グローバル分岐履歴を用いたスラック予測器

福田匡則[†] 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，グローバル分岐履歴を用いたスラックを予測器について述べる．Gshare 分岐予測器と同様，命令のアドレスとグローバル分岐履歴の排他的論理和によって予測表にアクセスする方法を評価した．グローバル分岐履歴長を 0 から 3 まで増やしたところ，予測精度がわずかながら向上することが認められた．

Slack Predictor with Global Branch History

Masanori Fukuda,[†] 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 a slack predictor with global branch history. Exclusive-OR of an instruction address and global branch history is used to access the prediction table in the same way of the gshare branch predictor. The prediction accuracy is slightly improved as the length of global branch history is increased from 0 to 3.

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)]．

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 (dynamic voltage scaling) 制御を行うことを提案している[8] が，これも上述の省電力アーキテクチャの延長線上にあると考えてよい．キャッシュ・ミスの処理を待ってプロセッサがストールしているときには，ミスを起こしたロード命令はクリティカルであり，その他の命令はすべてクリティカルでない．この場合にプロセッサを低電圧化することは，クリティカルでない命令の実行に関わるハードウェアを低電圧化することと考えることができる．

さて，従来このような研究の多くは，論文の題目にも顕著に表われているように，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

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

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．

その上，本来の目的のためには，命令がクリティカル・パス上にあるかどうかを知りたいのではない．本当に必要なのは，その命令の実行の遅れがプロセッサによるプログラムの実行時間をどの程度増大させるか，すなわち，その命令のクリティカリティそのものである．

そこで我々は，クリティカル・パスではなく，命令の**スラック** (slack)[9] 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[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 である．

文献 10) では，各命令のローカルな履歴に基づくスラック予測器について述べた．本稿では，分岐予測では既に一般的になっているように，スラック予測器にグローバルな分岐履歴を導入する．条件分岐と同様，スラックにもグローバル分岐履歴に対する依存性があると考えられるからである．

なお本稿では，スラック予測器の予測精度を評価するに留め，上述した命令スケジューリングや省電力アーキテクチャにスラック予測器を応用した場合の結果までは評価しない．スラック予測器に必要とされる詳細は応用によって異なり，研究の初期から応用を絞ると，一般性を失う恐れがあるためである．

以下，2 章でスラック予測器について述べ，3 章で予測精度を評価した結果を示す．

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．最後に 2.5 節で，グローバル分岐履歴の導入の方法について説明する．

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

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．

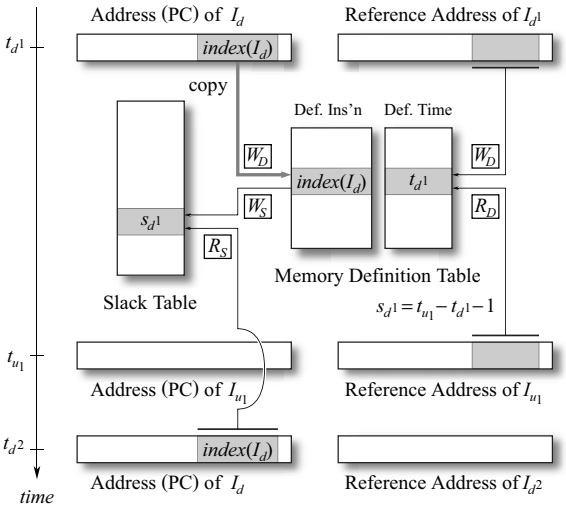

図 1　予測器に対する操作（ストア命令の場合）

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，定義時刻と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，当然のことながら，システム中のすべてのデータに対して定義時刻と定義命令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と定義命令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 1 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 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，状態回復がそれだか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，3 章の評価で用いる MIPS R10000 プロセッサの場合，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．

2.4 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，3 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．

2.5 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 (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．

3. 評　　価

シミュレーションにより，スラック予測器の評価を行った．なお本稿では，前述したとおり，スラック予測器の予測精度を求めることのみを目的としており，スラック予測器を命令スケジューリングや省電力アーキテクチャに応用した場合の結果までは評価していない．

3.1 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予測精度を測定した．**表 1** に示す 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．

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	122M
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

表 2　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
	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.

