1G－3

統合型並列化コンパイラ・システム －局所コード・スケジューリング技法－

入江直彦　音成 幹　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

現在我々は，統合型並列化コンパイラ・システム開発の一環として，命令レベル並列処理を行うスーパースカラ・プロセッサ『新風』[1] を対象の一つとした最適化コンパイラの開発を進めている．本稿では，最適化の主要部分の一つである局所コード・スケジューリング技法について述べる．

2. 最適化コンパイラの構成

最適化コンパイラは以下の4つのフェーズから構成される．

① トランスレーション：
プログラム・グラフを，仮想コードレベルの中間コードに変換する．中間コードには，オペレーション，オペランドといった最低限必要な情報の他に，制御依存情報やデータ依存情報が付加される．

② 広域最適化：
ループ内不変式の移動，共通部分式の削除といった従来の最適化手法に加え，基本ブロック間に存在する並列性を抽出するため，基本ブロックの範囲を越えたコードの移動（広域コード移動）を行う．

③ コード生成：
『新風』のコード生成，および，レジスタ割付けを行う．

④ 局所最適化：
基本ブロックの配置，および，基本ブロック内コードの配置（局所コード・スケジューリング）を行う．

3. 静的コード・スケジューリング

『新風』は均質構造の命令パイプラインを4本実装して，単一命令流を並列に処理する．また，プロセッサ内部で拡張Tomasuloアルゴリズムに基づく動的コード・スケジューリングを行ない高速化を図っている．しかし，動的に並べ換えられる命令はプロセッサ内部に存在する命令のみに限定されるため，プログラム全体に内在する並列性を利用するためにコンパイル時において静的コード・スケジューリングを行う．

静的コード・スケジューリングは広域コード移動と局所コード・スケジューリングから構成されるが，これらをレジスタ割付けの前に行うか（Prepass Scheduling），後に行うか（Postpass Scheduling）が問題となる．Prepass Schedulingは，使用できるレジスタの個数に制限されることなく柔軟にスケジューリングを行なえるという利点がある．一方，Postpass Schedulingは，レジスタspillコードの数を最小に抑えることができる．したがってここでは，広域コード移動に関しては，並列性の抽出を主な目的とするため，レジスタの個数に制限されないようレジスタ割付けの前に行い，また局所コード・スケジューリングについては，レジスタspillコードに対処するため，レジスタ割付けの後に行なうようにする．

広域コード移動アルゴリズムについては，コードの移動先を明示的に指定することで，分岐命令の実行に偏りのないプログラムについても柔軟に対処できるようなアルゴリズムを検討している[2]．このアルゴリズムは，『新風』のハードウェア構成に依存せず，基本ブロック内の“潜在的な”並列性を増加させるようなコード移動を行うため，他の命令レベル並列処理プロセッサについても対応可能である．

4. 局所コード・スケジューリング

広域コード移動によって並列性の向上したプログラムを実際のプロセッサ上で効率よく処理するには，VLIWプロセッサやRISCプロセッサを対象に行なわれてきたような，ターゲット・マシンの属性を意識した局所コード・スケジューリングが必要である[3][4]．

4.1 『新風』における動的要因への対処

『新風』は内部で種々の動的制御を行う．これらは処理の高速化に寄与する反面，局所コード・スケジューリングの妨げとなる．特に以下の2点については考慮が必要である．

① 命令ブロック・アラインメント：
命令ブロック・アラインメントとは，命令供給時に命令の並べ換えを行うことである．これにより命令パイプラインの使用効率向上が期待できるが，静的コード・スケジューラにとっては命令ブロック（すなわち，並列処理単位）が静的に固定されないという弊害が生じる．このため，この機構を実効的に無効とするようなコード生成，すなわち，基本ブロックの先頭コードを4ワードバウンダリに配置するようなコード生成を行う．

② 動的コード・スケジューリング：
動的コード・スケジューリングにより，パイプライン内のコードはその投入順序や制御依存の有無に関係なく，フロー依存にのみ従い実行開始（発火）可能となる．この制御によりフロー依存関係にある2命令間の距離（先行コードが発火してから後続コードが発火するまでのサイクル数）が動的に変化する．したがって，静的コード・スケジューリングを行う際には，(a) 基本ブロックの先頭でliveであるレジスタの値は，当該基本ブロックの先頭コードが発行された時点で使用可能である，(b) 基本ブロックの直前の分岐命令は，当該基本ブロックの先頭コードが発行された時点では既にコミットされている，といった仮定を設ける．

4.2 スケジューリング・アルゴリズムの概要

局所コード・スケジューリング・アルゴリズムにおいては
(1) コード間の適正な順序関係（先行制約）をどう表現するか
(2) コードをスケジュールする際の順序をどう決定するか
(3) ターゲット・マシンをどう表現するか
といった点が問題となる．本スケジューラでは，以下のように対処する（図1参照）．

(1) 2つのコード間のデータ先行制約を示すDAG（DDDAG：Data Dependency Directed Acyclic Graph）を用いる．データ先行制約には，①フロー依存（RAW），②逆依存（WAR），③出力依存（WAW）が存在する．②，③は，レジスタ割付けをスケジューリングの前に行なっているため生じる．ただし，これら

An Integrated Parallelizing Compiler System : Local Code Scheduling Algorithm
Naohiko IRIE, Miki OTONARI, Kazuaki MURAKAMI, and Shinji TOMITA
Kyushu University

