『新風』プロセッサのための静的コードスケジューリング

6X-6

入江直彦 久我守弘 村上和彰 富田眞治
(九州大学大学院総合理工学研究科)

1 はじめに

我々は,低レベル並列処理とパイプライン処理とを融合したプロセッサアーキテクチャとしてSIMP(単一命令流/多重命令パイプライン)方式を提案し,この方式に基づく試作プロセッサ『新風』を開発している[1].『新風』では,与えられた命令流に内在する並列性を最大限に活かすために,動的コードスケジューリングを行っている.しかし『新風』の性能を更に引き出すためには,コンパイル時における広範囲な並列性抽出,すなわち,静的コードスケジューリングが必要となる.

本稿では,『新風』に必要な静的コードスケジューリングについて述べる.

2 静的コードスケジューリングの目的

2.1 静的コードスケジューリング

静的コードスケジューリングとは,与えられたプログラムに対してその意味を保存しながら,実行時間/コードサイズを最小にするために,目的とするデータ構造に変換することである.スケジューリングの対象となるコード,あるいは,オペレーションをここではOPと呼ぶ.スケジューラへの入力は,基本ブロックをノードしノード間のアークが制御依存関係を表わすようなフローグラフと,各OPをノードとしノード間のアークがデータ依存関係を示すようなデータ先行DAG(Directed Acyclic Graph)の形で与えられるとする(図1参照).データ先行DAGにおいて,データの生産者になる命令を先行者(predecessor)と呼び,データの消費者となる命令を後続者(successor)と呼ぶ.

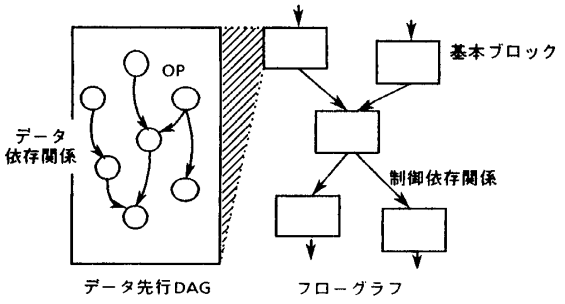

図1 プログラムの構造

2.2 VLIW方式における静的コードスケジューリング

Trace Scheduling[2]に代表されるようなこれまでの静的コードスケジューリング技法は,主としてVLIWプロセッサを対象としている.VLIWプロセッサ用のコードスケジューラは以下の要求を達成する必要がある.

①ハザードの回避

VLIWプロセッサは動的にハザードを検出する機構を持たない.したがってデータ依存,および,制御依存に起因するハザードに対しては,OPの処理時間まで考慮して静的に対処する必要がある.

②資源の競合の回避

VLIWプロセッサは並列動作可能なユニットが非均質構造であるため,同時に実行可能なOPについては事前に資源の競合を回避しておく必要がある.

③並列性の抽出(=コンパクション)

VLIWプロセッサでは,並列実行可能な単位が一つの超長形式命令として明示的に与えられる.したがって,並列性を抽出することは一つの命令にどれだけ多くのOPを埋め込めるかということと等価である.

VLIWプロセッサにおける静的コードスケジューラが用いるデータ構造を図2に示す.すなわち,行方向には並列実行動作する機能ユニットを示すフィールドが並び,列方向にはスケジューリングのためのスロットが与えられる.各スロットはVLIWの一命令を表すと同時にマシンサイクルをも反映している.

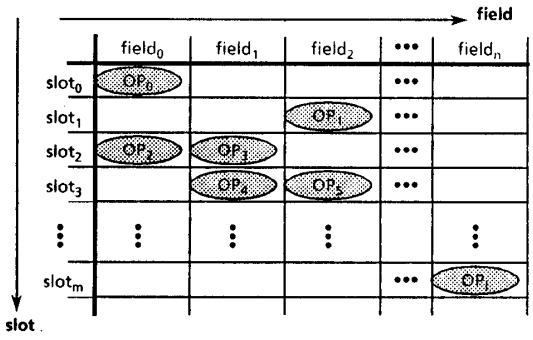

図2 スケジューリングに用いるデータ構造(VLIW)

スケジューリングは

(1) $slot(OP_i) > slot(predecessor(OP_i)) + latency(predecessor(OP_i))$

(latency : OPが実行するのに必要なサイクル数)

(2) if $field(OP_i) = field(OP_j)$ then $slot(OP_i) \neq slot(OP_j)$

の条件を満たし,かつ,対象とする命令流(もしくは命令流の一部)に必要なスロット数を最小にするように,コードを(slot,field)で示される場所に配置する.

2.3 『新風』のための静的コードスケジューリング

『新風』においてスケジューリングの対象となるOPは命令と等価である.2.2で述べた要求を 新風 の静的コードスケジューラに当てはめて考えると以下のようになる.

①ハザードの回避について

『新風』プロセッサは,内部にハザードを回避するための機構を持っており,静的コードスケジューラは依存関係のある命令についてはその順序を守るだけでよい.

