修士論文

スラック予測を用いた
クラスタ型スーパースカラ・プロセッサ向け
命令ステアリング

指導教員　　富田 眞治 教授

京都大学大学院情報学研究科
修士課程通信情報システム専攻

福山 智久

平成19年2月2日

スラック予測を用いた
クラスタ型スーパースカラ・プロセッサ向け命令ステアリング

福山 智久

内容梗概

我々の研究室では，命令のスラック (slack) に基づくクリティカリティ予測を提案している．ある命令の実行を s サイクル遅らせてもプログラムの実行時間が増大しないとき，s の最大値をその命令のスラックという．このスラックを履歴に基づいて予測を行う．つまり，前回の実行時のスラックを予測表に登録しておくことによって，それを今回の予測値とすることができる．

本稿では，スラック予測をクラスタ型スーパースカラ・プロセッサの命令ステアリングに応用する方法を提案する．クラスタ型スーパースカラ・プロセッサでは，実行ユニットをクラスタと呼ばれる複数のグループに分割し，クラスタ間のオペランド・バイパスを省略する．オペランド・バイパスの配線長の短縮により，配線遅延も短縮され，動作周波数を高めることができる．一方，クラスタ間データ通信による遅延や特定クラスタへの命令の集中によってプロセッサの IPC（Instruction Per Cycle）は低下する．そのため，クラスタ型スーパースカラ・プロセッサでは，どの命令をどのクラスタで実行するか選択する，いわゆる命令ステアリングという処理が重要となる．

スラック予測を用いた提案命令ステアリングでは，スラックの予測値がクラスタ間通信遅延以上であれば，依存関係を無視し負荷が最小のクラスタへステアリングするなどして，クラスタの負荷を分散させる．スラックが大きな命令はその命令の実行が通信遅延分遅れても，後続の命令の実行には影響しない．そのような命令を負荷の小さなクラスタへステアリングすることで，通信遅延による性能への影響を抑えた負荷分散が期待できる．

シミュレーションによる評価の結果，実効幅が 8 命令のプロセッサを 8 つのクラスタに分割しクラスタ間のデータ通信遅延が 1 サイクルとした場合において，クラスタ化されていないプロセッサと比較した場合，提案方式は約 6.5 % 程度 IPC が低下することが分かった．これは既存ステアリング方式より約 5 % の IPC 向上となる．

Instruction Steering for Clustered Superscalar Processor with Slack Prediction

Tomohisa Fukuyama

Abstract

Our laboratory have proposed an instruction criticality prediction technique based on a prediction of instruction slacks. When the execution time of a program doesn't become longer even if an instruction of the program is delayed by s cycles, the maximum of s is referred to as the slack of the instruction. We have proposed a history-based slack predictor. That is, the slack value is stored to the prediction table to be a predicted value for the next time.

This paper describes an instruction steering scheme for a clustered superscalar processor with the slack prediction. In the clustered superscalar processor, execution units are partitioned into some groups called cluster, and the bypasses between clusters are left out. By reducing wire delays with short bypasses, the higher clock rate processor is achieved. On the other side, IPC (Instruction Per Cycle) is decreased because of the communication delays between clusters and the reduction of the issue width per cluster. Accordingly, an instruction steering, mechanism that determines the assignment of instructions to clusters, is important for the clustered superscalar processor.

With the instruction steering scheme with slack prediction, the instructions whose slacks are more than the communication delays between clusters are steered to the lowest workload cluster. This scheme is an effective for the load balancing of clusters. If the instructions with large slacks are delayed, the execution time of a program doesn't become longer. Therefore, the steering scheme with the slack prediction probably enable load balancing without performance decrease.

The evaluation result by simulation shows that the instruction steering scheme with the slack prediction improves IPC by 5% compared to the existing instruction steering scheme.

スラック予測を用いた
クラスタ型スーパースカラ・プロセッサ向け命令ステアリング

目次

第1章　はじめに **1**

第2章　スラック予測 **5**

2.1　クリティカル・パス 5
2.2　スラック予測器 7
2.2.1　スラック予測器の構成 7
2.2.2　スラック予測器の動作 8
2.2.3　スラック予測器へのアクセス・タイミング 9
2.2.4　条件分岐命令 11
2.2.5　スラック表，メモリ定義表 のミス 11
2.2.6　グローバル分岐履歴 12
2.3　第２章のまとめ 12

第3章　クラスタ型スーパースカラ・プロセッサ **14**

3.1　プロセッサの構成 14
3.2　Wakeup 遅延と Select 遅延 16
3.2.1　Wakeup と Select 16
3.2.2　Wakeup 遅延 17
3.2.3　Select 遅延 18
3.2.4　集中と分散のバランス 19
3.3　既存の命令ステアリング 19
3.3.1　Round-Robin 型ステアリング 20
3.3.2　Dependence Based ステアリング 20
3.3.3　発行時間差に基づいた命令ステアリング 21
3.4　第３章のまとめ 23

第4章　スラック予測を用いた命令ステアリング **24**

4.1　スラックの利用 24
4.2　スラックの計算 25
4.3　スラック予測を用いた提案命令ステアリング方式 27

4.3.1 Path Based ステアリング 27
4.3.2 スラックの予測値を発行時間差とする方法 31
4.3.3 スラック表にカウンタを付加する方法 33
4.4 第 4 章のまとめ 35

第 5 章 評価 36

5.1 評価環境 36
5.2 評価結果 39
5.3 考察 43

第 6 章 おわりに 44

謝辞 46

参考文献 47

第1章　はじめに

命令の**クリティカリティ**(criticality)，すなわち，命令がどれほどクリティカルかを知ることは，スーパースカラ・プロセッサの高性能化と省電力化の両方に効果がある．

例えば，命令をスケジューリングするときには，よりクリティカルな命令を優先的に発行した方がよい．小林らは，クラスタ化された演算器を持つプロセッサに対して，クリティカル・パスの情報を用いたスケジューリング手法を提案している [1]．また，クリティカルでない命令のみを低速 / 低消費電力の演算器で実行することで，性能を大きく低下させることなく省電力化を図ることができる [2, 3, 4, 5, 6]．

さて，従来クリティカリティに関する研究の多くは，プログラムのクリティカル・パスに基づいて行われてきた．しかし，クリティカル・パスに基づく方法には，以下のような 3 つの問題点がある：

1. **論理的**　キャッシュ・ミスや演算器不足など，実行しているプロセッサの物理的な制約が反映されていない．
2. **二値的**　最もクリティカルな命令を教えるのみで，それ以外の命令がどの程度クリティカルで**ない**のか判定できない．
3. **クリティカル・パスの判定が困難**　実行中のプログラムのクリティカル・パスを判定することはそれほど容易ではない．

そこで我々の研究室では，クリティカル・パスではなく，命令の**スラック**(slack)[7] を履歴に基づいて予測することにより，命令のクリティカリティを測ることを提案している [8, 9]．ある命令の実行を s サイクル遅らせてもプログラムの実行時間が増大しないような s の最大値をその命令のスラックという．したがって，クリティカルな命令のスラックは 0 サイクルである．

図 1 に，命令が実行される様子を表すタイム・チャートを示す．図中，“I” が命令の実行を表し，“I” の長さはその命令の実行レイテンシを表す．上下の “I” の間にある横線は，フロー依存関係を表す．図では，命令 I_d が定義した結果を**最初に**参照する命令は I_u である．提案するスラック予測器は，I_d がデータを定義した時刻と I_u がデータを参照した時刻の差を I_d のスラックと近似する．つまり，I_1 のスラック s は，原則的には，定義命令 I_d による定義時刻 t_d と，使用命令 I_u に

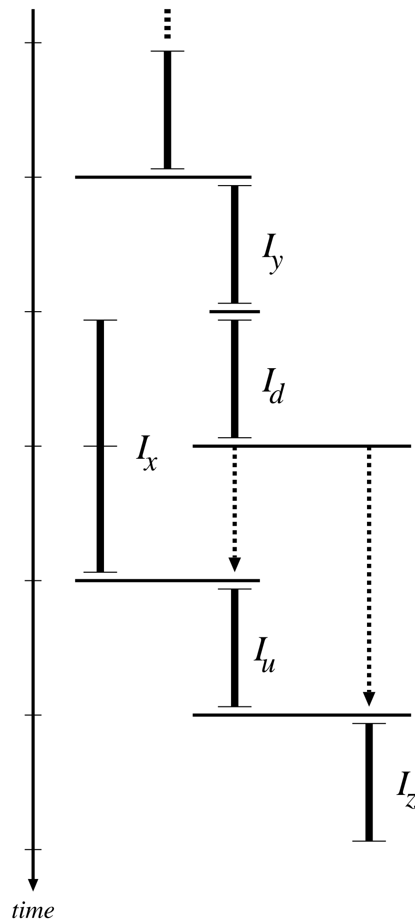

図 1: 命令実行のタイム・チャート

よる使用時刻 t_u の差:

$$s = t_u - t_d - 1 \tag{1}$$

によって得られる．この式に従えば，I_d のスラックは 1 サイクルとなる．また，I_z も I_d が定義した結果を参照しているが，I_u が先に I_d を参照しているので I_d のスラックは 2 サイクルとはならないことに注意されたい．なお，以下ではスラックの単位を省略し，「I_d のスラックは 1」のように言うことにする．

このようにして求めたスラックの値を予測表に記憶しておき，次回実行時に予測表を読み出すことによりスラックの予測を行う．

スラック予測によりクリティカリティ予測を行うことで，前述したクリティカル・パスに基づく方法の問題点は以下のように解決されると期待できる:

1. **実効的**　履歴に基づいてスラックを予測するので，物理的，実効的なクリティカリティが反映される．

2. **多値的**　クリティカリティの大 / 小は，スラックの小 / 大によって多値的に表現される．
3. **スラックの判定は容易**　クリティカル・パスとは異なり，スラックは容易に求めることができる．

本稿では，スラック予測を用いた高速化へのアプローチとして，クラスタ型スーパースカラ・プロセッサ [10] における命令ステアリングに，スラック予測器による予測結果を用いる方法について述べる．

近年のプロセッサは主に，パイプラインを深くすることと半導体技術の微細化により動作周波数を高め，性能を向上させてきた．しかし，今後さらに微細化が進むと，配線遅延がゲート遅延に対して相対的に増大し，支配的になることが知られている．中でも，実行ユニットの出力と入力を接続するバスであるオペランド・バイパスは，その配線長がほぼ実行ユニット数に比例する．そのため，配線遅延の影響を強く受け，プロセッサの動作周波数を高める上での足枷となっている．この問題に対して注目されている技術が実行ユニットのクラスタ化である [10].

クラスタ型スーパースカラ・プロセッサでは，実行ユニットをクラスタと呼ばれる複数のグループに分割し，クラスタ間のオペランド・バイパスを省略する．このため，クラスタ内のバイパスは約 1 /（クラスタ数）に短縮される．配線遅延は配線長の 2 乗に比例するため，2 つのクラスタに分割するだけでも，大幅な遅延の短縮が可能となる．このため，高速なバイパシングが可能となり，プロセッサの動作周波数を高めることができる．また，クラスタ型スーパースカラ・プロセッサは性能の面だけでなく，消費電力面の有利さからも注目されている [11].

一方，実行ユニットのクラスタ化により，IPC（Instruction Per Cycle）は低下する．依存関係にある命令を別々のクラスタで実行した場合，オペランド・バイパスが利用できない．そのため，同一クラスタで実行した場合に比べ，クラスタ間データ通信に数サイクル余分に遅延が生じ，後続の依存命令の実行が遅れてしまう．また，個々のクラスタはクラスタ化されていないプロセッサに比べスループットが小さい．そのため，特定のクラスタに命令が集中することにより，後続の依存命令は実行資源が獲得できず実行が遅れてしまう．よって，クラスタ型スーパースカラ・プロセッサでは，依存関係にある命令を同一クラスタにステアリングし，なおかつ各クラスタの負荷が均一になるようなバランスの良い命令ステ

アリング方式が求められる [1, 12, 13] .

そこで，この命令ステアリングにスラック予測器による予測結果を応用することを考える．たとえば，クラスタ通信遅延が C サイクルの場合，スラックが C 以上の命令は，依存する先行命令とは別のクラスタで実行しても，命令の実行は遅れないことになる．このように，スラック予測を応用することで，より高精度な命令ステアリングが期待できる．

以下，まず第 2 章でスラック予測と本稿で用いた予測器の構成などについて説明し，次に第 3 章で本稿で想定するクラスタ型スーパースカラ･プロセッサの構造と IPC の低下要因となる Wakeup 遅延と Select 遅延について説明し，既存のステアリング方式について説明する．そして，第 4 章 でスラックを用いた命令ステアリングについて述べ，第 5 章 で提案方式の評価を行う．

第2章　スラック予測

冒頭でも述べたように，命令のクリティカリティに関する情報を知ることは，スーパースカラ・プロセッサの高性能化と省電力化の両方に効果がある．

これまで，命令のクリティカリティに関する研究の多くはプログラムのクリティカル・パスに基づいて行われてきた．これに対し，我々の研究室では，命令のスラックを履歴に基づいて予測することにより，命令のクリティカリティを測ることを提案している．

本章では，まず2.1節でクリティカル・パスに基づいたクリティカリティに関する研究とその問題点について述べた後，2.2節で本稿で用いるスラック予測器について説明する．

2.1　クリティカル・パス

クリティカリティに関する研究の多くは，以下のように，プログラムのクリティカル・パスに基づいている．

小林らは，クラスタ化されたプロセッサ[14]の命令スケジューリングのため，パス情報テーブル（PIT: Path Information Table）を提案している[1]．このテーブルは，命令ウィンドウ内にある命令に対して近似的なデータ・フロー・グラフを構築し，そのグラフの最長パス，そして，そのパスの先頭にある命令を答えることができる．

Tuneらは，適当なヒューリスティクスに基づいて，『クリティカル・パス上の命令らしさ』を推測している[4]．ヒューリスティクスとしては，例えば，「命令ウィンドウ内で，最も古い命令（QOLD）」とか「命令ウィンドウ内で，その結果が最も多くの命令に使用される命令（QCONS）」などといったものである．Tuneらは，命令ごとのローカルな履歴のみを用いたクリティカル・パス予測器を評価している．また，千代延らはいわゆるグローバル履歴を用いた2レベル予測器を評価している[15]．

しかし，これらのクリティカル・パスに基づく手法には以下のような3つの問題がある：

1. **論理的**　データ・フロー・グラフに現れるようなプログラムの論理的な制約のみを考慮しており，実行しているプロセッサの物理的な制約を考慮していない．そのため，以下のような食い違いが生じる：

ミス　キャッシュ・ミスを起こすロード命令，分岐予測ミスを起こす条件分岐命令などは，やはり論理的にはクリティカルではなくても，実効的にクリティカルになる可能性が高い．

資源制約　資源制約，特に演算器の個数が考慮されていない．例えば，同じ演算器を使用する命令が多数連続する場合，それらの命令が論理的にはクリティカルでなくても，実効的にクリティカルになり得る．

メモリを介した依存　小林らの方法も Tune らの方法も，メモリを介した依存を考慮できていない．

2. **二値的**　最もクリティカルな命令を教えるのみで，それ以外の命令がどの程度クリティカルで**ない**のか判定でない：
   - 2 番目にクリティカルな命令を優遇しなかった結果として，その命令が実質的にクリティカルになることがあり得る．
   - ミス・ペナルティの大きい技術には適用できない．例えば DVS 制御では，電圧の制御に時間がかかるため，単にクリティカルでないと予測されただけでなく，非常にクリティカルでないと予測されなければ適用できない．
3. **クリティカル・パスの判定が困難**　プログラム全体，あるいは，関数などのクリティカル・パスを求めることに対して，実行中のプログラムのクリティカル・パスを判定することは，原理的に不可能と言ってもよい；実行中のプログラムの場合，データ・フロー・グラフの始点と終点を定めることができないからである．

   また，データ・フロー・グラフは一般的なアサイクリック・グラフであり，ハードウェアで取り扱うにはやや複雑過ぎる．そのため，精度とハードウェア・コストのバランスをとることが難しい：
   - 小林らのパス情報テーブルは（命令ウィンドウ内の命令に限れば）クリティカル・パスをかなり正確に推定できるものの，複雑なハードウェアを必要とする．
   - Tune らの用いているヒューリスティクスは，実装は比較的容易であるが，実際にクリティカル・パス上の命令を正しく判定できない [6].

