4P－2

# スーパースカラパイプラインによるブロック並列実行方式

高田 滋　朝生 良教　桐山 佳隆　林 達也

名古屋工業大学 電気情報工学科

## 1 はじめに

近年のマイクロプロセッサは集積度が上がり、また開発環境も充実しかなり複雑な設計まで行なえるようになった。プロセッサの演算性能もそれに比例して向上し、1サイクルで並列に複数の演算が実行できるスーパースカラ方式へと発展してきた。しかし命令間の依存関係などにより、並列に実行できる命令はそう多くはなく、分岐命令によるペナルティーも大きい。また命令を逐次実行していては、処理の高速化が図れない。

我々は、より高い並列度で実行し、かつ投機的に命令を実行する高速化手法の1つとして、ブロック並列実行方式を提案し、その実効性を検証する。

## 2 ブロック並列実行方式の概要

図1において、逐次実行ではA,I,J,Lの順に実行されるが、ブロック並列実行方式[1]では、Aブロックとその分岐先候補のB,Iブロックを同時に並列実行し、Bが途中で無効化される。Iブロックの実行が完了した時点で、次のブロックJ,K,Lの3つを同時に並列実行する。

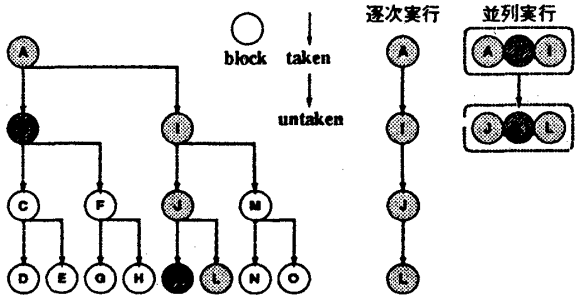


図1: ブロック実行方式の摸式図

## 3 ソフトウェアからの支援

順序性を保証しつつ、並列に投機的実行ブロックを実行するには、あらかじめデータの依存関係を把握する必要がある。親ブロックに依存する可能性のある子ブロックのデータにはコンパイル時に把握されて依存ビットが立てられている。子ブロックから見て親ブロックの候補は複数存在し得るので、可能性のある全ての依存関係を把握し、依存ビットを割付ける。実行時にはフラグにより、親ブロックでデータが生成されるまで参照を待たせる。

一方親ブロックでは、子ブロックが生成を待っているそのデータは、依存ビットに呼応する提供ビットがコンパイル時に付加されており、提供ビットの立っているデータが生成されたら、子ブロックへの参照が許可される。

この様に、全てのブロックについて、ブロック間で依存する可能性のあるデータを全て調べ、依存ビット・提供ビットをソフトウェアにより命令語に埋め込む。RISC命令セットにはまだ余裕があるので、これらの依存ビット・提供ビットは、オペランド毎に1bitのフラグを用意すればよい。従って、ソフトウェアで合計3bitのフラグを命令語に埋め込む。

親ブロックの候補が複数存在し得るので、依存ビットが立っているのに実際には依存が生じない場合も発生し得る。子ブロックでデータを待ち続けるのを回避するため、親ブロックが全命令をデコードしたら全ての依存ビットによるフラグを解消すべく信号を出す。

## 4 アーキテクチャ上の支援

各ブロックを実行するスーパースカラパイプラインは、命令レベル並列性を向上させるために4命令を同時にin-orderに発行し、実行はデータ依存関係を動的に解消するTomasuloアルゴリズム[2]を拡張適用してout-of-order実行を許す機構である。

拡張したTomasuloアルゴリズムによる、アーキテクチャの基本構成を図2に示す。レジスタファイル(Register File)と、それに対応する管理情報を記憶するレジスタ状態(Register Condition)、発行された命令を待機させるリザベーションステーション(Reservation Station)、ロードバッファ(Load Buffer)、ストアバッファ(Store Buffer)およびブロードキャストするため図中太線のコモン・データバス(CDB: Common Data Bus)から成る。ブロックを3つ並列実行するために、基本構成はこれらを3重化し、コモンデータバスで結んだものとなる。なお図中制

---

A Block Parallel Execution Method
in Superscalar Pipeline Architecture
Shigeru Takada, Yoshinori Asou, Yoshitaka Kiriyama,
Tatsuya Hayashi
Department of Electrical and Computer Engineering,
Nagoya Institute of Technology,
Gokiso-cho, Showa-ku, Nagoya, 466, Japan

