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

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[†]

我々は，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．したがって，いわゆるクリティカルな命令のスラックは 0 サイクルである．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，スラックの値が 1 以上である命令を低速 / 低消費電力の演算器で実行するという方法で演算器の省電力化を図った．評価したところ，4.5% の IPC の低下で 19% の EDP を削減できることが分かった．

Instruction Scheduling for Low-Power Architecture with Slack Prediction

Tomohisa Fukuyama,[†] Masanori Fukuda,[†] Miwa Shinobu,[†]
Masahito Konishi,[†††] Masahiro Goshima,[††] Yasuhiko Nakashima,[†]
Shin-ichiro Mori[†] and Shinji Tomita[†]

We proposed an instruction criticality prediction technique based on 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 as the slack of the instruction. Thus the slack of a critical instruction is zero cycles. The slack value is stored to the prediction table to be a predicted value for the next time. This paper describes instruction scheduling with slack. Evaluation result shows EDP is reduced 19% at the cost of 4.5% IPC degradation.

1. 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[1] 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[2]．クリティカリティの情報は，省電力アーキテクチャにおいても有用である．例えば，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[3)~7)]．

さて，従来このような研究の多くは，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，以下のような問題点がある：

(1) **論理的**　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．

一方我々は，クリティカル・パスではなく，命令の**スラック** (slack)[8] によって，命令のクリティカリティを測ることを提案した[9),10)]．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．

† 京都大学
Kyoto University
†† 東京大学
University of Tokyo
††† 大阪工業大学
Osaka Institute of Technology

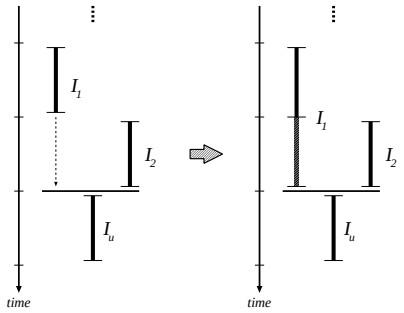

図 1　I_1 を 1サイクル遅らせた場合

スラック予測器を用いることで，前述し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．

(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．

スラックの計算

図 1 に，命令が実行される様子を表わすタイム・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．

同図では，命令 I_1 が定義した結果を最初に使用する命令は I_u となっている．すると 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 による使用時刻 t_u の差，つまり：

$$s = t_u - t_d - 1 \qquad (1)$$

によって得られる．この式に従えば，図 1 の I_1 のスラックは 1サイクル，I_2 のスラックは 0サイクルとなる．なお，以下ではスラックの単位を省略し，「命令 I_1 のスラックは 1」のように言うことにする．

本稿では，スラック予測器による予測結果を，実際に命令スケジューリングに応用し，演算器における消費電力を削減する方法について述べる．

スラックの利用

スラックが 1 以上の命令を低速 / 低消費電力の演算器で実行することによりプロセッサの性能を大きく低下させることなく省電力化を図る．

スラックが 1 以上の命令は，プログラムの実行時間を増大させることなく，その実行を遅らせることができる．例えば，図 1 左においてスラックが 1 と計算された命令 I_1 を，低速な演算器で実行することにより演算実行結果が使用可能になる時刻を 1サイクル遅らせた様子を図 1 右に示す．

図 1 右では，I_1 と I_2 が前回と同様のタイミングで実行された場合を示している．I_1 の実行結果が得られる時刻が 1サイクル遅れても，I_u の実行開始は遅れておらず，プログラムの実行時間は増大しないことに注意されたい．

上述したように，本来スラック予測とはクリティカリティを多値的に表現しているものである．しかし，本稿の命令スケジューリングでは，スラック0 であるか 1 以上であるかが重要となる．すなわち，ここで言うスラック予測とはスラック0 予測と言えよう．

以下，2 章でスラック予測器について，3 章でスラック予測を用いて省電力アーキテクチャ向け命令スケジューリングを実現する際のスラックの計算方法について，4 章でグローバル分岐履歴と 2ビット・カウンタの導入について述べ，5 章で本稿で述べた命令スケジューリングの評価結果を示す．