2.2　スラック予測器

前節では，クリティカル・パスに基づいたクリティカリティに関する研究について述べ，それらの方法には 1. 論理的，2. 二値的，3. クリティカル・パスの判定が困難といった問題点があることを説明した．

そこで我々の研究室では，クリティカル・パスではなく，命令のスラックを予測することにより，命令のクリティカリティを測ることを提案している．

本節では，スラック予測器の基本的な実装について述べる．以下，まず 2.2.1 項でスラック予測器のデータ構造についてまとめた後，2.2.2 項，2.2.3 項で予測器に対する登録，参照といった操作とアクセス・タイミングについて説明する．また，2.2.4 項以降では実装上の考慮点について考慮する．

2.2.1　スラック予測器の構成

スラック予測器は，主に**スラック表**と**定義表**の 2 種の表からなる．

スラック表　スラック表は，命令の過去のスラックを記録する予測表本体であり，値予測における VHT（Value History Table）に相当する．

定義表　定義表は，各データに対し，以下を記録する：

1. **定義時刻**　そのデータが定義された時刻
2. **定義命令**　そのデータを定義した命令

定義表は，スラック表に記録するスラック自体を計算するために用いられる．論理的には，レジスタ・ファイルやメモリ上の各データに対して，定義時刻，定義命令を記録するフィールドを付加したものと考えてよい．データが使用されるとき，データと同時に定義表に記録された定義時刻を読み出せば，定義命令のスラックを計算することができる．

ただし，システム中のすべてのデータに対して定義時刻，定義命令，クラスタ番号を記録することは非現実的である．予測精度とハードウェア・コストのバランスをとるためには，アクセス頻度の高いロケーションに対してエントリを提供することが肝要である．

アクセス頻度を考慮して，定義表は，データの格納場所がレジスタかメモリかによって 2 つに分け，それぞれ以下のように実装する：

1. **レジスタ定義表**　物理レジスタ番号をインデクスとする RAM によって構成する．すなわち，そのエントリ数は物理レジスタ・ファイルに等しく，まさに物理レジスタ・ファイルに定義時刻，定義命令を格納するフィールドを付加した

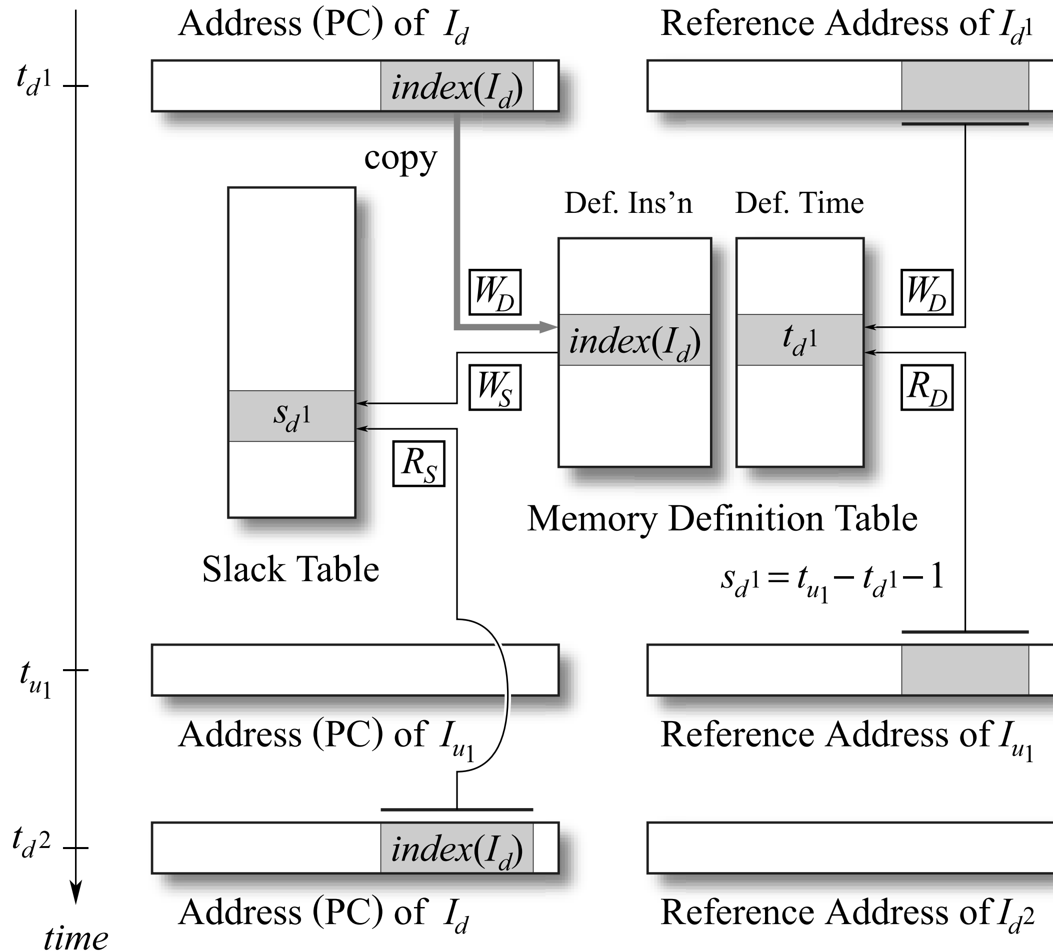

図 2: 予測器に対する操作（ストア命令の場合）

ものと考えてよい．

2. **メモリ定義表**　ロード / ストア命令の参照アドレスをインデクスとするキャッシュとして構成する．

したがって，メモリ定義表ではミスが発生することになる．メモリ定義表のミスや，エントリ数と連想度については後で詳しく述べることにして，次節では，これらの表を用いたスラック予測器の動作について述べる．

2.2.2　スラック予測器の動作

以下では，命令 I_d の n 回目 $(n \in \mathbf{N})$ の実行を，肩付き数字を用いて，I_{d^n} のように表すことにする．また，I_{d^n} が定義したデータを最初に使用する命令の実行を I_{u_n} と表す．なお，I_{d^i} と I_{d^j} $(i, j \in \mathbf{N},\ i \neq j)$ は同じ命令であるが，I_{u_i} と I_{u_j} は一

般に異なる．

図 2 では，ストア命令 I_{d^1} とロード命令 I_{u_1} が，それぞれ，時刻 t_{d^1} および t_{u_1} に実行されている．スラック表への登録，および，同スラック表への参照，すなわち，予測は，以下の様に行われる：

1. **登録**　登録は，以下のように，(a) I_{d^1} がデータを定義するときと，(b) I_{u_1} がそのデータを使用するとき の 2 つのフェーズからなる：
   (a) **定義**　I_{d^1} がデータを定義するとき，以下の操作が行われる：
   W_D　現時刻 t_{d^1} と I_{d^1} 自身（のアドレス）を定義表に書き込む．
   (b) **使用**　I_{u_1} がそのデータを使用するとき，以下の 2 つの操作が連続して行われる：
   R_D　定義表を読み出して，定義時刻 t_{d^1} と定義命令 I_{d^1}（のアドレス）を得る．
   W_S　定義時刻 t_{d^1} と現時刻 t_{u_1} から，I_{d^1} のスラック s_{d^1} が，$s_{d^1} = t_{u_1} - t_{d^1} - 1$ と求まる．スラック表の定義命令 I_{d^1} のエントリに，求めた s_{d^1} を書き込む．
2. **予測**　命令 I_d が再びフェッチされると，以下の操作が行われる：
   R_S　I_d のアドレスをインデクスとしてスラック表を直接読み出すことで，前回のスラック s_{d^1} が得られる．

ストア命令以外の場合には，基本的には，メモリ定義表をレジスタ定義表に，参照アドレスを物理レジスタ番号に，それぞれ読み替えればよい．

なお，スラックの計算を行うのは，そのデータが最初に使用されるときのみである．したがって，R_D において，読み出された定義表のエントリを無効化し，以降同じエントリが繰り返し読み出されないようにする．このことは，メモリ定義表のエントリの有効利用にも効果がある．

2.2.3　スラック予測器へのアクセス・タイミング

図 3 および図 4 に，前述したストア命令の場合のと，それ以外の命令の場合の，予測器へのアクセスのタイミングを示す．図中の W_D〜R_S は，前述した操作と対応している．スラック予測器へのアクセス・タイミングは，以下のとおりである：

1. **登録**　ストア命令の場合には，定義側，使用側共に，アドレス計算（図 3 中，A）の次のステージからアクセスを開始できる．
   一方，ストア以外の命令の場合，表へのインデクスとして用いる物理レジス

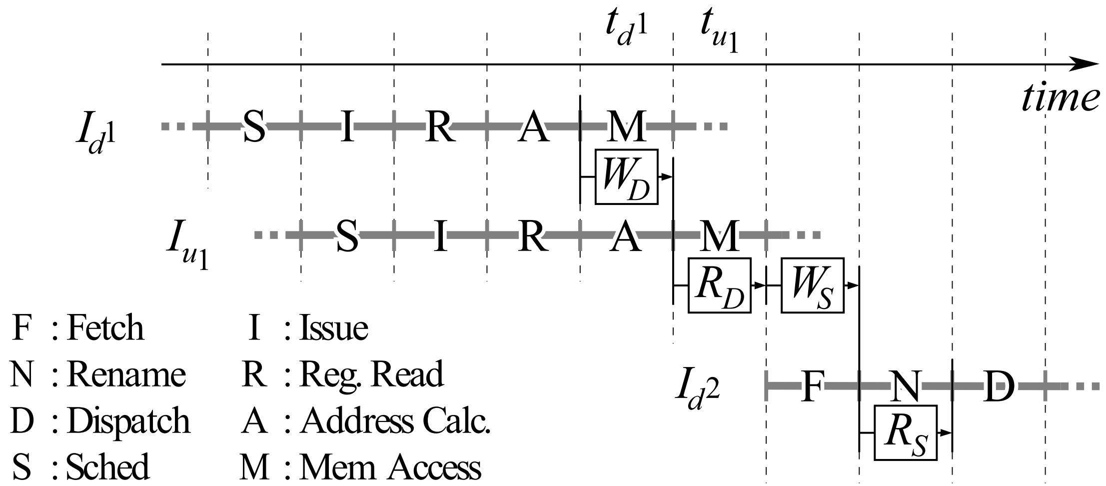

図 3: 予測器に対する操作のタイミング（ストア命令の場合）

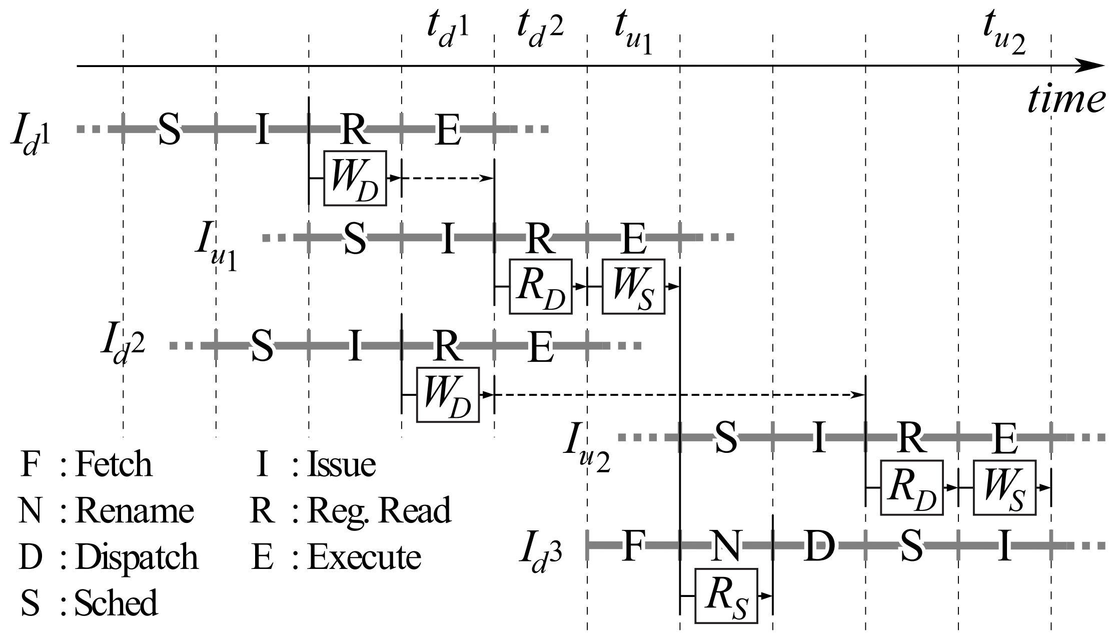

図 4: 予測器に対する操作のタイミング（ストア命令以外の場合）

タ番号が既に分かっているため，1 サイクル早く，発行（図 4 中，I）の次のステージからアクセスを開始できる．

ストア以外の命令の場合に，実際に 1 サイクル早くアクセスを開始した方がよいか，ストア命令に合わせた方がよいかは，プロセッサの基本的な設計，特に，様々な投機ミスからのリカバリの方法に依存する．

2. **予測**　分岐予測器，値予測器などと同様，命令のアドレスをもってアクセスす

ればよいので，命令フェッチと同時にアクセスを開始することができる；ただし，得られたスラックは専ら命令スケジューリングに使用されるため，ディスパッチ（図3，D）に間に合うように読み出せばよい．

2.2.4　条件分岐命令

分岐命令は，その実行結果を参照する命令が存在しないため，上述の方法でスラックを計算することはできない．そこで，以下に述べる理由により，分岐予測ミスを起こした分岐命令のスラックは0，ヒットした分岐命令のスラックは0または1とする．

分岐予測ミスを起こす分岐命令は，できるだけ早く実行した方がよい．分岐予測ミスを起こす分岐命令より下流にある命令の実行はすべて無駄である．その分岐命令を早く実行すると，この無駄が省かれるとともに，状態回復がそれだけ早く開始されるからである．

一方，分岐予測ヒットする分岐命令は，論理的にはクリティカルにはならない．しかし，以下に述べる理由から，できるだけ早く実行した方がよい; フェッチ後，未完了のまま (pending) にできる分岐命令の数には，分岐予測に関する資源の量に起因して，他の命令より強い制約がある．例えば，MIPS R10000 プロセッサ [16] の場合，たかだか4個の条件分岐命令しか未完了にできない．資源が不足した場合には，フロントエンドがストールすることになる．