☆ タグは，一部省略できる可能性がある[11]．

プロセッサのモデル

プロセッサのモデルとしては，MIPS R10000 プロセッサ[12] を用いた．R10000 プロセッサは，整数，ロード / ストア，浮動小数点のそれぞれに，ディスパッチ幅，フェッチ幅 2 命令，深さ 16 命令の命令ウィンドウを持つ．キャッシュ，メモリのパラメタを**表 2** にまとめる．

ただし，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．また，分岐予測には，同ツールセットに用意されている Bimodal 予測器を用いた．

テーブルのパラメタ

表 2 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．

3.2 評 価 結 果

図 3 に，絶対誤差の度数分布を示す．

x 軸は予測誤差を，y 軸はその絶対誤差の発生頻度を表す．同図は両対数グラフであることに注意されたい．また，対数グラフで表すため，x 軸の目盛は絶対誤差 +1 としている; すなわち，同グラフで $x = 1$ である点が，実際の絶対誤差が 0 である頻度を示している．

同グラフには，4 本の曲線があり，それぞれ，グローバル分岐履歴長が 0，1，2，および，3 の場合を表している．

また，**図 2** は，図 3 の $x = 1$ 付近を拡大したものであり，こちらの目盛は対数ではない．

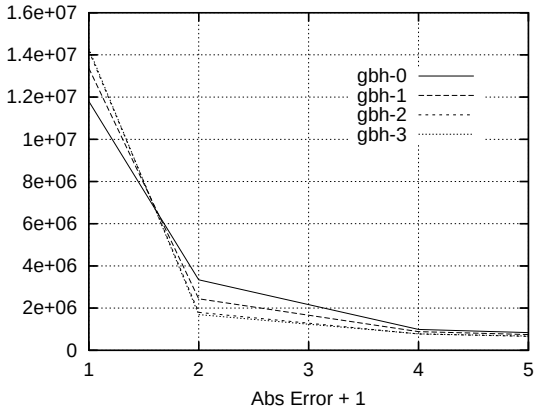

図 2　絶対誤差の度数分布

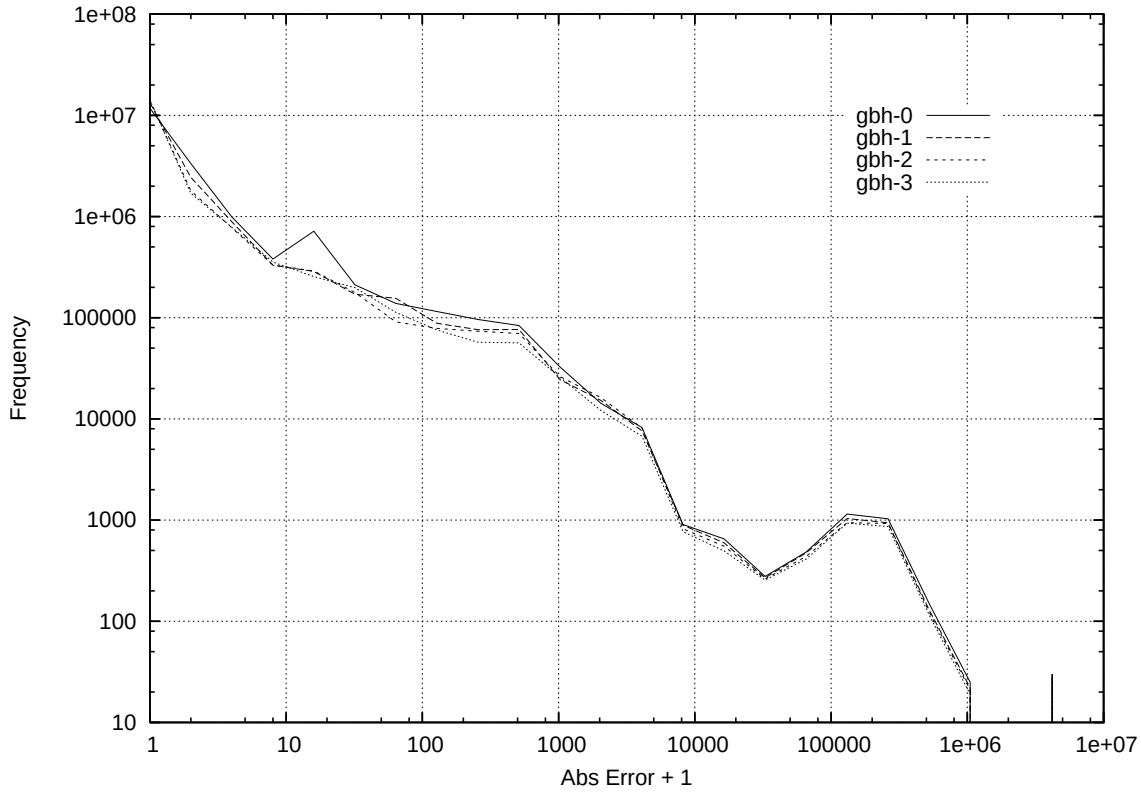

図 3　絶対誤差の度数分布

グローバル分岐履歴長が増えるにつれ，絶対誤差が 0 である頻度 （グラフ中，$x = 1$）が増大し，1 以上の値である頻度が減少していることが分かる．

特に，図 3 中，履歴長 0 では $x = 20$ 近辺にあるピークが，履歴長 1 では消えていることに注目されたい．これは，以下に述べる理由による．

図 4 と**図 5** に，履歴長 0 の場合と 1 の場合の，ストア命令のスラックの予測値と実際の値の度数分布をそれぞれ示す．

図 4 では，$(30, 8)$ と $(8, 30)$ のところにピークがあり，これが図 3 の $x = 22$ のピークの原因となっている．一方，履歴長を 1 とした図 5 では，このピークは $(8, 8)$ と $(30, 30)$ に移っている．