2. スラック予測器

本章では，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 2.1 節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 節で予測器に対する登録，参照といった操作について説明する．

2.1 スラック予測器の構成

スラック予測器は，主に以下の 2 種の表からなる：

(1) **スラック表**　命令のスラックを記録する表．

(2) **定義表**　各データに対し，以下を記録する：

(a) **定義時刻**　そのデータが定義された時刻

(b) **定義命令**　そのデータを定義した命令

(c) **フラグ**　命令を遅らせて実行したかどうか

フラグについては，3.4 節で詳しく述べる．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．一方，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．

定　義　表

定義表は，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，フラグ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とフラグの値を読み出せば，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，フラグ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

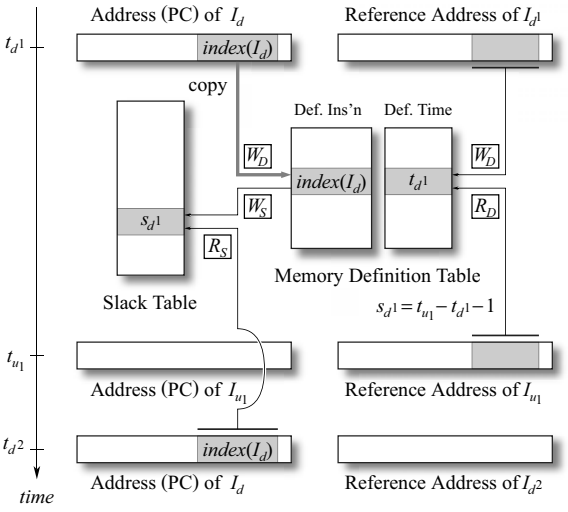

図 2　予測器に対する操作（ストア命令の場合）

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，フラグを格納するフィールドを付加したものと考えてよい．

(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　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，$\boldsymbol{I_{d^n}}$ 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 $\boldsymbol{I_{u_n}}$ 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 $\boldsymbol{t_{d^1}}$ および $\boldsymbol{t_{u_1}}$ 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：

$\boldsymbol{W_D}$　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：

$\boldsymbol{R_D}$　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．

$\boldsymbol{W_S}$　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：

$\boldsymbol{R_S}$　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．

2.3　スラック予測器へのアクセス・タイミング

図 3 と**図 4** に，前述したストア命令の場合と，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それぞれ示す．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には，表へのインデクスとして用いる物理レジス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．

(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, 4 中，D）に間に合うように読み出せばよい．

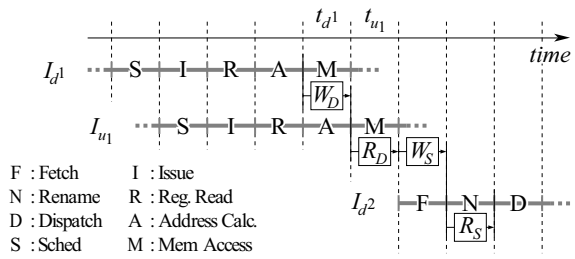

図 3 予測器に対する操作のタイミング（ストア命令の場合）

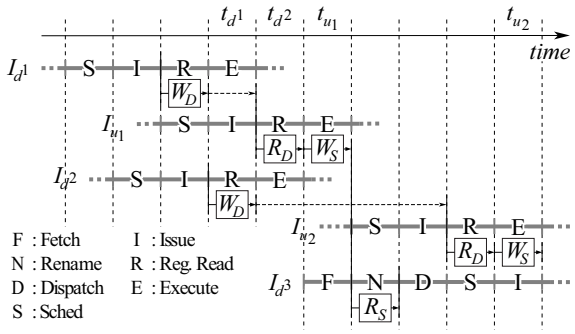

図 4 予測器に対する操作のタイミング（ストア命令以外の場合）

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[11] 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．