②資源の競合の回避について

Static Code Scheduling for the [ʃímpu:] Processor
Naohiko IRIE, Morihiro KUGA, Kazuaki MURAKAMI, and Sinji TOMITA
Kyushu University

「新風」において，並列実行可能なユニット(=命令パイプライン)は均質構造をしており，静的には全く資源の競合を考慮せずに済む．

③並列性の抽出について

「新風」において並列実行する単位は，命令供給機構がプロセッサに投入する単位である命令ブロックであるが，分岐命令の出現や「新風」の命令アラインメント機構などにより命令ブロックは静的には定まらない．よって，VLIWでのスケジューリングのような，サイクルに対応したスロットを前提としたスケジューリングを行うことはできない．

「新風」はその内部で，局所データ駆動に基づく実行制御を採る．したがって，命令の発火するタイミングはその命令が必要とするオペランドの状態によってのみ定まる．しかも，制御依存関係を考慮したデータ先行DAGを内部で生成するため，実行の可否が判明しないうちに発火を許可する(条件付き先行実行)．こういった機構により，静的コードスケジューラは明示的なNOPを挿入する必要がない．

「新風」での静的コードスケジューラが用いるデータ構造は，図3のような二つの形態をとる．(a)は並列抽出のためのデータ構造であり，(b)は実際に生成する命令流に対応するデータ構造である．(a)については，NOPを挿入する必要がないためデータ先行DAGの構造をそのまま反映できる．また，行方向に関する制約はない．列方向についてはVLIWにおけるスロットの代わりにDAGのレベル(=高さ)を導入する．レベルは，依存関係にある命令の相対的な距離を示す．

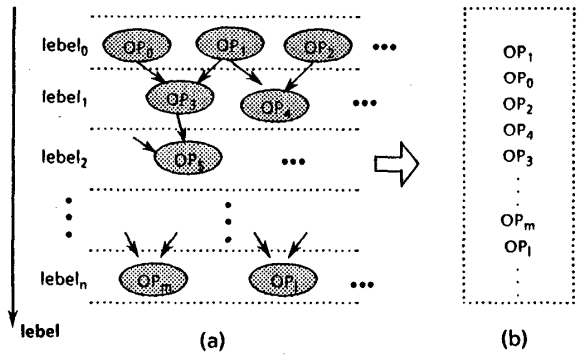

図3 スケジューリングに用いるデータ構造(「新風」)

スケジューラは，与えられたプログラムについてデータ先行DAGを作成し，これを(a)の形態に"スケジュール"する．さらに，ノードの優先順位に従って(b)で示される命令流に変換する．優先順位の設定については，DAG中のクリティカルパスや命令のlatencyを考慮する．変換後の命令流は，並列処理のための情報を一切持っておらず，この意味において従来のプロセッサで用いられているような命令流の形式と全く同一である．「新風」にとっては，データ依存関係のある命令間により多くの独立な命令が挿入されていることに意味がある．「新風」にとって最適な命令流は，この間にある命令数が最大であるような命令流である．

3 広域コード移動アルゴリズムの検討

並列を抽出する際に基本ブロック内のみでデータ先行DAGを生成するとこれより得られる並列性は非常に限定される．したがって基本ブロックを越えたコードスケジューリング(広域コードスケジューリング)が必要となる．広域スケジューリング技法については，Trace Schedulingを始めとして種々の技法[2][3][4]が提案されている．これらについて，コード移動のアプローチという観点から整理する．

①基本ブロックを無視したデータ先行DAGを作成する

これは，Trace Schedulingなどのアプローチである．この手法の利点は，広域スケジューリングが局所スケジューリング(基本ブロック内スケジューリング)と同じ形で行なえることである．一方，この手法で問題となるのは，基本ブロックを無視する範囲をどう設定するかである．Trace Schedulingでは，最も通り易いパス(これをTraceと呼ぶ)を選択する．しかし，選択した範囲がどの程度妥当性があるかによって，実際の性能が大きく左右される．さらにこの手法では，スケジューリングの後の補正コードの挿入(bookkeeping)が問題となる．bookkeepingのためには，スケジューリング後のコードについて，分岐のjoin，および，移動したコードの探索が必要であるが，この処理はかなり複雑となる．

②基本ブロック間で，明示的にコードの移動を行なう

これは，Percolation Scheduling[4]などで用いられている手法である．Percolation Schedulingではコード移動規則として，(1)OPの移動，(2)条件分岐の移動，(3)削除，(4)単一化，の4つを定義している．この手法の利点は，制御構造を意識してコードの移動を行なうため，bookkeepingが複雑になるのを防ぐ．また①のアプローチに比べ，コード移動の範囲があまり限定されない．さらには，分岐の確率にあまり依存しないようなスケジューリングが可能となる．一方，この手法の問題点は，移動させるべきコードの探索，および，そのコードの配置アルゴリズムをどうするかである．①では，この部分がリストスケジューリングなどにより自然な形で行なえたのに対して，複雑になる．

「新風」では，対象とするアプリケーションを限っておらず，したがってデータにかなり依存した条件分岐を含むようなアプリケーションに対しても，ある程度の性能が要求される．したがって，②のアプローチを採る．

4 おわりに

以上，「新風」に必要な静的コードスケジューリングの概要について述べた．今後の課題としては，基本ブロックを越えたコード移動アルゴリズムの確立とこれによる性能向上の評価，さらには，高級言語を意識した最適化コンパイラの開発があげられる．

参考文献

[1] K.Murakami, et al. : SIMP(Single Instruction stream / Multiple instruction Pipelining): A Novel High-Speed Single-Processor Architecture, *Proc. 16th Int'l Symp. Comp. Arch.*, 1989

[2] J.A.Fisher : Trace Scheduling:A Technique for Global Microcode Compaction, *IEEE Trans. Comp.*, vol.c-30, No.7, 1981

[3] B.Su and S.Ding : Some Experiments in Global Microcode Compaction, *Proc. 18th Int'l Microprogramming Workshop*, 1986

[4] A.Aiken and A.Nicolau : A Development Environment for Horizontal Microcode, *IEEE Tracs. Soft. Eng.*, vol.14, No.5, 1988