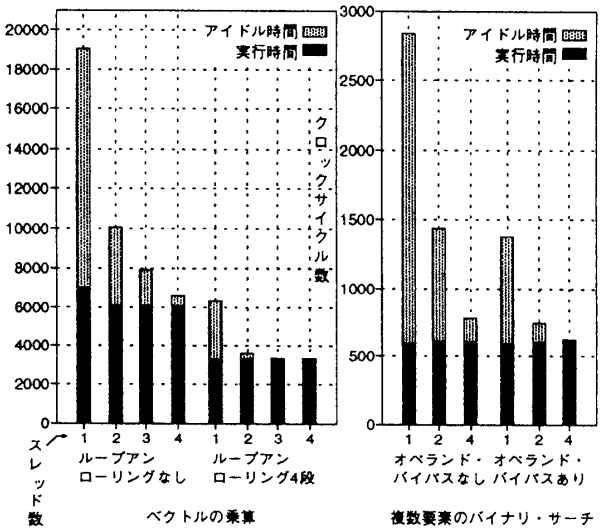

図 3: 実行サイクル数

パイプラインのスループットを十分に高めることが可能である。

4 おわりに

今後、スーパパイプライン処理技術と命令レベルの並列処理技術が進歩した場合に、両者から得られる並列度を単一命令ストリームにより有効利用することは困難である。このため、マルチスレッド処理を導入して並列処理性能を引き出すプロセッサ・アーキテクチャを提案し、その有効性を確認した。

参考文献

[1] Edward McLellan: The Alpha AXP Architecture and 21064 Processor, IEEE Micro,Vol.13, No.3,June 1993, pp.36-47.

[2] Norman P. Jouppi and David W. Wall: Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines, Third International Conference on Architectural Support for Programming Languages and Operating Systems, pp.272-282, 1989.

[3] Gurindar S. Sohi and Sriram Vajapeyam: Tradeoffs in Instruction Format Design for Horizontal Architectures, Third International Conference on Architectural Support for Programming Languages and Operating Systems, pp.15-25, 1989.