2.5 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ず，スラック表への登録を行うことはできない．

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，5 章で述べるように，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．

3. スラックの計算方法

本章では，スラック予測を命令スケジューリングに応用する際に用いる 3 種類のスラックの計算方法を提案する．以下，3.1 節で基本的な計算方法，3.2 節で見かけのスラック，3.3 節で見かけ上クリティカルになっている命令について述べ，最後に，3.4 節で 3 つの計算がスラック予測器の定義表にフラグを用意することによって可能であることを示した．

3.1 スラックの基本計算

1 章で述べたように，スラック s の基本となる計算式は，定義命令 I_d がデータを定義した時刻 t_d とそのデータを最初に参照する命令 I_u がデータを使用する使用時刻 t_u の差，つまり式 (1)：

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

によって与えられる．

ただし，**図 5** 左下において，1サイクル遅らせた命令 I_1 については，そのスラックを再計算する際注意が必要である．式（1）にしたがって計算すると，遅らせた命令 I_1 のスラックは 0 となる．すると次回 I_1 をスケジューリングする際には，I_1 はクリティカルであると判断され，実行時には 1サイクル遅らせることなく実行されてしまう．その結果，次回には再び図 5 左上のように実行され，以降，上下の状態を繰り返すことになる．

しかし，この繰り返しが発生するのは，遅らせた命

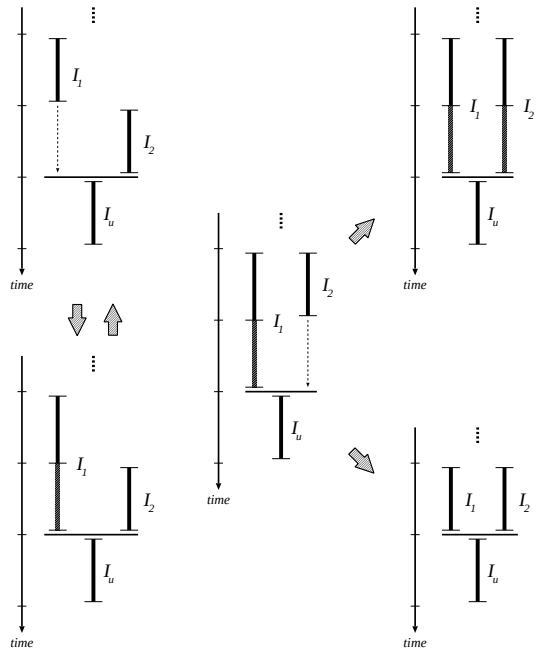

図 5 状態の遷移

令のスラックが式（1）によって 0 と計算される時のみ，つまり，遅らせなかった場合のスラックが 1 の命令のみである．よって，遅らせずに実行した時にスラックが常に 1 となる命令は，毎回遅らせることができるにもかかわらず，2 回に 1 回しか遅らせることができなくなってしまう．

3.2 見かけのスラック

遅らせた命令 I_1 の 0 というスラックは，故意に 1サイクル遅らせたことによって生ずる「見かけの」のスラックであるといえる．I_1 の真のスラックは，図 5 左上の場合と同様，遅らせなかった場合の定義時刻をスラックを元に計算すればよい．この遅らせなかった場合の定義時刻を**実効定義時刻**と呼ぶことにする．つまり，遅らせなかった場合の定義時刻 t_e は：

$$t_e = t_u - 1 \tag{2}$$

によって与えられる．この場合のスラックは，使用時刻と実効定義時刻の差，つまり：

$$\begin{aligned} s &= t_u - t_e - 1 \\ &= t_u - t_d \end{aligned} \tag{3}$$

によって得られる．

このようにすれば，発振が生じ 2 つの状態を繰り返すことはなくなる．

3.3 見かけのスラック（その 2）

前節までの説明では，命令 I_1 と I_2 は，毎回同じタイミングで実行が開始される場合を考えていた．しかし，必ず前回と同じタイミングで実行開始されるとは限らない．図 5 中に示すように，I_1 と I_2 が同時に実行開始されることもあり得る．さらには，I_2 の方が早く実行される可能性もある．このような場合，前回スラックが 1 以上であったため 1サイクル遅らせた命令 I_1 が，今回クリティカルになってしまう．このような命令を**見かけ上クリティカルになっている命令**と呼ぶことにする．