このように分岐命令は，予測ヒット / ミスに関わらず他の命令より早く実行した方がよく，その中ではミスする命令の方がよりクリティカルである．例えば，分岐命令の実行ユニットが1つしか空いていない場合には，ヒットするものよりミスするものを先に実行した方がよい．したがって，このことを考慮したい場合には，ミスすると予測される命令のスラックは0，ヒットすると予測される命令のスラックは1とするとよいと考えられる．

2.2.5　スラック表，メモリ定義表 のミス

メモリ定義表，スラック表は，キャッシュであり，ミスが起こる．メモリ定義表，スラック表共に，W_D，W_S では，ミスが起こっても割り当てられたエントリに上書きするだけであるから，その時点でのミスは性能上影響がない．したがって，W_D，W_S で割り当てられたエントリが，R_D，R_S までにリプレースされてしまった場合のことを考えればよい．

R_D，R_S におけるミスは，以下のようにする：

メモリ定義表　定義表のエントリがリプレースされたのだから，定義側の命令

を特定することができないので，スラック表への登録を行うことは不可能である．

スラック表 スラック表ミス時には，スラックは 0 と予測するのが安全である．また，スラックが 0 である命令は全体の半分以上に上り，0 としておいても相応の予測ヒット率が期待できるからである．

2.2.6 グローバル分岐履歴

スラック予測器にグローバル分岐履歴を導入するには，gshare 分岐予測器などと同様にする方法がまず考えられる；すなわち，スラック表のインデクスとして，命令のアドレスとグローバル分岐履歴の排他的論理和を用いるのである．ただし以下で述べるように，スラック表はタグを持っていることに注意する必要がある．

分岐予測で用いられる PHT (Pattern History Table) には，競合が発生する；すなわち，通常 PHT はタグを保持しておらず，異なる 2 つの命令が同一の PHT エントリを使用することを許している．

一方，値予測で用いられる VHT では，タグ比較を行い，競合を許さないことが普通である．

スラック表も，VHT と同様，現在ではタグを保持している．グローバル分岐履歴を導入するにあたっては，インデクスに命令のアドレスとグローバル分岐履歴との排他的論理和を用いるとともに，タグにもグローバル分岐履歴を加え，競合を排除している．そのため，破壊的競合ではなく，ヒット率の低下によって予測精度が低下するおそれがある．

なお，本稿の評価におけるシミュレーションではスラック予測器で用いたグローバル分岐履歴の履歴長を全て 1 ビットとした．スラック予測器にグローバル分岐履歴を用いることで予測精度が向上することが認められている [9] が，その度合いはわずかで今回の評価にはほとんど影響が現れなかったためである．

2.3 第 2 章のまとめ

本章では，クリティカリティ予測のためのスラック予測について述べた．本章で説明したスラック予測器では，データを定義した時刻と使用した時刻の差により定義命令のスラックを計算し，そのスラックを予測表に登録しておくことにより，それを次回実行時の予測値とする．

スラック予測器を用いることで，クリティカル・パスに基づく手法における 1. 論理的，2. 二値的，3. クリティカル・パスの判定が困難といった問題点は以下のよう

に解決されると期待できる：

1. **実効的**　履歴に基づいてスラックを予測するので，物理的，実効的なクリティカリティが反映される：

   ミス　ロード命令がキャッシュ・ミスを起こすかどうか，条件分岐命令が分岐予測ミスを起こすかどうかには，履歴依存性があることが既によく知られている．そのような命令に対して，予測器は小さいスラックを出力すると期待できる．

   資源制約　同じ演算器を使用する命令が多数連続するような状況にも，少なからず履歴依存性があると推測される．そのような命令に対しても，予測器は小さいスラックを出力すると期待できる．

   メモリを介した依存　計算されたアドレスに基づいて，メモリを介した依存を考慮することができる．

2. **多値的**　クリティカリティの大 / 小は，スラックの小 / 大によって多値的に表現される．そのため，以下のようなことが可能になる：
   - 例えば，3 つの命令のスラックがそれぞれ 0，1，および，10 であった場合，最初の 2 つを優先するとよいだろう．
   - 命令ウィンドウ内に残っている命令のスラックがすべて大きい値，例えば 100 程度以上であった場合には，DVS 制御をかけてもよいだろう．

3. **スラックの判定は容易**　クリティカル・パスとは異なり，スラックは容易に求めることができる．

本稿では，このスラック予測器による予測結果をクラスタ型スーパースカラ・プロセッサにおける命令ステアリングに応用する方法について述べる．次章では，クラスタ型スーパースカラ・プロセッサの構造や既存の命令ステアリングについて説明する．

第3章　クラスタ型スーパースカラ・プロセッサ

近年のプロセッサは，半導体技術の微細化により，配線遅延がゲート遅延に対して相対的に増大している．中でも，実行ユニットの出力と入力を接続するバスであるオペランド･バイパスは，その配線長がほぼ実行ユニット数に比例する．そのため，配線遅延の影響を受けやすく，プロセッサの動作周波数を高める上での足枷となっている．

この問題に対して，DEC[1)] Alpha 21264 プロセッサ [14] などで採用されている実行ユニットのクラスタ化と呼ばれるアプローチが注目されている．実行ユニットのクラスタ化は，実行ユニットをクラスタと呼ばれるいくつかのグループに分割し，クラスタ間にまたがるオペランド･バイパスを省略する．これにより，オペランド･バイパスの配線長が短くなり配線遅延を短縮することができる．配線遅延は配線長の 2 乗に比例するため，2 つのクラスタに分割するだけでも，大幅な遅延の短縮が期待できる．

しかし，冒頭でも述べたように，クラスタ型スーパースカラ･プロセッサではクラスタ化されていないプロセッサと比べ，クラスタ間のデータ通信遅延と特定クラスタへの命令の集中によりプロセッサの IPC は低下してしまう．このため，クラスタ型スーパースカラ･プロセッサでは，命令をどのクラスタで実行するか選択する，いわゆる命令ステアリングと呼ばれる処理がその性能を左右する．

本章では，まず 3.1 節で本稿で想定する一般的なクラスタ型スーパースカラ･プロセッサの構造について述べた後，3.2 節でクラスタ型スーパースカラ･プロセッサにおける IPC 低下要因となる Wakeup 遅延と Select 遅延について説明する．そして，3.3 節で既存の命令ステアリングについて述べる．

3.1　プロセッサの構成

図 5 に一般的なクラスタ型スーパースカラ･プロセッサの構成を示す．以降，評価などで用いるプロセッサのモデルは，本節で説明するクラスタ型スーパースカラ･プロセッサを想定するものとする．

図 5 に示すように，想定するクラスタ型スーパースカラ･プロセッサでは，実行ユニットを複数のクラスタに分割し，クラスタ間のオペランド･バイパスを省略する．このため，クラスタ内のバイパスは約 1 /（クラスタ数）に短縮され，高速な

[1)] 発表当時.

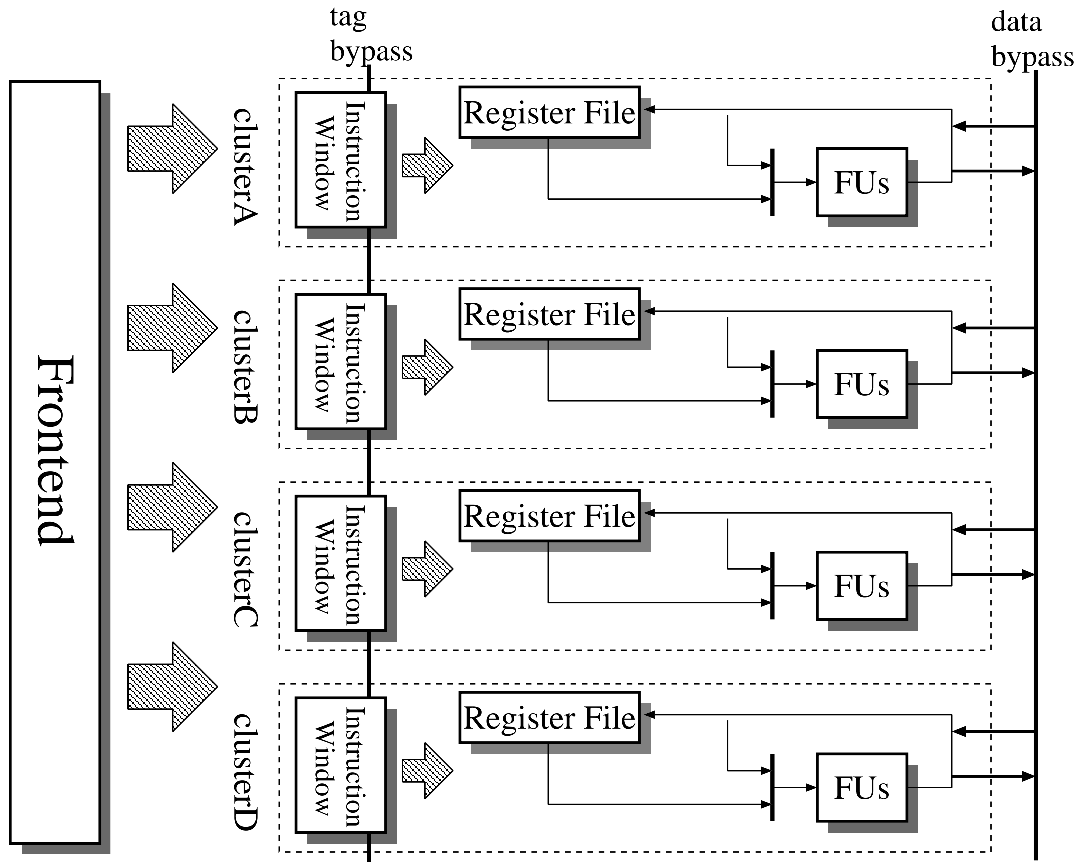

図 5: クラスタ型スーパースカラ・プロセッサ

バイパシングが可能となる．一方，クラスタ間のデータ通信にはクラスタ間バスを用いる．ただし，このクラスタ間バスを用いた通信は，クラスタ内のバイパスを用いた場合より数サイクル余分に遅延が生じることになる．

命令の実行結果は，実行されたクラスタにはクラスタ内のバイパスを通じてすぐに反映されるが，異なるクラスタへの反映はクラスタ間バスの通信遅延分だけ遅れる．このため，依存関係にある 2 つの命令が異なるクラスタに割り当てられた場合には，引き続くサイクルに命令は連続して発行することができない．

また，図 5 に示すように，命令ウィンドウも各クラスタに分散させている．命令の発行時には，後続命令のオペランドが利用可能になること（Wakeup）の判断のために，命令ウィンドウの全エントリに発行された命令の命令タグをブロードキャストする．命令ウィンドウも各クラスタに分散させることにより，分散させない場合に比べ，命令ウィンドウ自体のサイズ，各実行ユニットへの発行幅が共に小さくなり，高速に動作させることができる．しかし，クラスタ間の命令タグの送

信は，オペランド・バイパスによるデータ転送と同様，クラスタ内で行われる命令タグの送信と比べ，クラスタ間通信遅延分だけ余分に時間が掛かってしまう．

フロントエンドでフェッチ，デコード，リネームなどの処理をされた命令はいずれかのクラスタの命令ウィンドウにディスパッチされ，実行条件が揃えばクラスタ内の実行ユニットに発行される．発行された命令は，各クラスタに備わったレジスタから値を読み出し，演算を行う．

本稿で想定するクラスタ型スーパースカラ・プロセッサでは，命令ステアリングに焦点を絞るため，比較的実装が容易な複製分散型レジスタ方式を採用した．この方式では，全てのクラスタがレジスタ・ファイルを保持し各クラスタのレジスタ・ファイルが同じ内容を持つ．あるクラスタでレジスタが更新されるときには，その内容を他の全てのクラスタにブロードキャストする．レジスタ読み出しはクラスタ内でのみ行われるため，レジスタの読み出しポート数を減らすことができる．一方，各クラスタでの更新内容を全てのクラスタにブロードキャストするためクラスタ間通信が増大したり，プロセッサにおけるレジスタ・ファイルの占める面積が大きいなどの問題がある．

3.2 Wakeup 遅延と Select 遅延

前節では，一般的なクラスタ型スーパースカラ・プロセッサの構造を述べた．クラスタ型スーパースカラ・プロセッサは，小さな命令ウィンドウと高速なバイパシングにより，クラスタ化されていないプロセッサと比べ，高周波数で動作できる．しかし，その一方で，プロセッサの IPC は低下してしまう．

本節では，クラスタ型スーパースカラ・プロセッサにおける IPC 低下要因について述べる．まず，3.2.1 項で命令パイプラインにおける Wakeup フェーズ，Select フェーズでの処理について述べ，3.2.2 項，3.2.3 項で Wakeup 遅延，Select 遅延について説明する．

3.2.1 Wakeup と Select

図 6 に，ステアリング以降の命令パイプラインの様子を示す．EXEC は命令の実行を，RF →と RF ←はレジスタ・ファイルに対する読み出しと書き込みを表し，縦破線はクロックを表す．ただし，図は命令パイプラインの乱れが生じず，命令が滞りなく実行された場合のものを表している．

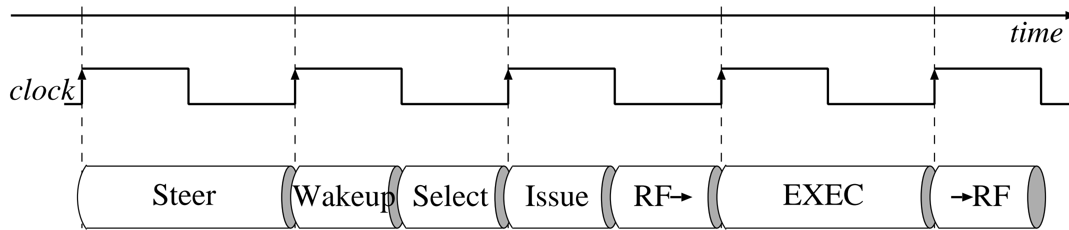

図 6: 命令パイプライン

Wakeup フェーズ

各クラスタへステアリングされた命令は，Wakeup フェーズで実行時に使用するソース・オペランドが全て利用可能な状態となっているかチェックをされる．このチェックの結果，ソース・オペランドが全て利用可能である場合，その命令は Wakeup され，同サイクル内に Select フェーズへと移る．逆に，ソース・オペランドが全て利用可能でない場合には，次のサイクルに，再び同様のチェックが行われる．

Select フェーズ

実行ユニットなどの演算資源の数には限りがあるので，命令が Wakeup されてもすぐさま実行できるわけではない．Select フェーズでは，Wakeup フェーズで検出された実行可能な命令の中から，実際に発行する命令の選択が行われる．命令が使用する演算資源を獲得できれば，その命令は次のサイクルに発行され，さらに 1 サイクル後に実行されることが決定的となる．演算資源が獲得できない場合には，獲得できるまでストールする．

なお，命令が Select された場合には，その命令が生成するデータが利用可能になることを後続命令に知らせるために，命令タグを命令ウィンドウにブロードキャストする．その結果，命令ウィンドウ内の命令は他に利用可能になっていないソース・オペランドがなければ，次のサイクルで Wakeup される．

Wakeup 遅延，Select 遅延とは，プロセッサをクラスタ化することにより，それぞれ Wakeup フェーズ，Select フェーズで生じる遅延である．

3.2.2 Wakeup 遅延

データの Producer が発行されてから Consumer が Wakeup されるまでの時間差を Wakeup 遅延と呼ぶ．Producer と Consumer を異なるクラスタで実行する場合は，命令タグの送信がクラスタ間通信遅延分だけ遅れることになる．