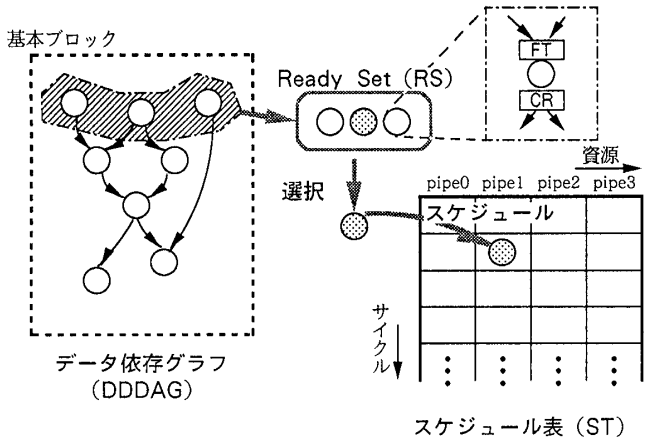

図1 局所コード・スケジューリングの概念図

は『新風』では命令パイプラインの乱れの要因とならないため，①とは区別して取り扱う．

(2) 最適スケジュールを導出するにはコード数の指数オーダの時間を要するため，発見的な手法に頼る必要がある．ここでは，DDDAGにおける幾何的順序，各コードの発火可能タイミング(FT : Firable Timing)，および，各コードの後続コードに与える影響(CR : CRitical value)といった情報を基に順序付けを行う．

(3) VLIWマシンと同様に，行方向を資源(命令パイプライン)，列方向をサイクルとする二次元配列(ST : Scheduling Table)で表わすことができる．ただしVLIWマシンと異なり，行方向にコードを配置する際に資源の競合を考慮する必要がない．

4.3 スケジューリング・アルゴリズムの詳細

局所コード・スケジューリング・アルゴリズムは図2に示すように以下の過程から成る．

① スケジューリング対象となる候補コードの抽出：
与えられたDDDAGにおいて先行コードを持たないものを候補コードの集合(RS : Ready Set)に加える．

② RSの中における最も最適な候補の選択：
まず，最も早く発火する可能性のあるもの，すなわち，FTの最も小さいものを選択し，サイクル番号を求める．FTはそのコードの先行コードがスケジュールされた時点で設定される．FTの同一のものが複数存在するときは，影響を与える後続コードの多いコード，すなわち，各コードにおけるCRの大きい方を選択する．CRはDDDAGの生成時に以下の式により設定される．

CR (i) = Σ {iの後続ノードのCR}
+ (iの実行に要するサイクル数)

③ スケジューリング：
与えられたサイクル番号，パイプライン番号に対応するSTのフィールドにコードを配置する．空のまま残ったフィールドはno-opとなる．

④ 後続コードのFTの設定：
スケジュールされたコードの後続コードについて，FTを以下の式に基づき設定する．

if (依存関係がフロー依存)
FT=max {現在のFT, (先行コードがスケジュールされたサイクル番号 + 先行コードの実行サイクル数)}
else
FT=max {現在のFT, 先行コードがスケジュールされたサイクル番号}

```
procedure Local__Code__Scheduling
   INPUT : ある基本ブロックにおけるDDDAG
   OUTPUT : スケジュールされたコード列
{
 RS : スケジューリング対象となるコードの集合
 ST〔Number__of__Pipeline〕〔Cycle__Max〕: スケジュール表

 Append__to__RS (DDDAG) ;
 /* 候補コードの抽出 */
 cycle=0 ;
 while (RSが空でない) {
   pipe__num=0 ;
   while (pipe__num < Number__of__Pipeline) {
     target=Select__Code (RS)  ;
     /* RSの中から最適なコードを選択 */
     if (target.FT ≦ cycle) {
       Schdule__to__Pipeline (target, ST, pipe__num, cycle) ;
       /* 対象コードをスケジューリングする */
       set__FT__of__successors (target, cycle) ;
       /* 対象コードの後続コードのFTを設定 */
       delete__from__DDDAG (target, DDDAG) ;
       /* 対象コードをDDDAGから除く */
       Append__to__RS (DDDAG) ;
       pipe__num=pipe__num+1 ;
     } else
         break  ;
   } /* end while */
   cycle=cycle+1 ;
 } /* end while */
 load__balance (ST) ;  /* 負荷分散 */
} /* end procedure */
```

図2 局所コード・スケジューリング・アルゴリズム

⑤ 負荷分散：
スケジューリング後はパイプライン番号の若いものの方にコードが偏っているため，コードが各パイプライン均等に配置されるよう，行単位でno-opの移動を行う．また，全てのフィールドがno-opである行は削除する．

5. おわりに

以上，『新風』を対象とした局所コード・スケジューリング技法について述べた．今後は，プログラムにおけるループ構造を対象としたスケジューリング・アルゴリズムの検討，および，これらアルゴリズムの評価を行う予定である．

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 ISCA, pp.78-85, May 1989.

[2] 入江ほか : SIMP方式に基づくスーパースカラ・プロセッサ『新風』のための静的コード・スケジューリング技法，情処研報，89-ARC-79-6 (1989年11月)．

[3] S.Davidson et al. : Some Experiments in Local Microcode Compaction for Horizontal Machines, IEEE Trans. on Comp. Vol. c-30, No.7, pp.460-477, July 1981.

[4] P.B.Gibbons and S.S.Muchnick : Efficient Instruction Scheduling for a Pipelined Architecture, ACM SIGPLAN'86 Symp. on Compiler Const., pp.11-16, June 1986.