図 5 中のような場合，I_1 のスラックは実効定義時刻に基づいて，I_2 のスラックは基本どおりに，それぞれ 1 と計算され，次回実行時には図 5 右上のようになる．しかし実際には，図 5 右下のように I_1，I_2 のスラックを共に 0 とし実行した方が I_u を 1サイクル早く実行することができる．

スラックを共に 1 と計算した図 5 右上の場合は，すべての命令を無条件に 1サイクル遅らせたのと変わりがない．また，一旦この状態になるとスラックが 1 のまま 0 に戻らないという可能性もある．

結局，図 5 中の I_1、I_2 のように見かけ上クリティカルになっている命令がある場合には，次回は図 5 右下のようにするために，I_1、I_2 のスラックを共に 0 とする．

図 5 左下のように，遅らせていない命令（図では I_2）がクリティカルになっている状況では，I_1 は見かけ上クリティカルになっている命令とは言わない．逆に言えば，図 5 中のように，クリティカルになっている命令のすべて（図では，I_1 と I_2）が遅らせた命令である場合に，これらの命令を見かけ上クリティカルになっている命令と言う．

見かけ上クリティカルになっている命令がある場合，命令を遅らせたサイクル数（図では 1サイクル）だけ無駄に実行が遅れている．したがって各命令の実効スラックは，式（1）,（3）により得られたスラックの値から遅らせたサイクル数を減ずればよい．

つまり，見かけ上クリティカルになっている命令が生じている場合，遅らせた命令のスラックは式（3）から 1 減じた式，つまり式（1）と同じ式により得られ，遅らせなかった命令は式（1）から 1 減じた式：

$$s = t_u - t_d - 2 \tag{4}$$

によって得られる．

3.4 問題の発見

２つの見かけのスラックの問題を発見するには，その命令が遅らせたものかどうかを表すフラグが必要になる．そこで，スラック予測器の定義表にフラグを用意し，命令を遅らせていない場合には 0 を，遅らせた場合には 1 を命令の実行時に記憶する．

まず、式（1）により暫定的なスラックを求める．この暫定的なスラックの値が 0 となる命令のフラグが全

て 1 であるとき，見かけ上クリティカルになっている命令がある状況が生じていることとなる．

このように見かけ上クリティカルになっている命令がある状況はフラグを容易することで発見でき，この場合の実効スラックはフラグが 0 である命令に関しては暫定的なスラックから 1 減じた値，つまり式（4）で，フラグが 1 である命令に関しては暫定的なスラックの値，つまり式（1）で得ることができる．

見かけ上クリティカルになっている命令がない場合には、フラグが 0 の命令は暫定的なスラックの値，つまり式（1）で得られる値が、フラグ 1 の命令は暫定的なスラックに 1 足した、つまり式（3）で得られる値がそれぞれ実効スラックとなる．

このようにして得られた実効スラックの値をスラック表に記憶する．

4. 予測精度の向上

本章では，スラック予測の精度の向上を試みるために，スラック予測器にグローバル分岐履歴と飽和型 2 ビット・カウンタを導入する方法について述べる．

4.1 グローバル分岐履歴の導入

スラック予測器にグローバル分岐履歴を導入することにより，予測精度が向上することが認められている[10]．グローバル分岐履歴の導入にはスラック表のインデクスとして，gshare 分岐予測器などと同様に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た．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 (Value History Table) 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，競合を完全に排除している．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．

4.2 飽和型 2ビット・カウンタの導入

これまで述べてきたスラック予測の方法では，スラックの最新の値 (last value) を記憶している．一方，1 章でも述べたようにスラックが 1 以上の命令を 1 サイクル遅らせる場合，スラックの厳密な値は必要なく，スラックが 0 かどうかを記憶しておけばよい．