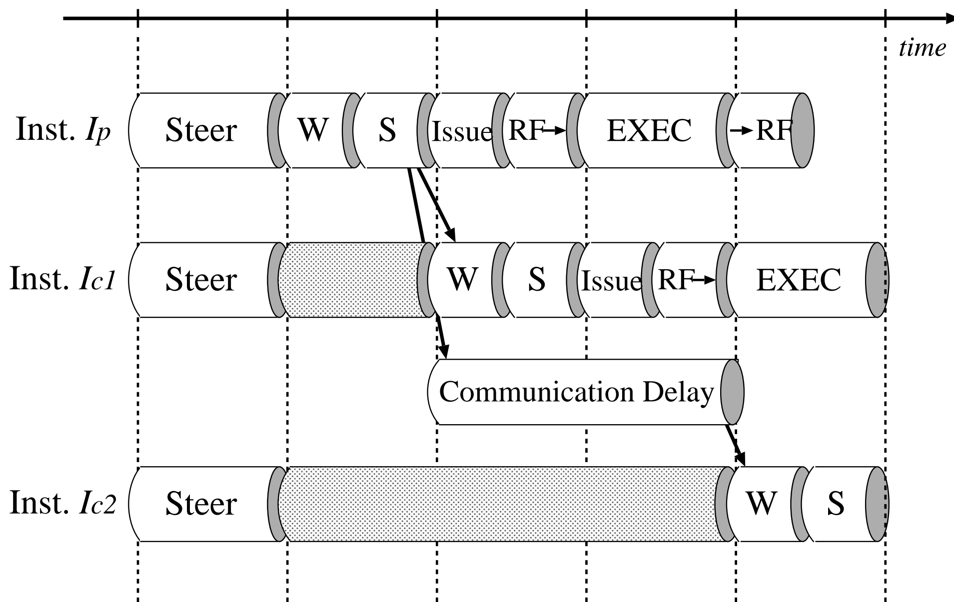

図 7: Wakeup 遅延

その様子を図 7 に示す．図中，W は Wakeup，S は Select の略で，命令間の矢印はフロー依存を示している．I_p はデータの Producer となる命令を，I_{c1}，I_{c2} は Consumer となる命令を表し，I_p と I_{c1} は同じクラスタで実行され，I_{c2} はそれとは別のクラスタで実行された様子を表す．I_p と同じクラスタで実行される I_{c1} は I_p の Select された次のサイクルで Wakeup される．一方，I_p とは別クラスタで実行される I_{c2} は，クラスタ間通信遅延分（図では 2 サイクル）だけ Wakeup 遅延が発生する．なお，I_{c1} に比べ，I_{c2} が I_p の実行結果を使用できる時刻もクラスタ間通信遅延分だけ遅れるが，その通信遅延による新たな実行の遅延は発生しないことに注意されたい．

よって，Wakeup 遅延を解消するには，依存関係のある命令同士は同じクラスタへ集中させるのが望ましい．

3.2.3　Select 遅延

命令が Wakeup されてから Select されるまでの時間を Select 遅延と呼ぶ．一般に，クラスタ化されたプロセッサは，クラスタ化されていない通常のプロセッサと比べ，クラスタ毎の命令発行幅が小さい．このため，同一クラスタの命令ウィンドウに複数の Wakeup 済みの命令が集中している場合，すぐに演算資源を獲得できず Select 遅延が発生する．

この様子を図 8 に示す．命令 I_x，I_y，I_z の間には依存関係はなく，I_x，I_y は同

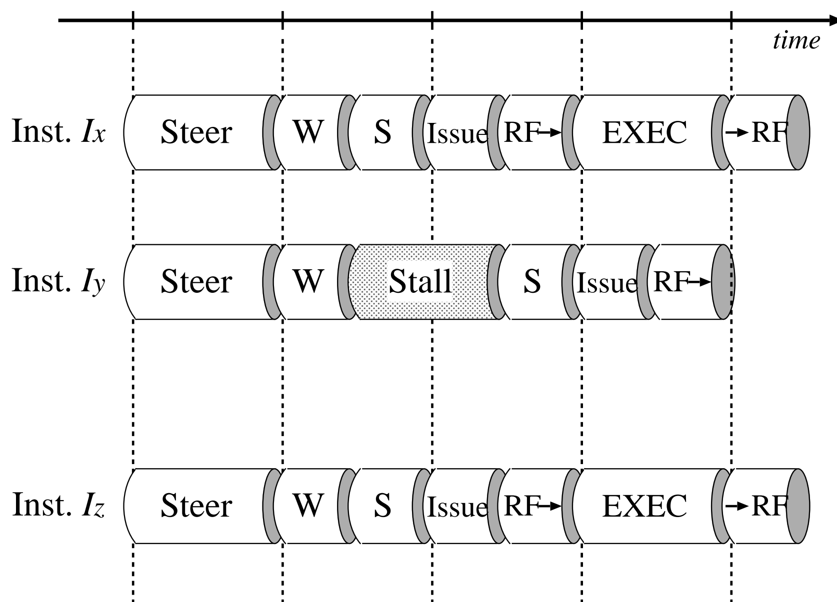

図 8: Select 遅延

じクラスタで，I_z は別クラスタで実行された様子を示す．クラスタ毎の発行幅が 1 命令の場合，命令 I_y は演算資源が利用可能になる 1 サイクル後までストールする．これに対して，命令 I_z は命令 I_x と同時に Select される．

よって，Select 遅延を軽減するには，各クラスタの負荷を均等にし，命令の Wakeup 後すぐさま Select されるようにするのが望ましい．

3.2.4 集中と分散のバランス

ここまでで述べたように，クラスタ型スーパースカラ・プロセッサでは，Wakeup 遅延と Select 遅延をなるべく少なく抑え，命令を速やかに発行することが重要となる．Wakeup 遅延は，フロー依存関係にある命令を同じクラスタへ集中させることで解消でき，Select 遅延は，命令を分散させ各クラスタの負荷を均一にすることにより軽減できる．よって，命令の集中と分散のバランスを取り，Wakeup 遅延と Select 遅延の発生が少ない命令ステアリングがクラスタ型スーパースカラ・プロセッサでは重要となる．

3.3 既存の命令ステアリング

前節で，クラスタ型スーパースカラ・プロセッサでは，Wakeup 遅延と Select 遅延の 2 つの遅延の発生が少ない高精度な命令ステアリングが重要であることを述べた．本節では既存の命令ステアリング方式について述べる．

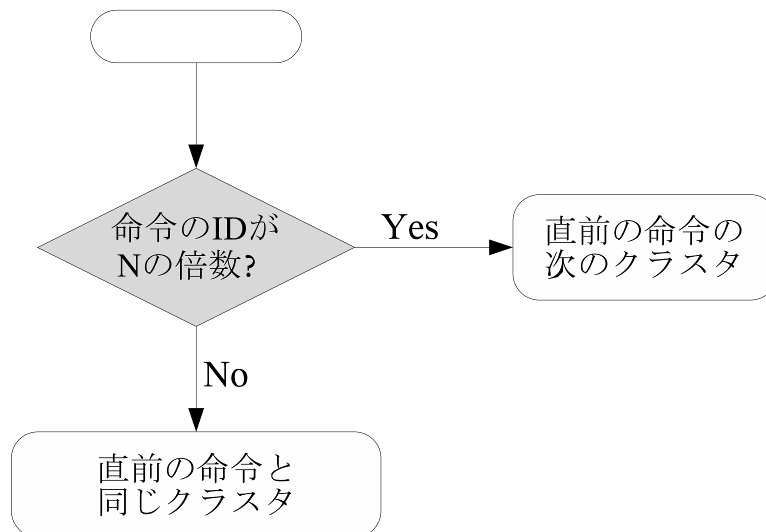

図 9: Round-Robin 型ステアリング方式のアルゴリズム

まず，3.3.1 項で Select 遅延回避を重視した Round-Robin 型の命令ステアリングを，3.3.2 項で Wakeup 遅延回避を重視した命令ステアリングについて説明した後，3.3.3 項で服部らが提案する『発行時間差に基づいた命令ステアリング』について説明する．

3.3.1 Round-Robin 型ステアリング

Select 遅延回避に特化した単純なステアリング方式として Round-Robin 型のステアリングが挙げられる．この方式では，連続した N 個の命令を同じクラスタにステアリングする．X 番目にフェッチされた命令は，クラスタ数を C とすると [X / N mod C] 番のクラスタへステアリングされる（図 9）．

命令が各クラスタに均等にステアリングされるため Select 遅延が発生しにくい．また，連続する命令が依存関係にある場合，これらの命令は同じクラスタにステアリングされやすいので，Wakeup 遅延回避もある程度考慮できる．しかし，後述する Dependence Based ステアリングなどの方式に比べると Wakeup 遅延が多く発生してしまう．そのため，クラスタ間通信遅延が大きいような構成のプロセッサには不向きである．

3.3.2 Dependence Based ステアリング

Round-Robin 型のステアリングが Select 遅延回避に特化していたのに対し，Wakeup 遅延回避に特化したステアリング方式として，Dependence Based ステアリングが挙げられる．この方式では，対象命令に未解決のオペランドがある場合

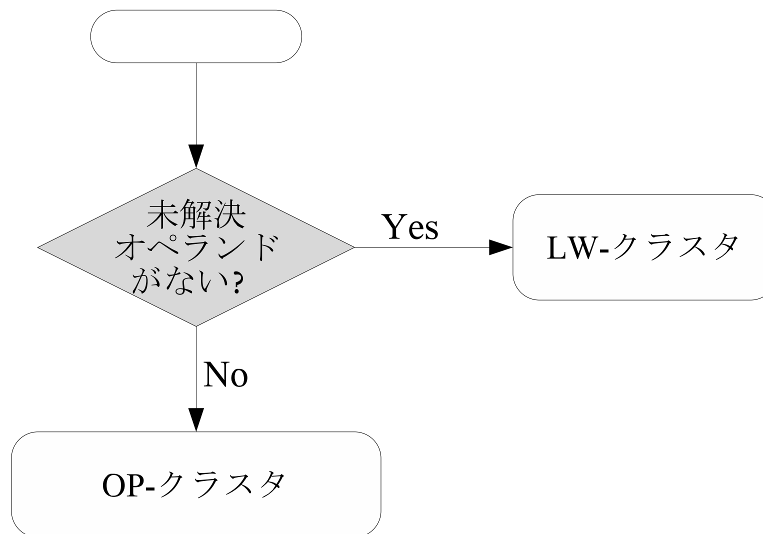

図 10: Dependence Based ステアリング方式のアルゴリズム

には未解決オペランドと同じクラスタ（以下 OP-クラスタと略記: OP は Operand Producer の略）へその対象命令をステアリングする．未解決オペランドが存在しない場合は，負荷が最小のクラスタ（以下 LW-クラスタと略記: LW は Lowest Workload の略）へステアリングを行う（図 10）.

OP-クラスタは，命令のリネーム時に割り当てられる命令タグとクラスタ番号の対応付けを行うテーブル（STT: Steering Target Table）を参照することにより特定される．まず，レジスタ・リネーミングを終えた命令は，ソース・オペランドに対応した命令タグをキーとし，STT を参照し OP-クラスタのクラスタ番号を得る．ステアリング先のクラスタ番号が決定すれば，デスティネーション・オペランドに対応した命令タグとクラスタ番号を登録することにより STT を更新する．

ただし，レジスタ・リネーミングは複雑な論理を必要とするため，STT への参照はリネーム・ステージの後に 1 ステージのステアリング・ステージを追加して行われる．フロントエンドでのステージの増加は，分岐予測のミス・ペナルティの増加に繋がり，IPC の低下を招く．

この方式は，Wakeup 遅延回避に特化している一方，特定のクラスタに命令が集中しやすく Select 遅延の影響を受けやすい．このため，クラスタ毎のスループットが小さい場合には不向きである．

3.3.3　発行時間差に基づいた命令ステアリング

クラスタ型スーパースカラ・プロセッサでは，滞りなく命令を発行するために，

ステアリング時に『最も早く発行可能なクラスタ』を特定し，そのクラスタへ命令をステアリングすることが望ましい．服部らは，データの Producer と Consumer の間の距離（命令数）に着目し，依存命令の発行時間差を近似的に計算することにより，最も早く発行可能なクラスタを推定している [13].

Producer が発行してから Consumer が発行するまでの時間差（IssueDelay）は，Wakeup 遅延と Select 遅延の和：

$$IssueDelay = WakeupDelay + SelectDelay$$

で求められる．

対象命令の未解決オペランドと同じクラスタ内の命令発行スループットを T，OP-クラスタにおける Producer と Consumer の間の命令数を Δ とすると，OP-クラスタにおける Select 遅延は Δ/T である．また，LW-クラスタにおける Select 遅延は小さいと考えられるため，0 に近似する．よって，通信遅延を C とすると以下の近似式が成り立つ：

$$WakeupDelay(OP) = 0$$

$$WakeupDelay(LW) = C$$

$$SelectDelay(OP) = \Delta/T$$

$$SelectDelay(LW) = 0$$

従って，発行時間差は以下の式で求められる．

$$IssueDelay(OP) = \Delta/T$$

$$IssueDelay(LW) = C$$

すると，『最も早く発行可能なクラスタ』は Δ/T と C の大小比較，つまり Δ と $C \times T$ の大小比較で決定できる．

Δ を正確に算出する方法として，ステアリングされた命令に対して，クラスタ毎に通し番号を付ける．命令のステアリング時に，Producer との通し番号の差を発行時間差と近似する．この通し番号の差を Local Distance と呼ぶ．

この発行時間差に基づいたステアリング方式のアルゴリズムを図 11 に示す．まず，ステアリング時に未解決オペランドがない場合には，Dependence Based と同

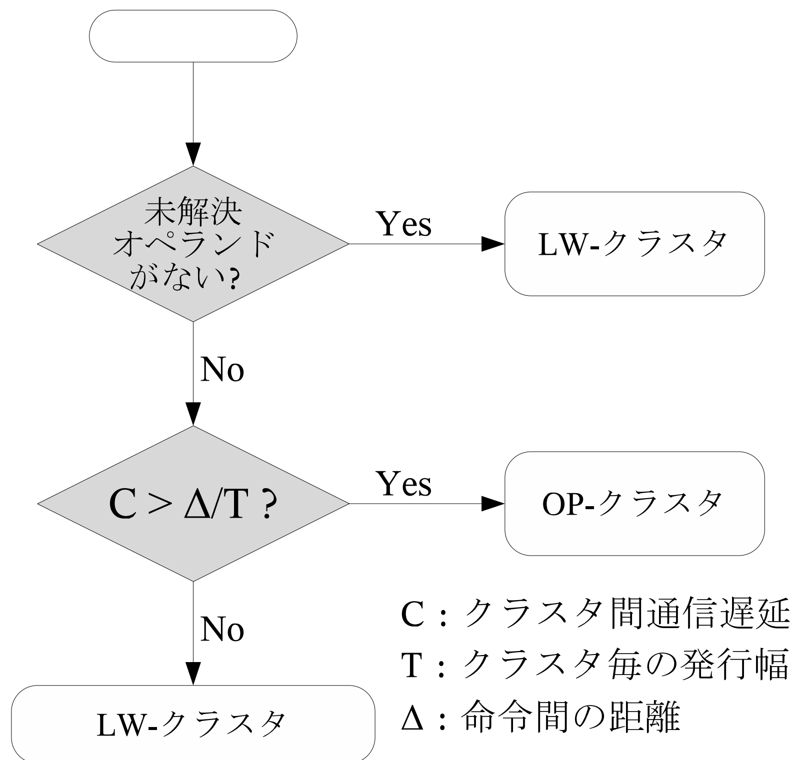

図 11: 発行時間差に基づいたステアリング方式のアルゴリズム

様 LW-クラスタへステアリングされる．未解決オペランドが 1 つ以上ある場合には，Producer との発行時間差を計算し，その値がクラスタ間通信遅延より大きければ LW-クラスタへ，小さければ OP-クラスタへステアリングする．