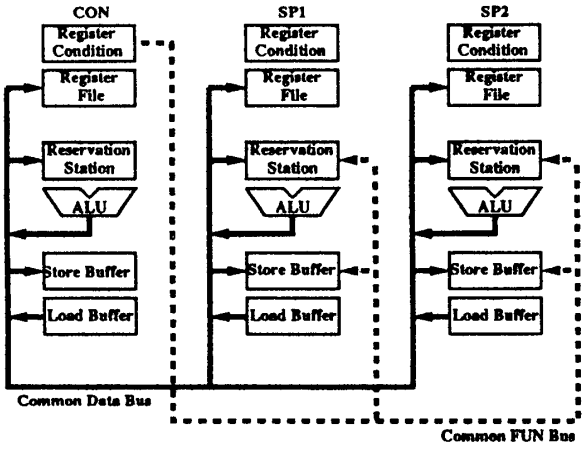


図 2: 基本構成

御機構とバスは省略してある。

各スーパースカラパイプラインは並列実行の組合せに固定的に対応し、当該実行ブロックを実行するものを当該実行スーパースカラパイプライン（図中 CON ）、投機的実行ブロックを実行するものを投機的実行スーパースカラパイプライン（図中 SP1,SP2 ）と呼ぶ。

また投機的実行スーパースカラパイプラインの持つレジスタファイルが投機的実行結果のバッファリングをし、並列実行の完了後に投機的実行ユニットのレジスタ状態と共に速やかに当該実行スーパースカラパイプラインのレジスタファイルとレジスタ状態に書き戻す。従って投機的実行ブロックの命令は必要なソースオペランドをレジスタから参照した後は、投機的実行ユニットのレジスタに格納し参照する。

また Tomasulo アルゴリズムではデータのタグとして機能ユニット番号 (FUN: Functional Unit Number) を利用して発行することで遅延を回避するが、ブロック並列実行の場合は依存するソースオペランドを変更する命令が未発行なため、機能ユニット番号を利用できない。従って当該実行ブロックで機能ユニット番号が決定され次第専用のバス、図中点線の共通ユニット番号バス (CFB: Common FUN Bus) を利用してブロードキャストを行なう。但しタグはレジスタ番号である。

これにより、ブロック内の依存関係は Tomasulo アルゴリズムを用いることで動的に解消され、ブロック間の依存関係は、依存ビット・提供ビットというソフトウェアの支援を受け、順序性を保ちながら並列に実行される。

当該実行ブロックの分岐命令がリターン命令であった場合は、投機的実行ブロックの先頭アドレスは動的に変化するので、静的には知ることはできない。

しかしアドレス自体はリターン・アドレスを保持するレジスタに入っているはずなので、現在実行しているブロック組が終り次のブロック組に遷移する時に、アドレスを動的に読むことができる。従って分岐命令をコーディングする際に投機的実行ブロックの先頭アドレスは動的にレジスタから読むように特殊なコーディングを施すことで、この問題は回避できる。

## 5 実験

mips R3000 アーキテクチャと pixie ユーティリティにより、ベンチマークにはブロック間の依存関係が顕著なクイックソートを使用し、プロセッササイクルレベルでのトレースを行なった。なお CDB の多重度やリザベーションステーション内のバッファなどのリソースは、無限大としてシミュレートした。

## 6 結果

全く最適化していないスカラ命令列では、CDB の平均必要多重度は 3.35 であることがわかった。

また、必ず分岐する無条件分岐命令が、分岐命令全体の 34% にのぼることがわかった。当該実行ブロックが無条件分岐命令であった場合、投機的に実行できるブロックは、分岐先の一つのみである。

また、分岐不成立側のブロックは当該実行ブロックの直後に位置するので、当該実行ブロックの延長として、いわば自動的にフェッチされる場所である。更に条件分岐命令の分岐頻度が 50% を上回ることを考慮すると、分岐不成立側を投機的に実行することは、ハードウェアの負担が大きい割には効率が悪いと思われる。分岐不成立側の投機的実行をしない場合のトレースを行ない、もう一度性能評価をするのが今後の課題である。

また、メモリ操作命令に関して現在のところ、検討中である。クイックソート・ベンチマークの場合、命令の 3 割はメモリ操作であり、それらの高速化は必須であるが、Tomasulo アルゴリズムではメモリアドレス幅の 32bit のタグを使用することは不可能である。従って親ブロックの最後のストア命令に提供ビットを付加し、子ブロックの全てのメモリアクセス命令に依存ビットを立てることで、メモリアクセスの論理的な順序整合を保証している。しかしロード命令の頻度を考えると、効率の改善が必要である。

## 参考文献

[1] 朝生 良教, “スーパースカラパイプラインによるブロック並列実行方式”, 情報処理学会研究報告 94-ARC-106, pp.1-8, 1994

[2] David A. Patterson, John L. Hennessy(富田, 村上訳),“Computer Architecture : A Quantitative Approach.” Morgan Kaufmann Publishers Inc. （日経 BP 社）, 1992