結局，スラックが 0 かどうかを予測する上では，これまで述べてきた方法は，いわば 1 ビット・カウンタによる予測であると言える．

そこで，飽和型 2 ビット・カウンタを用いることにより，命令のローカルな履歴を反映させることが考えられる．スラックの計算による値が 0 であればカウンタを 1 減じ，1 以上であれば 1 足す．そしてスラック予測を行う際に，スラック表から読み出した値が 0，1 であればスラックは 0 と予測し，2，3 であればスラックは 1 以上と予測する．

5. 評　　価

シミュレーションにより，3 章で提案した 3 種類のスラックの計算方法を実際に命令スケジューリングに用いた場合について評価する．

なお，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 2 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[10] 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．

5.1 評 価 方 法

SimpleScalar ツールセット (ver. 3.0) の sim-outorder シミュレータに対して，スラック予測器を実装し，SPEC ベンチマークを用いて本稿で述べた命令スケジューリングによる省電力アーキテクチャの効果を測定した．測定には**表 1** に示す SPEC CINT95 の 8 つのプログラムを実行した．

コンパイラは，gcc (ver. 2.7.2.3) を用いた．最適化オプションは，-O6 -funroll-loops である．

プロセッサのモデル

評価では，本稿で述べた命令スケジューリング方式を整数演算器に適用する．用いたプロセッサの整数系の

表 1　SPEC CINT95 ベンチマーク・プログラム

プログラム	入力セット	実行命令数
099.go	9 9	132M
124.m88ksim	dcrand.big	120M
126.gcc	genrecog.i	32M
129.compress	10000 q 2131	35M
130.li	train.lsp	183M
132.ijpeg	vigo.ppm -GO	26M
134.perl	primes.in	10M
147.vortex	persons.250	157M

☆ タグは，一部省略できる可能性がある[12]．

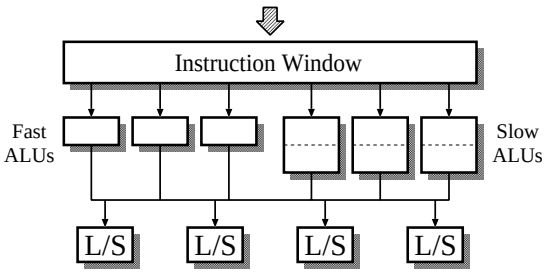

図 6 プロセッサのモデル

ブロック図を**図 6** に示す.

同図に示すように，プロセッサは高速な ALU と低速な ALU をそれぞれ 3 個ずつ，合計 6 個持つ．高速な ALU と低速な ALU のレイテンシは，それぞれ，1 サイクル，および，2 サイクルである．低速な ALU は，高速な ALU をパイプライン動作させるものに相当し，レイテンシは 2 に増加するが，スループットは 1 のままである．これらの ALU の電源電圧は，それぞれ，1.1 V, および，0.7 V とした[13]．スラックが 0 と予測された命令を高速な ALU，1 以上と予測された命令を低速な ALU にスケジューリングする．

実際にキャッシュへのアクセスを行うロード / ストア・ユニットは，4 個あり，同図のように接続されている．すなわち，ロード / ストア命令のアドレス計算は，これらの ALU のいずれかで実行される．アドレス計算を実行した ALU とロード / ストア・ユニットの間は，完全結合とした．すなわち，アドレス計算を行った ALU とロード / ストア・ユニットの組み合わせには制限はない.

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.

プロセッサのそれ以外のパラメタを**表 2** に示す.

テーブルのパラメタ

表 3 に，テーブルのパラメタをまとめる．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.

スラックの計算方法

それぞれの命令をどちらの ALU で実行するかを決定する際に用いるスラックの計算方法は 3 章で提案した次の 3 つとする.

BASE 3.1 節で述べた基本となる計算方法．発振が生じ遅らせることができる命令にもかかわらず遅らせずに実行されてしまう可能性がある.