この方式でも，リネーム処理の後に，STT を参照し OP-クラスタを特定する．そのため，Dependence Based ステアリングと同様，ステアリング先のクラスタを決定するステージが追加される．

3.4　第 3 章のまとめ

本章では，クラスタ型スーパースカラ・プロセッサについて述べた．クラスタ化されたプロセッサでは，クラスタ化されていないプロセッサに比べ，クラスタ間のデータ転送遅延による Wakeup 遅延と特定クラスタへの命令の集中による Select 遅延により IPC が低下する．そのため，命令の集中と分散のバランスを取り，Wakeup 遅延と Select 遅延の発生が少ない命令ステアリングが重要となる．

次章では，クラスタ型スーパースカラ・プロセッサにおける命令ステアリングにスラック予測の予測結果を応用する方法について述べる．

第4章　スラック予測を用いた命令ステアリング

本稿では，クラスタ型スーパースカラ･プロセッサにおける命令ステアリングにスラック予測による予測結果を応用する方法について述べる．

冒頭でも述べたように，スラックとは『ある命令の実行を s サイクル遅らせてもプログラムの実行時間が増大しないような s の最大値』と定義される．よって，例えばクラスタ通信遅延が C サイクルの場合，スラックが C 以上の命令は，依存する先行命令とは別のクラスタで実行しても，命令の実行は遅れない．このように，スラック予測による予測値をクラスタ型スーパースカラ･プロセッサの命令ステアリングに応用することができる．

以下，まず 4.1 節と 4.2 節でスラック予測を命令ステアリングに応用する利点と計算における注意点を述べた後，4.3 節でスラック予測を用いた提案命令ステアリング方式について説明する．

4.1　スラックの利用

図 12 はクラスタ型スーパースカラ･プロセッサにおける命令の実行の様子を表すタイム･チャートである．図中の “I” の左側の添字 $C1$，$C2$ は命令を実行したクラスタ番号を表しており，矢印はクラスタ間通信による遅延（この図では 1 サイクル）を表している．つまり，図 12・左でクラスタ $C1$ で実行される I_1 とクラスタ $C2$ で実行される I_3 との間には通信遅延として 1 サイクル必要となり，両方ともクラスタ $C2$ で実行される I_2 と I_3 の間には通信遅延は生じない．

本稿の冒頭でも述べたように，スラックはサイクルを単位とし多値的に表現される．よって，クラスタ間通信遅延が C サイクルの場合，スラックが C 以上の命令を後続命令と別クラスタで実行しても後続命令の実行開始は遅れない．例えば，図 12・左のようにスラックが 1 の命令 I_1 と後続命令 I_3 を別クラスタで実行した場合，クラスタ間通信遅延により I_1 の実行結果を使用できる時刻が 1 サイクル遅れるが，I_3 の実行開始は遅れておらず，プログラム全体としての実行時間は増大しない．一方，図 12･中ではクラスタ間通信遅延により I_3 の実行が 1 サイクル遅れている．このような場合，クラスタ毎の発行幅が 2 命令以上であれば，図 12･右のように I_1，I_2，I_3 の 3 命令を同じクラスタで実行することにより，1 サイクル早く I_3 を実行できる．

このように，命令ステアリングの際にスラックの予測値を用いることにより，

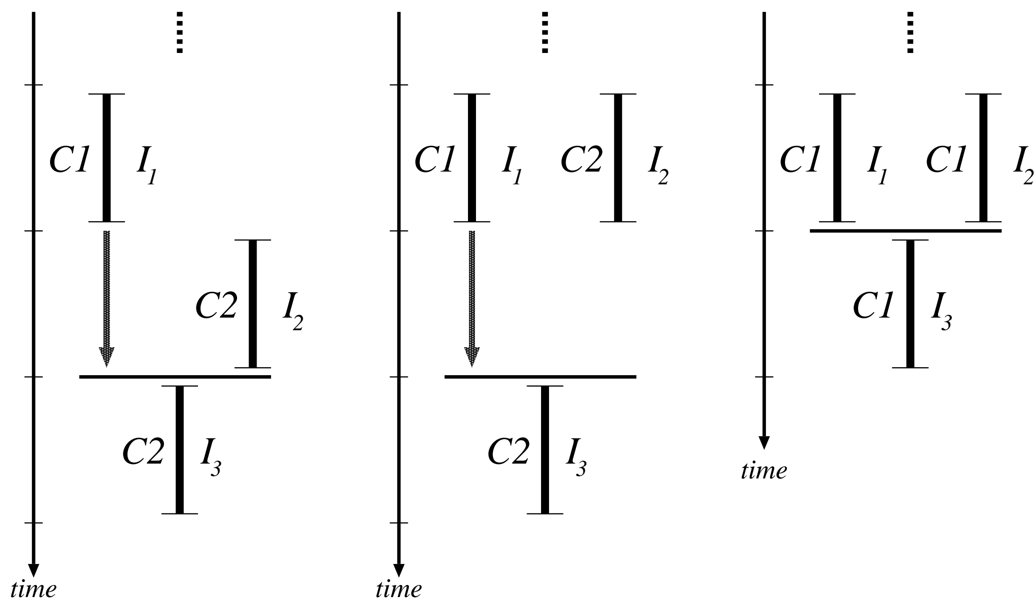

図 12: スラックの利用・計算

Wakeup 遅延発生によるプログラムの実行時間の増大を防ぐことができると期待できる．

また，スラック予測はプロセッサの物理的な制約も考慮している．つまり，特定クラスタへ命令が集中し演算資源を獲得できない Select 遅延が発生した場合，次回実行時にはそれらの命令のスラックは大きく予測される．よって，スラックが大きな命令を負荷の少ないクラスタへステアリングする事により特定クラスタへの負荷の集中も防ぐことができると期待できる．

4.2 スラックの計算

前節でクラスタ型スーパースカラ・プロセッサにおける命令ステアリングにスラック予測を応用することでより高精度な命令ステアリングが期待できることを述べた．しかし，クラスタ型スーパースカラ・プロセッサでスラックの計算をする際には注意が必要である．

データを定義する命令と使用する命令を別クラスタで実行する場合，スラックの計算で用いる定義時刻は定義命令が実行されたクラスタでデータを定義した時刻であり，別クラスタでデータが使用可能になった時刻ではない．つまり，図 12・左のような場合，第 1 章の式 (1) に基づいてスラックの計算をすると，I_1 のスラックは 1，I_2 のスラックは 0 となる．その結果，次回実行時にも I_1 のみ別クラ

スタで実行しても，通信遅延により I_3 の実行開始は遅れないと予測される．

一方，図 12・中のような場合，式 (1) に基づき計算すると，I_1，I_2 の両方のスラックが 1 となる．これは次回実行時に 3 つの命令を別々のクラスタで実行し，1 サイクルのクラスタ間通信遅延が発生しても I_3 の実行は遅れないことを示す．ところが，前述したように 3 つの命令を同一クラスタにステアリングすることにより，図 12・右のように I_3 を 1 サイクル早く実行できる．つまり，実際には 1 サイクルも遅らせることができない命令をスラックが 1 と予測していることになる．

さらに，以降のスラックの予測値も 1 よりも大きくなる．このため，依存関係にある 2 命令が一度別々のクラスタで実行されると，ずっと別々のクラスタで実行されてしまう可能性がある．

図 12・中のような場合，クラスタ間通信遅延だけ無駄に I_3 の実行が遅れている．したがって，各命令の実効的なスラックは，式 (1) により得られたスラックの値からクラスタ間通信遅延分だけ減ずればよい．つまり，データを定義した時刻を t_d，使用した時刻を t_u，クラスタ間通信遅延を C サイクルとしたとき，スラックは:

$$s = t_u - t_d - 1 - C \tag{2}$$

によって得られる．

そこで，クラスタ間通信遅延分だけ無駄に遅れている状況を発見するために，スラック予測器の定義表にデータの定義命令を実行したクラスタ番号を記憶しておく．そして，クリティカルになっている命令が全てデータの使用命令とは別のクラスタで実行されていれば，クラスタ間通信による遅延の分だけスラックを小さくする [17]．たとえば，図 12・中のように，クリティカルな命令が全て別のクラスタで実行された場合には，式 (2) によりスラックを求める．その結果，図の I_1，I_2 の両方のスラックは 0 となる．なお，図 12・左のような場合には，I_3 と同じクラスタで実行された I_2 がクリティカルになっているため，通常通り式 (1) によりスラックが計算されることに注意されたい．このようにして計算したスラックの値をスラック表に記憶し，次回実行時には図 12・右のように全ての命令を同じクラスタで実行した方がよいと予測をする．

4.3　スラック予測を用いた提案命令ステアリング方式

前節までで，クラスタ型スーパースカラ・プロセッサにおける命令ステアリングに対して，スラック予測による予測結果が応用可能であることを述べた．

本節では，提案するスラック予測を用いた命令ステアリング方式について述べる．まず，4.3.1 項ではスラック予測器を拡張することにより，フェッチ時に同一実行パス上にある次の命令を特定し，実行パス単位で命令をステアリングする方法について説明する．また，4.3.2 項では服部らが提案する「命令間の距離に基づき発行時間差を推定しステアリングするクラスタを決定する手法」にスラック予測を利用し，より実効的な距離に基づき発行時間差を推定する方法について説明する．

4.3.1　Path Based ステアリング

3.3.2 項，3.3.3 項でも述べたように，Dependence Based や発行時間差に基づく命令ステアリングでは，命令タグとクラスタ番号の対応付けを行う STT（Steering Target Table）を参照することで，未解決オペランドがあるクラスタ（OP-クラスタ）を特定した．命令タグはレジスタ・リネーミングによる依存関係の解析後に割り当てられる．しかし，レジスタ・リネーミングは複雑な論理を必要とするため，STT への参照はリネーム・ステージの後に 1 ステージのステアリング・ ステージを追加して行われる．フロントエンドでのステージの増加は，分岐予測のミス・ペナルティの増加に繋がり，IPC の低下を招く．

そこで，SST への参照のキーとして命令タグではなく，**命令 ID** を使用することを考える．命令 ID は命令のフェッチ時に与えられるその命令固有の番号である．これにより，命令のフェッチ時にこのテーブルにアクセスが開始でき，フロントエンドにおけるステージの増加は必要ない．

この命令 ID を特定するためにスラック予測器を利用する．第 2 章で説明したように，スラック予測は『あるデータを定義命令 I_d が定義した時刻と I_d が定義したデータを使用する命令 I_u の使用時刻の差が s サイクルであったとき，それらの命令が次回再び実行されるときの定義時刻と使用時刻の差も，また s サイクルである.』という予測の下で行われる．

命令 ID の割り当ては，命令のフェッチ時に in-order に行われる．そのため，スラックと同様に，データの定義命令 I_d とデータの使用命令 I_u の命令 ID の差が Δ であったとき，それらの命令が次回再び実行されるときの命令 ID の差も，また

Δである可能性が高い．よって，スラック予測器を拡張し，I_d と I_u の前回実行時の命令 ID の差をスラック表に登録しておけば，次の I_d のフェッチ時にスラック表を読み出すことにより，同一実行パス上の次の命令である I_u の命令 ID が特定できる．そして，この I_u の命令 ID と I_d のステアリング先のクラスタ番号を STT に登録しておき，I_u のフェッチ時に命令 ID をキーとし STT を参照することにより，I_d をステアリングした OP-クラスタのクラスタ番号を得ることができる．このように，スラック予測器の拡張により，命令のリネームを待たずに STT への参照を開始できるので，フロントエンドにステージを増加させずに命令の依存関係を考慮した命令ステアリングが実現できる．

スラック予測器に命令 ID の差を求める機能を追加するために，定義表にデータの定義命令の ID を記録するフィールドを，またスラック表に依存関係のある 2 命令間の命令 ID の差を記録するフィールドを付加する．

スラック予測器の登録，予測に追加する操作を以下に示す：

1. **登録**　データを使用する命令の実行時にスラックの値を計算するのと同様に，命令 ID の差も計算する．なお，W_D，R_D，W_S は 2.2.2 項の図 2 中のそれとそれぞれ対応している．なお，命令 ID の差を計算するために新たに追加した操作は太字で表している．
   (a) **定義**　I_d がデータを定義するとき，以下の操作が行われる：
   W_D　I_d がデータを定義するとき，現時刻と I_d 自身（のアドレス）に加え，**命令 ID を定義表に書き込む**．
   (b) **使用**　I_u がそのデータを使用するとき，以下の 2 つの操作が連続して行われる：
   R_D　定義表を読み出して，定義時刻 t_d，定義命令 I_d（のアドレス），**定義命令 I_d の ID を得る**．
   W_S　定義表を読み出して得られた定義時刻と現時刻の差でスラック s_d を求めたのと同様，定義表から読み出した **I_d の命令 ID と I_u の命令 ID の差 Δ_d を求める**．スラック表の定義命令 I_d のエントリに，求めた s_d と **Δ_d を書き込む**．
2. **予測**　命令 I_d が再びフェッチされると，次の操作が行われる．なお，R_S，W_{STT}，R_{STT} は図 13 のそれとそれぞれ対応している：
   R_S　I_d のアドレスをインデクスとしてスラック表を読み出し，スラックの予測値 s_d と Consumer との命令 ID の差 Δ_d を得る．