このことは，最近の分岐の結果によって 8 と 30 の

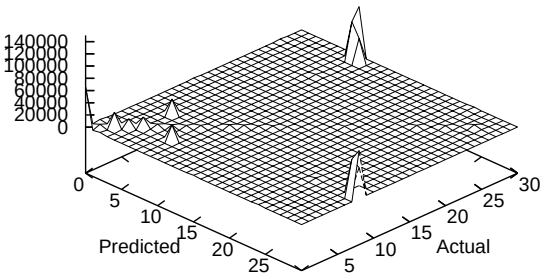

図 4　予測値 (Predicted) と実際の値 (Actual) の度数分布，履歴長 0

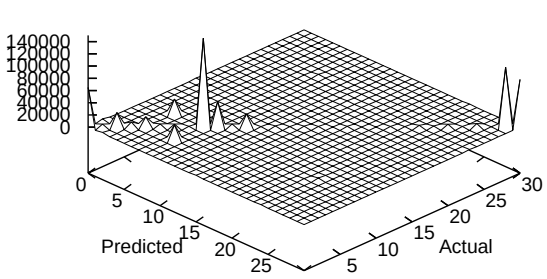

図 5　予測値 (Predicted) と実際の値 (Actual) の度数分布，履歴長 1

2 つの値を交互に取るストア命令があり，履歴長 1 のグローバル分岐履歴を導入することによって正しく予測できるようになったことを示している．

4. おわりに

本稿では，グローバル分岐履歴を導入したスラック予測器について述べた．評価結果は，グローバル分岐履歴を導入することによってスラック予測器の予測精度が向上することを示している．ただしその向上の度合いは，それだけでその有効性が明らかになるというほどではなかった．

今後は，スラックが 0 か 1 以上であるかを予測する方法などを検討する予定である．

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），21 世紀 COE プログラム（課題番号 14213201），ならびに，文部科学省 特定領域研究 S（課題番号 13224050）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) 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).
9) Casmira, J. and Grunwald, D.: Dynamic Instruction Scheduling Slack, *Kool Chips Workshop (in conjunction with MICRO-33)* (2000).
10) 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).
11) 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).
12) Yeager, K. C.: The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, Vol. 16, No. 2, pp. 28–40 (1996).