EDT 3.2 節で述べた実効定義時刻（ EDT： effective definition time）を用いる計算方法．BASE とは逆に命令を遅らせ過ぎる状況が起こる.

ACC 3.3 節で述べた見かけ上クリティカルになっている命令な命令を発見し遅らせ過ぎる状況をなくすことにより, より正確な（ACC：accurate）スラックを求める計算方法.

上記の 3 つのそれぞれの場合について，前回の履歴にのみ依存している場合と飽和型 2 ビット・ カウンタを用いた場合の 2 通りをシミュレーションし，それぞれ **-1b**，**-2b** として区別する.

それぞれの場合について，性能，ALU における消費電力，低速の ALU で実行した命令の割合を測定し，

表 2 プロセッサの各パラメタ

パラメタ	サイズ
FETCH 幅	8
命令ウィンドウ	16
ISSUE 幅	8
COMMIT 幅	8
Functional Units	Int:6，FP:4，MemPort:4
レイテンシ（Latency/Throughput）	iALU 1/1，iMULT 3/1，iDIV 20/19，fADD 2/1，fCMP 2/1，fCVT 2/1，fMULT 4/1，fDIV 12/12，fSQRT 24/24
分岐予測	4K-entry 8-histry-length gshare predictor 6cycle extra misprediction latency 8-entry return address stack

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.

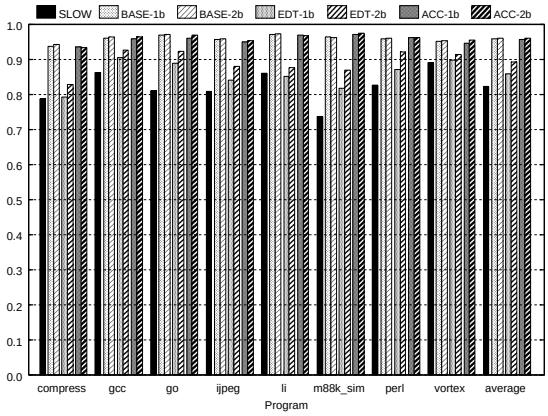

図 7 性能の比

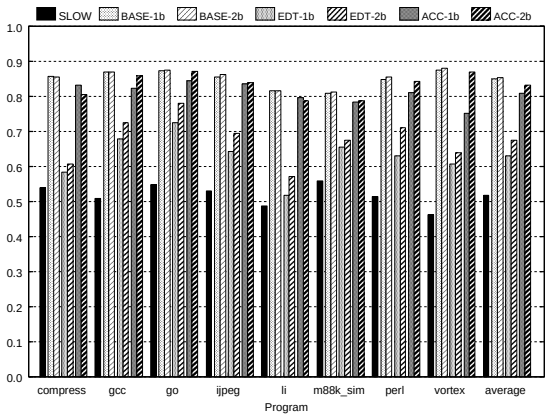

図 8 EDP の比

全ての ALU を高速にした場合（FAST とする）や低速にした場合（SLOW とする）と比較することによって評価を行う．

なお，ここで言う消費電力とは，ALU のみにおける消費電力でありプロセッサ全体の消費電力ではない．よって，スラック表や各定義表における消費電力は考慮していない．

5.2 評価結果

図 7，**図 8**，**図 9** に結果を示す．図 7 は性能を，図 8 は EDP を，図 9 は低速な ALU で実行した命令の割合を示している．図 7，図 8 には，9 組のバーがあり，左の 8 組は SPEC CINT95 の 8 つのプログラムに対応しており，最右 1 組はそれらの平均である．図 7 は，FAST に対する IPC の比を，図 8 は EDP の比を表わしている．各組には，バーが 7 本ずつあり，それぞれ左から前節で述べた SLOW，BASE-1b，BASE-2b，EDT-1b，EDT-2b，ACC-1b，ACC-2b の場合のものを指している．