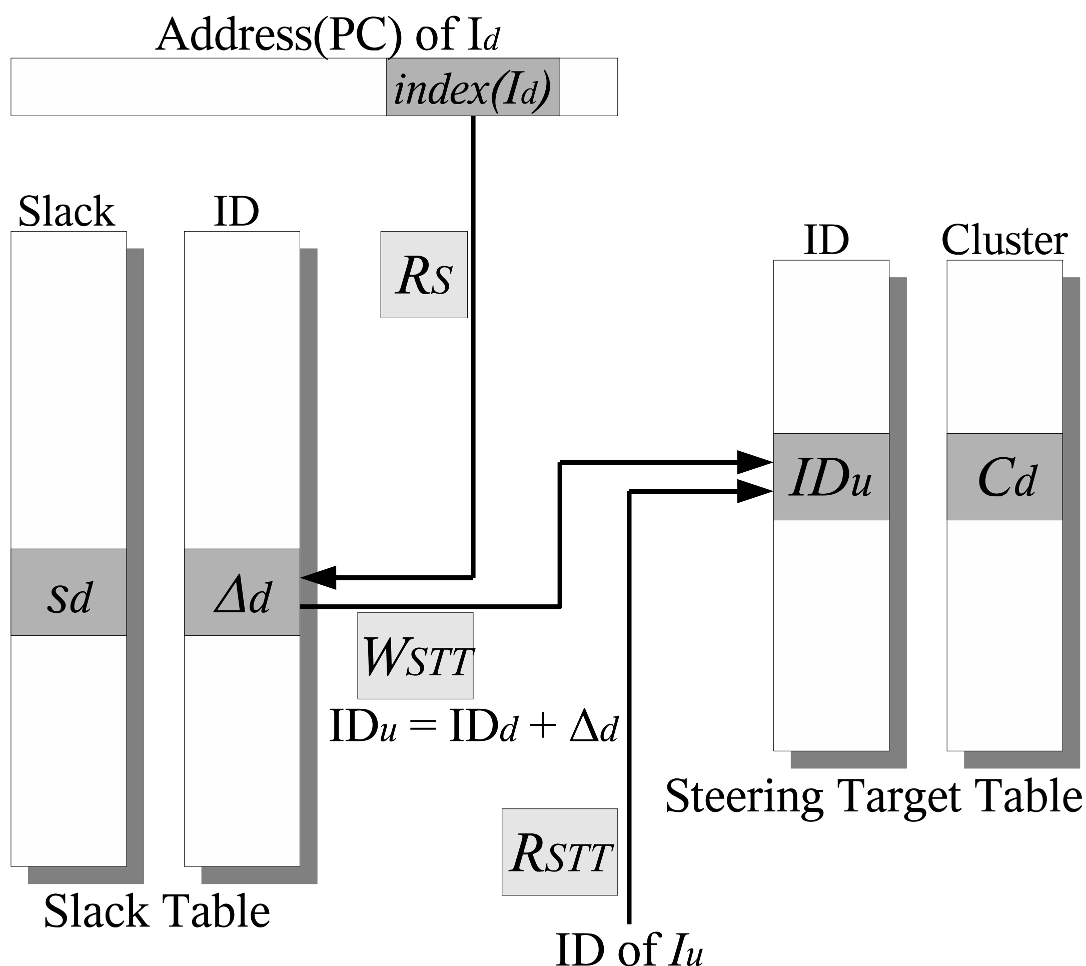

図 13: STT に対する操作

W_{STT}　R_S で得た s_d がクラスタ間通信遅延 C 未満であった場合に限り，以下の操作を行う．s_d が C 以上であった場合には何も行わない．

- 命令 I_d の ID_u と R_S で得た Δ_d から，命令 I_u の ID が，$ID_u = ID_d + \Delta_d$ と求まる．求めた ID_u と I_d のステアリング先のクラスタ番号 C_d を STT に登録する．

R_{STT}　命令 I_u のフェッチ時に，I_u の命令 ID を用いて STT を参照し，登録されている C_d を得る．同時に，R_S，W_{STT} の操作も行い，STT を更新する．

STT へのアクセスがミスした場合には，クラスタの負荷が最小の LW-クラスタにステアリングする．W_{STT} の操作時に s_d が C 以上で STT に対する登録が行われなかった場合には，命令 I_u による STT に対するアクセ

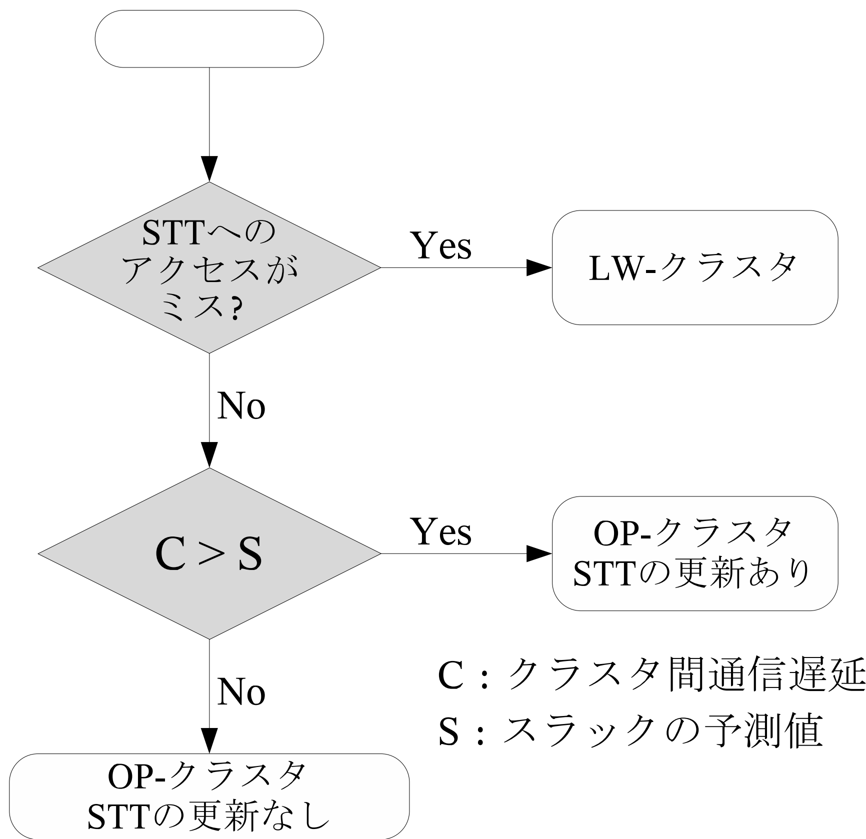

図 14: Path Based ステアリング方式のアルゴリズム

スはミスする．その結果，I_u は LW-クラスタへステアリングされる．しかし，スラックの予測値が C 以上であるので，C サイクルのクラスタ間通信遅延が生じても I_u の実行には影響しないと予測できる．このことは，Select 遅延回避の効果がある．

この Path Based ステアリング方式のアルゴリズムを図 14 に示す．まず，フェッチされた命令は，スラック表を参照すると共に，命令 ID をキーとし STT を参照する．STT への参照がミスであれば，その命令を LW-クラスタへステアリングし，ヒットであれば STT から得られたクラスタ番号へステアリングする．また，STT への参照がヒットし，スラックの予測値がクラスタ間通信遅延 C よりも小さければ STT を更新する．

このように，本節で説明した命令ステアリングでは，リネーム・ステージでの依存関係の解析を待たずに依存関係を考慮した命令ステアリングが可能となる．よって，この方式では，Dependence Baseステアリングのようにディスパッチの前にステアリング・ステージを必要としない．このことは，分岐予測ミス・ペナルティ

の軽減にも効果がある．また，スラックの予測値がクラスタ間通信遅延以上である命令をLW-クラスタへステアリングするため，Wakeup 遅延と Select 遅延の両方を考慮したバランスのよい命令ステアリングができると期待できる．

4.3.2　スラックの予測値を発行時間差とする方法

3.3.3 項で，命令のステアリング時に，Producer と Consumer 間の距離（命令数）から依存する 2 命令の発行時間差と近似することにより，「最も早く発行可能なクラスタ」を推定する服部らの提案方式について説明した．

しかし，本稿のクラスタ型スーパースカラ･プロセッサは Out-of-Order 実行することを想定している．つまり，命令ウィンドウ内の命令は使用するオペランドが全て利用可能となっていれば実行が可能となり，命令は各クラスタへステアリングされた順に実行されるとは限らない．そのため，Producer と Consumer の間にある命令が，Producer より早く発行される可能性がある．逆に，Producer がさらに先行する命令よりも早く発行される可能性もある．このような場合には，Producer と Consumer の 2 命令間の距離（Δ）をクラスタ毎発行幅（T）で割った値（Δ/T）が発行時間差とはならない．そのため，2 命令間の距離に基づき発行時間差を推定するステアリング方式では，最も早く発行可能なクラスタではないクラスタへステアリングされる可能性がある．

そこで，本項では，より正確に発行時間差を推定するためにステアリング時にスラック予測器による予測値を利用することを考える．

Producer のスラックの予測値は，前回実行時に Producer がデータを定義した時刻と Consumer がデータを利用した時刻の差によって求められる．発行された命令は，次のサイクルで実行されることが決定的になっている．このことから，スラックの予測値は，前回実行時の『発行時間差』と言い換えることができる．よって，3.3.3 項で説明した発行時間差を推定する方法にスラックの予測値を利用することで，より実効的な発行時間差を推定できると期待できる．

3.3.3 項で説明した手法では，Producer と Consumer との距離に基づいた発行時間差の近似値（Δ/T）がクラスタ間通信遅延 C 未満であれば，未解決オペランドがある OP-クラスタへステアリングした．一方，本項で提案するスラックの利用した命令ステアリング方式では，Δ/T がクラスタ間通信遅延 C 未満であっても，Producer のスラックの予測値が C 以上であれば命令数が最少の LW-クラスタへステアリングする．これは，『前回の発行時間差が C 以上であるので，今回も同様に発行時間差が C 以上になるであろう．』と予測できるためである．

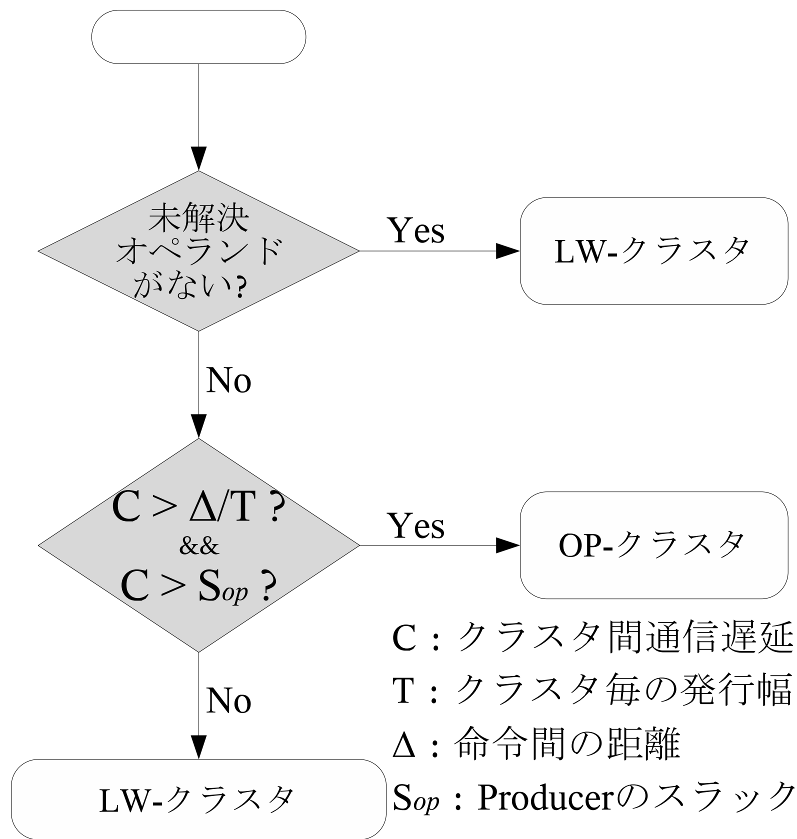

図 15: 提案ステアリング方式のアルゴリズム

また逆に，『Δ/T が C 以上であっても Producer のスラックの予測値が C 未満であればOP-クラスタへステアリングする』ことも考えられるが，ここでは行わない．命令数が最少の LW-クラスタへステアリングされた命令は，命令ウィンドウ内の命令数が少ないため Wakeup 後すぐに Select される可能性が高い．そのため，次回実行時には Producer のスラックの予測値は小さくなると予想される．このような命令を全て OP-クラスタへステアリングすると，負荷が集中してしまう恐れがあるためである．

提案する命令ステアリング方式のアルゴリズムを図 15 に示す．服部らが提案する方式では，Producer との距離に基づく発行時間差（Δ/T）がクラスタ間通信遅延 C より小さければOP-クラスタへステアリングしていた．これに対し，提案方式では，Δ/T が C より小さく，かつ，Producer のスラックの予測値 S_{op} が C より小さい命令を OP-クラスタへステアリングする．なお，この方式では，Dependence Based ステアリングなどと同様，リネーム処理の後に STT を参照し OP-クラスタを特定する．よって，リネーム・ステージの後に，ステアリング・ステージを必要と

する．

この提案方式では，命令のステアリング時に Producer のスラックの予測値を知る必要がある．そこで，STT を拡張しスラックを記憶するフィールドを付加する．これにより，レジスタ・リネーミングを終えた命令は，STT へのアクセスで OP-クラスタに加えて Producer のスラックの予測値を得ることが可能になる．その後，図 15 のアルゴリズムに従い，ステアリングされるクラスタが決まれば，その命令の命令タグ，クラスタ番号，スラックの予測値を STT に記録しテーブルの更新をする．

提案方式では，Producer との距離は小さいが Producer のスラックの予測値が大きいような命令を LW-クラスタへステアリングする．これにより，3.3.3 項で説明した方式と比べ，Wakeup 遅延による影響をあまり受けずに負荷を分散することができると期待できる．なお，提案方式ではスラックの予測値のみを使用するため，スラック予測器への拡張は必要ない．

4.3.3　スラック表にカウンタを付加する方法

前項の方式では，スラックの予測値がクラスタ間通信遅延より大きければ LW-クラスタへステアリングした．しかし，LW-クラスタは命令ウィンドウ内の命令数が少ないため Select 遅延は生じにくく，命令はクラスタ間遅延による Wakeup 遅延後にすぐ実行される可能性が高い．そのため，次回のスラックの予測値は小さくなると予想される．つまり，スラックの予測値が大きい命令は，一旦は負荷分散のため LW-クラスタへステアリングされるが，次の実行時にはスラックの予測値は小さくなり，再び OP-クラスタへステアリングされるようになる．このように，実際には毎回 LW-クラスタで実行しても Wakeup 遅延の影響を受けない命令であっても，OP-クラスタへステアリングされてしまうことがあり，効率よい負荷分散が期待できない．

そこで，スラック予測器のスラック表にカウンタを付加し，命令を OP-クラスタにステアリングしたが，その命令の実行の結果，LW-クラスタにステアリングしていても Wakeup 遅延の影響を受けないと判定できる回数をカウントする．つまり，スラック表へのスラックの登録時，データの Producer と Consumer の使用クラスタが同じで，計算したスラックの値がクラスタ間通信遅延 C 以上であれば，スラック表のカウンタの値を 1 増やす．そして，このカウンタの値が任意の定数 X より大きくなれば，その命令の Consumer は次回から LW-クラスタにステアリングされる．Producer のカウンタの値を知るために，STT にもカウンタの値を記憶す

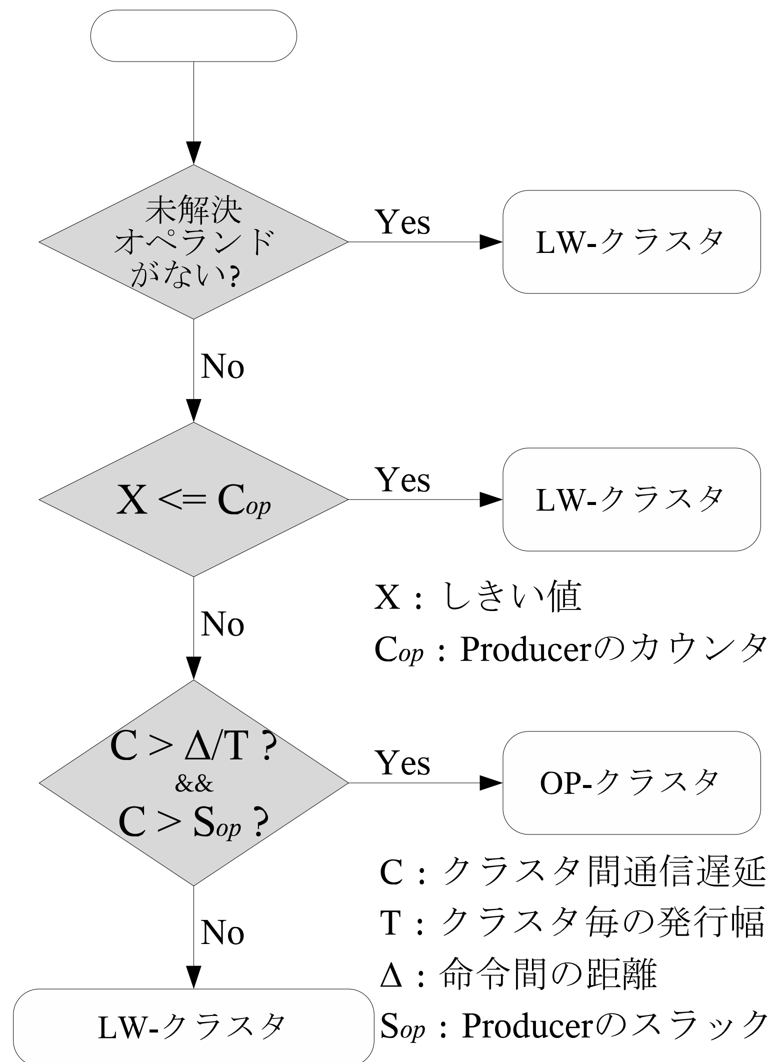

図 16: カウンタを付加した提案ステアリング方式のアルゴリズム

るフィールドを追加する．命令のステアリング時には，そのフィールドを読み出し，Producer のカウンタの値を得る．

この提案ステアリング方式のアルゴリズムを図 16 に示す．レジスタ・リネーミングを終えた命令は，STT へのアクセスにより，OP-クラスタのクラスタ番号，Producer のスラックの予測値，Producer のカウンタを得る．カウンタが任意の値 X より大きければ LW-クラスタにステアリングする．逆に，カウンタが X より小さければ，図 15 と同様のアルゴリズムに従いステアリングする．これにより，発行時間差の大きいと予測される命令を効率よく LW-クラスタで実行できると期待できる．ステアリングされるクラスタが決まれば，その命令の命令タグ，クラスタ番号，スラックの予測値，カウンタを STT に記録しテーブルの更新をする．

4.4 第4章のまとめ

本章では，クラスタ型スーパースカラ・プロセッサにおける命令ステアリングにスラック予測の予測結果を応用する方法について述べた．

4.3.1項では，スラック予測器を拡張することにより，フェッチ時に同一実行パス上にある次の命令を特定し，実行パス単位で命令をステアリングする方法について述べた．これにより，他の依存関係を考慮したステアリング方式のように，リネーム・ステージでの依存関係の解析を待たずにOP-クラスタを特定できるため，フロントエンドを簡単にできる．

4.3.2項では，服部らが提案する「命令間の距離により発行時間差を推定しステアリングするクラスタを決定する手法」にスラック予測を利用し，より実効的な距離に基づき発行時間差を推定する方法について述べた．ProducerとConsumerとの距離は小さいが，スラックが大きい場合に，命令をLW-クラスタへステアリングすることにより，3.3.3項で説明した服部らの方式と比べ，Wakeup遅延による影響をあまり受けずに負荷を分散することができると期待できる．

4.3.3項では，さらに効率良く負荷分散するために，スラック表にカウンタを付加した．スラックの値がクラスタ間通信遅延より大きい回数をカウントし，カウンタがしきい値より大きくなれば毎回LW-クラスタへステアリングする．

次章では，本章で提案したスラック予測を用いた命令ステアリングをシミュレーションにより評価を行う．

第5章　評価

第4章でスラック予測を応用した提案命令ステアリング方式をシミュレーションにより評価を行った．以下ではその結果を述べる．

5.1　評価環境

SimpleScalar ツールセット (ver. 3.0) の sim-outorder シミュレータに対して，スラック予測器とクラスタ型スーパースカラ･プロセッサを実装し，3.3 節で説明した既存の命令ステアリング方式と提案命令ステアリング方式を用いた場合の IPC を測定した．

測定には，表 1 に示す SPEC CINT2000 の 8 本のベンチマーク･プログラムを使用した．なお，入力には train 入力セットを用い，測定時間を短縮するため，最初の 1 G 命令をスキップし続く 100 M 命令を実行した．

プロセッサのモデル

プロセッサの主なパラメタを表 2 に示す．評価に用いたプロセッサのモデルは 2-CLUSTER，4-CLUSTER，8-CLUSTER の 3 つで，それぞれ実効幅が 8 命令のプロセッサを 2，4，8 つにクラスタ化したものである．つまり，各クラスタでの発行幅はそれぞれ 4 命令，2 命令，1 命令となる．クラスタ間の通信遅延は 1〜4 サイクルの 4 通りの場合を測定した．各クラスタの命令ウィンドウのサイズは 32 エントリとした．各クラスタは全種類の命令を実行できる，いわゆる万能実行ユニットを発行幅と同じ数だけ所持している．

表 1: SPEC CINT2000 ベンチマーク・プログラム

プログラム	入力セット
256.bzip2	input.compressed
176.gcc	cp-decl.i
164.gzip	input.conbined
181.mcf	inp.in
197.parser	train.in
253.perlbmk	scrabbl.in
255.vortex	lendian.raw
175.vpr	net.in arch.in

また，評価の対象としたモデルは，分離 (separate) ロード / ストア方式を採用している．すなわちロード / ストア命令は，ディスパッチ時に，アドレス計算を行う命令と，実際にメモリ（キャッシュ）アクセスを行う命令に分離され，個別にスケジューリングされる．評価では，分離されたそれぞれに対してスラックを予測し，スケジューリングを行っている．したがって，最初からアドレス計算命令とメモリ・アクセス命令の 2 つの命令があったものと考えてよい．

テーブルのパラメタ

表 3 に，テーブルのパラメタを示す．スラック表，および，メモリ定義表の容量は，それぞれ，1次命令，および，1次データ・キャッシュと同じ範囲をカバーできるようにした; すなわち，それぞれ 8K エントリである．ただし連想度は，1次命令，および，1次データ・キャッシュがそれぞれ 2 であるのに対して 4 とした．このように，やや大きな容量/ 連想度としたのは，ミスによる影響を評価結果から除外するためである．メモリのレイテンシ（18 サイクル）は，メモリ・インタフェイスを集積する AMD Athlon プロセッサのものを参考にした．

表 2: プロセッサの各パラメタ

パラメタ	サイズ		
モデル	2-CLUSTER	4-CLUSTER	8-CLUSTER
クラスタ数	2	4	8
クラスタ毎発行幅	4 命令	2 命令	1 命令
総実効幅	8 命令		
万能実行ユニット	計 8 個		
クラスタ間通信遅延	1〜4 サイクル		
命令ウィンドウ	各 32 エントリ		
分岐予測器			
予測方式	gshare		
PHT	4K エントリ		
グローバル分岐履歴長	12		
ミス・ペナルティ	6 サイクル		
リターン・アドレス・スタック	8 エントリ		

ステアリング方式

測定に用いたステアリング方式は次の6つ：

RR 3.3.1項で述べたSelect遅延回避を重視したRound-Robin型のステアリング方式で，連続したN個の命令を同じクラスタにステアリングする．各クラスタに均等にステアリングできるためSelect遅延が発生しにくい．一方で，依存関係にある命令を別々のクラスタにステアリングしてしまう可能性が高いため，Wakeup遅延が多く発生する．

DB 3.3.2項で述べたWakeup遅延回避を重視したDependence Basedのステアリング方式で，未解決オペランドがあるOP-クラスタへステアリングする．依存関係にある命令を同一クラスタへ集中させるため，クラスタ間通信遅延によるWakeup遅延の影響を受けにくい一方，特定のクラスタに命令が集中してしまうためSelect遅延が多く発生する．

LD 3.3.3項で説明した服部らが提案した方式．データのProducerとConsumerの距離（LD：Local Distance）によって発行時間差を近似し，もっとも早く発行可能なクラスタを推定する．近似した発行時間差がクラスタ間通信遅延未満であればOP-クラスタへ，そうでなければ命令数が最も少ないLW-クラスタへステアリングする．

PB 4.3.1項で説明した提案方式．スラック予測器を拡張することにより，同

表3: 各表，キャッシュ，メモリのパラメタ

	容量	ラインサイズ	連想度	レイテンシ (cycles)
スラック表	8K命令	—	4	1
レジスタ定義表	64命令	—	64	1
メモリ定義表	8K命令	—	4	1
1次 命令	8K命令*	8命令*	2	1
1次 データ	8Kワード	8ワード	2	1
2次	1MB	64B	2	6
メモリ	—	—	—	18†

*: SimpleScalarツールセットでは8B/命令．

†: 最初のワード．後続ワードには2サイクル / ワードが必要．

一実行パス上にある次の命令の ID を特定する．命令のフェッチ時にステアリングするクラスタを特定する STT へのアクセスを開始する．レジスタ・リネーミングの処理を待たずに OP-クラスタを特定でき，フロントエンドを簡単にできる．

ED 4.3.2 項で説明した提案方式．スラック予測器による予測結果を利用することで，Producer と Consumer のより実効的な距離（ED: Effective Distance）を求め，発行時間差を推測する．スラックの予測値が大きな命令を命令数の少ない LW-クラスタへステアリングするため，LD に比べ効率よく負荷分散が期待できる．

EDc ED と同じく 4.3.3 項で説明した提案方式．効率よく負荷分散するためにスラック表にカウンタを付加し，カウンタが任意の定数より大きくなれば LW-クラスタへステアリングする．カウンタが定数より小さければ ED と同じアルゴリズムでステアリングする．なお，評価でしきい値となる定数 X は 1〜4 の場合について測定し，最も IPC がよいものを示した．

これらの 6 つの方式を用いたクラスタ型スーパースカラ・プロセッサと IPC と実効幅が 8 命令のクラスタ化されていないスーパースカラ・プロセッサについてシミュレーションを行い，8 本のベンチマーク・プログラムの IPC の平均を比較することにより評価を行う．なお，評価は IPC のみにより行い，プロセッサの動作周波数などは考慮しないこととする．

また，RR，PB 以外のステアリング方式についてはステアリング先のクラスタを決定するためのステージがフロントエンドに追加される．そのため，RR，PB 以外のステアリング方式では分岐予測のミス・ペナルティを RR，PB の方式よりも 1 大きい 7 サイクルとした．ステアリングに用いる負荷情報は命令ウィンドウ内の命令数とした．

5.2 評価結果

図 17，図 18，図 19 に結果を示す．図 17 は 2-CLUSTER，図 18 は 4-CLUSTER，図 19 は 8-CLUSTER のモデルにおける結果である．

3 つの図には 4 組のバーがあり，それぞれ左からクラスタ間遅延を 1〜4 サイクルとしたものに対応している．また，各組の 6 本のバーは，これまで説明した 6 つのステアリング方式に対応しており，クラスタ化していないプロセッサに対する IPC の比を表す．

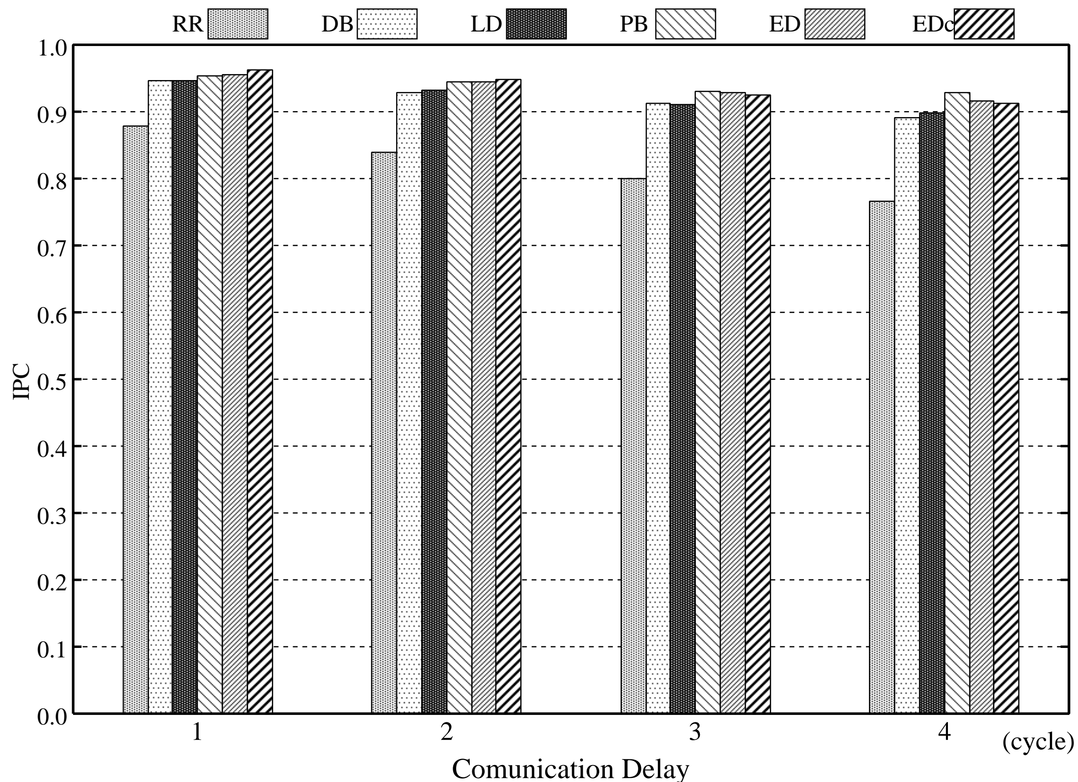

図 17: 2-CLUSTER モデルにおける IPC の比

2-CLUSTER モデルにおける IPC の比

まず，2-CLUSTER モデルにおける IPC の比を表す図 17 を見ると，クラスタ間通信遅延が 4 サイクルの時に，クラスタ化されていないプロセッサに対する IPC の比が，既存方式の LD で約 89.1 %，提案方式の PB で約 92.8 % と約 3.7 % の性能向上が見られた．しかし，通信遅延が 1 〜3 サイクルの場合では，依存関係を考慮した命令ステアリング方式である DB，LD，CT，ED，EDc の IPC はほとんど変わらなかった．特に，Select 遅延回避のための負荷分散をほとんど考慮していない DB も，その他の方式と比べ IPC がほとんど劣っていない．このことから，実効幅が 8 命令のプロセッサを 2 つのクラスタに分割した 2-CLUSTER モデルにおいては，負荷分散はそれ程重要ではなく，Wakeup 遅延回避のために依存関係のある命令を同じクラスタに集中させることが重要であるとわかる．このことは，Select 遅延回避を重視し Wakeup 遅延回避をほとんど考慮していない RR が他の方式に比べ大きく IPC が低下していることからもわかる．また，RR とそれ以外の方式との IPC の差はクラスタ間通信遅延が大きくなる程顕著になっている．