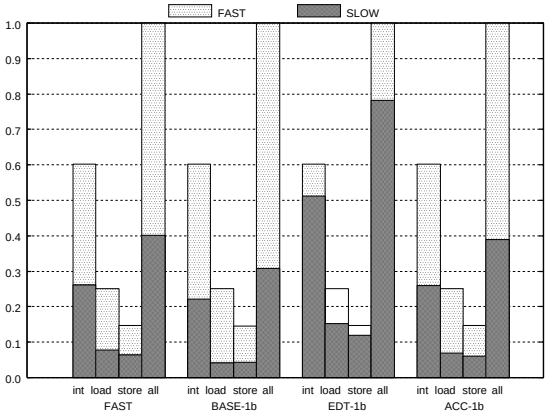

図 9 低速な ALU で実行した命令の割合

一方，図 9 には 4 組のバーがあり，左から FAST，BASE-1b，EDT-1b，ACC-1b の場合の 8 つのプログラムの平均である．それぞれの組の 1 番左の薄いバーは整数演算器で実行した全命令に対する通常の整数演算命令の比で，2 番目のバーはロードの，3 番目のバーはストアのアドレス計算の比で 4 番目のバーはそれらの合計である．1 番左の FAST の組の濃いバーはそれぞれの命令における全命令に対するスラックが 1 以上の命令の比を，薄いバーはスラック0 の命令の比を表わしている．また，その他の組の濃いバーは低速な ALU で実行した命令の比を，薄いバーは高速な ALU で実行した命令の比を表わしている．

図 7，図 8 を見ると，SLOW での IPC 低下は 18%，EDP 削減は 48% であるのに対し，

- BASE-1b では，FAST に比べ IPC，EDP がそれぞれ約 4.5%，約 14.5% 低下した．
- EDT-1b では，FAST に比べ IPC が約 14% 低下しているものの，約 37% もの EDP を削減できている．
- ACC-1b は，IPC では BASE とほぼ同じ約 4.5% であったが，EDP では約 19% と BASE よりも 4.5% 多く削減できている．
- -2b では，それぞれの場合の飽和型 2 ビット・カウンタを用いなかった場合よりも IPC，EDP の比が共に若干減少した程度であった．

一方，図 9 を見ると，FAST では，整数演算命令の 44%，ロード命令のアドレス計算部分の 31%，ストア命令のアドレス計算部分の 42% がスラック1 以上である．3 つを合わせると整数演算器を用いる全命令の 40% がスラック1 以上であることがわかる．

これに対して，

- BASE-1b では低速な ALU で実行した命令は FAST

においてスラック1 以上である命令よりも少ない．これは，発振が生じ遅らせることができる命令を遅らせられなかったためである．

- 逆に，EDT-1b は低速な ALU で実行した命令が大幅に多い．これは，3.3 節で述べた見かけ上クリティカルになっている命令が存在し，命令を遅らせ過ぎているためである．
- ACC-1b は FAST とほぼ一致しており，より正確にスラックの計算がされていると考えられる．

ES-1b は厳密にスラックを計算できているにもかかわらず，全ての ALU を高速な演算器にしたときに比べ 4.5% の性能が低下している．この原因として次の 2 つが挙げられる：

- 本当はクリティカルである命令を低速な ALU で実行した，つまりスラック予測の予測ミスによる性能低下．
- 高速な ALU を 6 つから 3 つに減らしたためにクリティカルな命令を 1サイクルにつき最大 3 命令までしか実行できない．それによって，使用する ALU に偏りが生じた場合には ALU が不足してしまい，性能低下の原因となる．

そこで，ACC-1b においてスラックが 1 以上の命令を実行する低速な ALU のレイテンシを高速な ALU と同様 1サイクルとしてシミュレーションを行った．こうすることにより，スラックの予測ミスが原因となる性能低下をなくし，使用 ALU の偏りが原因となる性能低下のみを図ることができると考えられるからである．このシミュレーションの結果，FAST に対する性能は平均して約 98.5% であった．よって，ACC-1b の 4.5% の性能低下のうち，約 1.5% が使用 ALU の偏りによるもので，スラックの予測ミスによる性能低下は約 3% 程度であると推測できる．