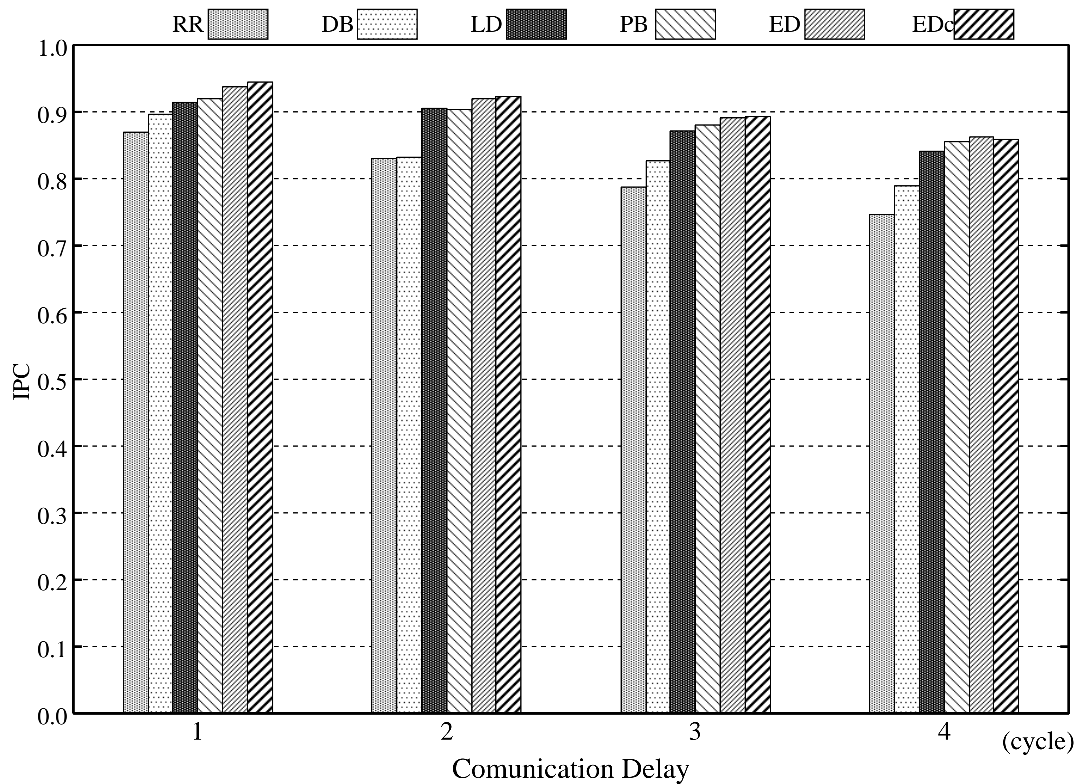

図 18: 4-CLUSTER モデルにおける IPC の比

4-CLUSTER モデルにおける IPC の比

次に，4-CLUSTER モデルにおける IPC の比を表す図 18 を見ると，既存ステアリング方式と提案方式の IPC の差は，クラスタ間通信遅延が 1 サイクルの時の LD（91.4 %）と EDc（94.5 %）が最大で約 3.1 % となっている．しかし，2-CLUSTER モデルと同様，LD，PB，ED，EDc の IPC の差は小さい．一方で，2-CLUSTER モデルでは，この 4 つのステアリング方式と同程度の IPC であった DB は IPC が大きく低下しており，通信遅延が 2 サイクルの場合には，RR と同程度の IPC となっている．また，RR とその他のステアリング方式の IPC の差は 2-CLUSTER モデルの場合よりも小さくなっている．よって，実効幅が 8 命令のプロセッサを 4 つのクラスタに分割した 4-CLUSTER モデルにおいては，依存関係のある命令を同一クラスタへ集中させるだけでなく，命令を分散させ各クラスタの負荷を均一にすることも重要となっていることが分かる．

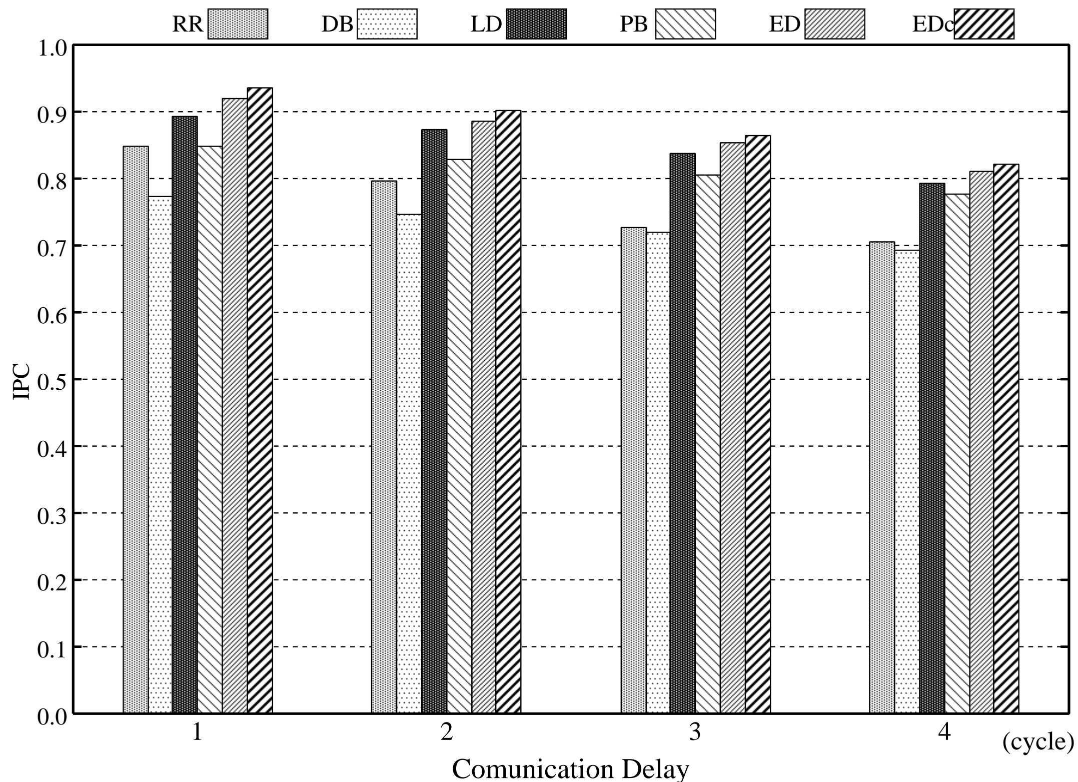

図 19: 8-CLUSTER モデルにおける IPC の比

8-CLUSTER モデルにおける IPC の比

次に，8-CLUSTER モデルにおける IPC の比を表す図 19 を見ると，クラスタ間通信遅延が 1〜4 サイクルのいずれの場合においても，提案方式 EDc がもっとも IPC が高い．既存ステアリング方式でもっとも IPC が高い LD と比較すると，通信遅延が 1 サイクルの場合に約 5 %，通信遅延が 2 サイクルの場合に約 4 %，通信遅延が 3 サイクル，4 サイクルの場合に約 3 % の IPC が向上している．また，ED も LD と比べ 2〜3 % 程度 IPC が向上している．

一方，前述した 2-CLUSTER モデル，4-CLUSTER モデルでは，他の提案ステアリング方式と同程度の IPC であった PB が，ED，EDc と比べ 4〜9 % 程度低い IPC となっている．8-CLUSTER モデルでは，各クラスタの発行幅が 1 命令しかないため，前述のモデルよりも命令の分散が重要となる．PB はこの負荷分散が十分ではなかったと予想される．

また，負荷分散をほとんど考慮しない DB は，依存命令の集中をほとんど考慮しない RR よりも IPC が低くなっている．

5.3 考察

2-CLUSTER モデル，4-CLUSTER モデルでは，既存ステアリング方式と提案ステアリング方式との差はほとんど見られなかった．一方，8-CLUSTER モデルでは最大で約 5 % の IPC 向上が見られた．

クラスタ数が少ないモデルでは，クラスタ毎の発行幅が大きいため，依存関係にある命令を集中させることが重要となる．逆に，クラスタ数が多いモデルにおいては，クラスタ毎の発行幅が小さいため，負荷分散がより重要になる．

このことから，スラック予測を用いた命令ステアリング方式では，既存方式と同程度に依存関係を考慮し，既存方式以上の負荷分散ができていると言える．スラック予測を用いた命令ステアリング方式では，スラックの大きな命令を負荷の少ないクラスタへステアリングさせている．このため，既存の方式に比べ，効率よく負荷分散ができていると推測できる．

当然のことながら，クラスタ数が多い程オペランド・バイパスが短くできるのでクラスタ化の効果は大きくなる．クラスタ数が最も多い 8-CLUSTER モデルにおいて IPC が最も向上していることから，クラスタ型スーパースカラ・プロセッサにスラック予測を応用することが有効であると言える．

第6章　おわりに

本稿では，スラック予測をクラスタ型スーパースカラ･プロセッサにおける命令ステアリングに応用する方法について述べた．

シミュレーションによる評価の結果，発行幅が8命令のプロセッサを8つのクラスタに分割しクラスタ間のデータ通信遅延が1サイクルとした場合において，クラスタ化されていないプロセッサと比較した場合，提案方式は約6.5％程度IPCが低下することが分かった．これは既存ステアリング方式より約5％のIPC向上となる．

このように，クラスタ型スーパースカラ･プロセッサにおける命令ステアリングにスラック予測の予測結果を用いることでIPCが向上することが分かったが，その向上の度合いは最大で約5％とあまり大きな効果は得られなかった．

しかし，本稿の冒頭でも述べたように，スラック予測器の応用先はクラスタ型スーパースカラ･プロセッサにおける命令ステアリングに限ったものではなく，様々な技術への応用が期待できる．スラック予測の応用先として期待される例を以下に示す:

1. **他の予測器**　スラック予測器は，分岐予測ヒット / ミスやキャッシュ･ヒット / ミスなどの結果を反映したスラックを出力する．そのため，それらの予測器と一部機能が重複することが考えられる．機能の重複に関しては，以下のようにまとめられる．

   (a) **分岐予測**

   スラック予測器の出力には，分岐予測ヒット / ミスの結果が繰り込まれる．当然のことではあるが，分岐予測器は，分岐の方向を予測するのであって，分岐予測のヒット / ミスを予測するのではない．したがって，スラック予測器と分岐予測器の間には，機能の重複はない．

   両パス実行を行うプロセッサは，分岐予測ヒット率の低い分岐に対してのみ両パス実行を行うため，分岐予測ヒット / ミスを予測するテーブルを持つ．このテーブルが，スラック予測器に近い働きをする．

   (b) **キャッシュ･ヒット / ミス予測**

   スラック予測器の出力はキャッシュ･ヒット / ミスを反映したものとなる．したがって，スラック予測器の出力をもってキャッシュ･ ヒット / ミス予測を行うことが考えられる．

2. **高性能化**　本稿で説明したクラスタ型スーパースカラ・プロセッサにおける命令ステアリングに対する応用の他に，命令をスケジューリングするときにスラックの予測値が小さい命令を優先的に発行することが考えられる．
3. **省電力化**　スラック予測のプロセッサの省電力化へのアプローチとして，(a) 省電力アーキテクチャ，(b) DVS 制御などへの応用が考えられる．
   (a) **省電力アーキテクチャ**
   スラックの値が大きな命令のみを低速 / 低消費電力の演算器で実行することで，性能を大きく低下させることなく演算器で消費される電力を抑えることができる [17].
   (b) **DVS 制御**
   近藤らは，主記憶によるキャッシュ・ミスのサービスを契機として，プロセッサに DVS 制御を行うことを提案している [18]．前述したように，スラック予測はキャッシュ・ヒット / ミスを考慮できる．よって，キャッシュ・ミスを起こすと予測されるロード命令が複数あった場合に，DVS 制御をかけることが考えられる．

謝辞

本研究の機会を与えていただいた富田眞治教授に深甚なる謝意を表します．

また，本研究に関して適切なご指導を賜った福井大学の森眞一郎教授，奈良先端科学技術大学院大学の中島康彦教授，東京大学の五島正裕助教授，大阪工業大学の小西将人講師，富田研究室の嶋田創助手，本学法学研究科の三輪忍情報担当助手，富田研究室の中島志保秘書に心から感謝いたします．

さらに，日頃から御討論頂いた京都大学情報学研究科通信情報システム専攻富田研究室の諸兄に心より感謝いたします．

参考文献

[1] 小林良太郎, 安藤秀樹, 島田俊夫: データフロー・グラフの最長パスに着目したクラスタ化スーパースカラ・プロセッサにおける命令発行機構, 並列処理シンポジウム JSPP 2001, pp. 31–38 (2001).

[2] Fields, B. and Blodik, S. R. R.: Focusing Processor Policies via Critical-Path Prediction, *28th Int'l Symp. on Computer Architecture (ISCA-28)* (2001).

[3] Fields, B., Bodik, R. and Hill, M. D.: Slack: Maximizing Performance under Technological Constraints, *29th. Int'l Symp. on Computer Architecture (ISCA-29)* (2002).

[4] Tune, E., Liang, D., Tullsen, D. M. and Calder, B.: Dynamic Prediction of Critical Path Instructions, *7th Int'l Symp. on High Performance Computer Architecture (HPCA7)* (2001).

[5] Grunwald, D.: Micro-architecture Design and Control Speculation for Energy Reduction, *Power Aware Computing*, Kluwer, ISBN 0-306-46786-0, chapter 4 (2002).

[6] 千代延昭宏, 佐藤寿倫: プログラム実行時における命令の重要度決定に関する検討, 情報処理学会研究報告 2003–ARC–154 (SWoPP 2003), pp. 1–6 (2003).

[7] Casmira, J. and Grunwald, D.: Dynamic Instruction Scheduling Slack, *Kool Chips Workshop (in conjunction with MICRO-33)* (2000).

[8] 劉小路, 小西将人, 五島正裕, 中島康彦, 森眞一郎, 富田眞治: クリティカリティ予測のためのスラック予測, 先進的計算基盤システムシンポジウム SACSIS 2004, pp. 187–196 (2004).

[9] 福田匡則, 小西将人, 五島正裕, 中島康彦, 森眞一郎, 富田眞治: グローバル分岐履歴を用いたスラック予測器, 情報処理学会研究報告 2004–ARC–159 (SWoPP 2004), pp. 25–30 (2004).

[10] Palacharla, S., Jouppi, N. P. and Smith, J. E.: Complexity-Effective Superscalar Processors, *Proc. 24th Int'l Symp. on Computer Architecture (ISCA24)* (1997).

[11] Zyuban, V. and Kogge, P.: Inherently lower-power high-performance superscalar architectures, *IEEE Transactions on Computers* (2001).

[12] 服部直也, 高田正法, 岡部淳, 入江英嗣, 坂井修一, 田中英彦: クリティカル

パス情報を用いた分散命令発行型マイクロプロセッサ向けステアリング方式, 情報処理学会論文誌：コンピューティングシステム, Vol. 45, No. SIG 6(ACS 6), pp. 12–22 (2004).

[13] 服部直也, 高田正法, 岡部淳, 入江英嗣, 坂井修一, 田中英彦: 発行時間差に基づいた命令ステアリング方式, 情報処理学会論文誌：コンピューティングシステム, Vol. 45, No. SIG 11(ACS 7), pp. 80–93 (2004).

[14] Keller, J.: The 21264: A Superscalar Alpha Processor with Out-of-Order Execution, *9th Annual Microprocessor Forum* (1996).

[15] 千代延昭宏, 佐藤寿倫, 有田五次郎: 低消費電力プロセッサアーキテクチャ向けクリティカルパス予測器の提案, 情報処理学会研究報告 2002–ARC–149 (SWoPP 2002), pp. 1–6 (2002).

[16] Yeager, K. C.: The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, Vol. 16, No. 2, pp. 28–40 (1996).

[17] 福山智久, 福田匡則, 三輪忍, 小西将人, 五島正裕, 中島康彦, 森眞一郎, 富田眞治: スラック予測を用いた省電力アーキテクチャ向け命令スケジューリング, 先進的計算基盤システムシンポジウム SACSIS 2005, pp. 123–132 (2005).

[18] 近藤正章, 藤田元信, 中村宏: 演算部とデータ供給部の動的周波数変更による低消費電力化手法の検討, 情報処理学会研究報告 2003–ARC–154 (SWoPP 2003), pp. 97–102 (2003).