5.3 関 連 研 究

千代延らは文献 7）において，小林らが提案しているデータフローグラフの最長パスを特定するためのパス情報テーブル[2)]（PIT：Path info Table）をクリティカル・パス予測器として，省電力アーキテクチャの設計に用いている．クリティカル・パス予測器によりクリティカル・パス上にないと予測された命令を低速 / 低消費電力の ALU で実行し性能と消費電力の計測を行い，その結果を彼ら自身が提案しているクリティカル・パス予測器と比較することにより性能と省電力化の評価を行っている．PIT を用いた省電力アーキテクチャでは約 5% の性能低下で 20% の EDP を削減できている．

使用したベンチマーク・プログラムやプロセッサのモデルが若干異なるものの本稿でのスラック予測を用いた省電力アーキテクチャでは約 4.5% の性能低下で 19% の EDP を削減できており，PIT と同等の性能と省電力化を実現できている．

また，PIT は複雑なハードウェアを必要とし，実際に実装することはとても困難である．一方，これまでも述べた通りスラックは容易に求めることができることにも注意が必要である．

命令のクリティカリティを判定する際に，クリティカル・パスに基づいた PIT を用いた場合と，我々が提案する履歴に基づいたスラック予測を用いた場合とで，同様の結果が得られていることも興味深いところである．

6. お わ り に

本稿では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について述べた．評価結果は，スラック予測が省電力アーキテクチャに十分応用できることを示している．

今後は，より厳密なスラックを求めるために以下のことを検討する予定である．

スラックの伝搬

本稿で用いたスラックは文献 9）にもあるように近似的なものであり，「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」というスラックの厳密な定義に従っていない．たとえば，**図 10** 左において，本稿で用いたスラックの求め方に基づけば，同図の命令 I_1，I_2 のスラックはそれぞれ 0，2 となり，本稿の省電力アーキテクチャを適用すると図 10 中のように I_2 のみが低速な ALU で実行され，I_1 は高速な ALU で実行されることとなる．しかし，厳密には I_1，I_2 のスラックは共に 2 であり，図 10 右のように両方の命令が共に低速な ALU で実行された方がよい．

そこで，我々は I_2 のスラックを I_1 に伝搬させるなどしてより厳密なスラックを求めることを検討している．

謝辞

本研究の一部は，日本学術振興会 科学研究費補助金 基盤研究 S（課題番号 16100001），および 21 世紀 COE プログラム（課題番号 14213201）による．

参 考 文 献

1) Keller, J.: The 21264: A Superscalar Alpha Processor with Out-of-Order Execution, *9th Annual Microprocessor Forum* (1996).
2) 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

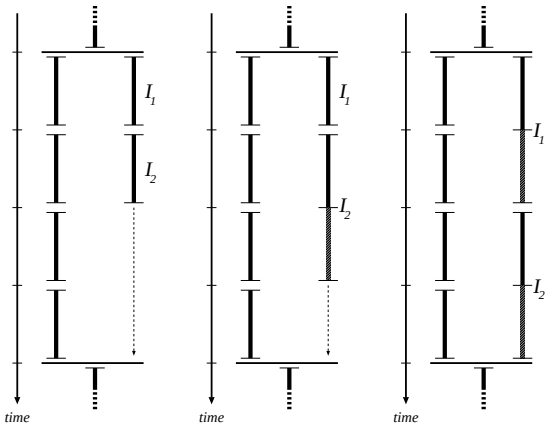

図 10 スラックの伝搬

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).

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

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

5) 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).

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

7) 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).

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

9) 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).

10) 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).

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

12) 佐藤寿倫, 有田五次郎: タグビット幅を考慮したデータ値予測機構のハードウェア量削減, 信学技報 CPSY 2000–3 (2000).

13) Levy, M.: SAMSUNG Twists ARM Past 1GHz, *Information Quarterly*, No. 1 (2002).