Vol. 42 No. SIG 9(HPS 3) 情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム Aug. 2001

両パス実行の性能評価と実行判定精度の改善

片山清和† 安藤秀樹† 島田俊夫†

両パス実行とは分岐命令の Taken パスと Not-Taken パスの両方のパスを実行する手法である．この手法は原理的に分岐予測ミスをなくすことができ，プロセッサの性能を向上させることが可能である．これまで両パス実行に対する研究がなされてきたが，近い将来に実現可能な規模のハードウェア量での評価が少なく，正確な有用性が十分には分かっていない．我々は，SPECint95 ベンチマークを用い，コンテキスト数，命令キャッシュポート数，命令キャッシュサイズによる性能に対する影響について評価を行い，コスト性能比の良い構成パラメータを明らかにした．そして，その構成パラメータにおいては，ハードウェアの複雑さの増加は大きくないことを示した．クロック・サイクル時間への影響を軽微と仮定し，実行サイクル数の評価を行った結果，近い将来実現可能と考えられる 8 命令発行，2 ポートの命令キャッシュ，4 ポートのデータキャッシュ，5 つのコンテキスト数のプロセッサにおいて，両パス実行は単一パス実行に対し，最大 20.5%，平均 11.2%の性能向上を見込めることが分かった．また，両パス実行の判定に使用する表における競合を抑制するために，分岐フィルタを導入した機構を提案した．これにより，従来の判定機構の場合に比べ，最大 8.1%の性能向上が得られることを確認した．

Performance Evaluation of Both-path Execution and Improvement of Execution Decision Accuracy

Kiyokazu Katayama,† Hideki Ando† and Toshio Shimada†

Both-path execution is to execute both paths following a branch instruction. This execution theoretically removes branch misprediction completely and can improve performance of processors. Although many researches on both-path execution have been done, there exist few evaluation studies under realistic hardware assumptions and the precise effect of both-path execution has not been clear yet. In this paper, we have evaluated the performance impact of the number of contexts, the number of instruction cache ports, and instruction cache size to find good cost/performance organization parameters. We have indicated that the increase of hardware complexity is not so large on the hardware organization we have found. Our cycle count evaluation shows that both-path execution which has 8-instruction issue width, 2-ported data cache, and 5 contexts is expected to improve performance by a maximum of 20.5% or by an average 11.2% over conventional single-path execution in a near-future superscalar processor assuming that the hardware complexity little affects the clock cycle time. In addition, we propose a mechanism that suppresses aliasing in the table used for decision of both-path execution by introducing branch filtering. We confirm that both-path execution with branch filtering achieves a maximum of 8.1% performance improvements, comparing to that with conventional decision mechanism.

1. はじめに

近年の高性能プロセッサは深いパイプラインやスーパスカラに代表される複数命令の同時発行機構により命令レベル並列性を引き出し，性能を向上させている．分岐予測精度は，こうした命令レベル並列性を利用するプロセッサの性能に大きな影響を与える．今後も命令発行幅はより広く，パイプラインはより深くなる傾向にあり，分岐予測精度に対する要求はさらに強まっていく[1]．この要求を満足させる方法として，分岐予測精度を向上させる方法と**両パス実行**を行う方法がある．

これまでさまざまな分岐予測機構が提案されてきた．なかでも **2 レベル分岐予測機構**[2)~4)]は，その分岐命令のアドレスと過去の分岐の履歴パターンの間に存在する相関を利用し，高い予測精度を達成している．しかし，これらの予測器はすでに 80%から 100%に近い予測精度を達成しており，さらに予測精度を向上させるのは困難である．

† 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University

一方，両パス実行は，Taken 方向と Not-Taken 方向の両方のパスを実行し，正しい分岐方向の結果のみを採用する手法である．両方のパスを実行するので，その分岐に関しては分岐予測ミスがなくなる．これにより予測ミスペナルティを被る頻度を下げることができる．しかしすべての分岐命令において両パス実行を行うには，複雑で大規模なハードウェアが必要となり非現実的である．そこで予測が困難な分岐に対してのみ両パス実行を行うよう制限し，必要なハードウェア量を減少させる研究がなされている[5)~9)]．

以上のように，両パス実行については研究はなされてきたものの，現実的なハードウェアでの評価に関する論文は少なく，真にどの程度有用であるかの知見は十分には得られていない．特にハードウェアの複雑さを決定する大きな要因となる，プロセッサ内で同時に扱うことができるパスの数に関する評価はされていない．本論文の第 1 の目的は，両パス実行の有用性を評価することである．

前述のように，両パス実行は必要な分岐に対してのみ行われるよう制限する必要があるが，我々は，これを助ける機構として**分岐フィルタ**[10)]が有用であることに着目した．分岐フィルタは，分岐方向の偏りが大きく予測が容易な分岐については予測を固定し，過去の分岐履歴を保持する**パターン履歴テーブル**（PHT: Pattern History Table）に履歴を格納しない．つまり，動的に分岐予測を行う対象を限定し，かつ，両パス実行を判断するためのテーブルでの競合を抑制することができる．これにより，両パス実行の判断の精度を向上させることができる．本論文の第 2 の目的は，両パス実行と分岐フィルタを組み合わせる機構を提案し，どの程度有効に機能するかを評価することである．

本論文の構成は以下のとおりである．2 章では，関連研究について述べる．3 章では，両パス実行アーキテクチャと両パス実行における問題をまとめる．4 章では分岐フィルタを付加した両パス実行の提案を行い，その構成について述べる．5 章で評価を行い，6 章で両パス実行実現のためのハードウェア量について述べ，7 章で本論文をまとめる．

2. 関 連 研 究

Uht ら[5)]はプログラムの制御構造を木構造として表現し，その根からの累積した分岐確率の大きい基本ブロックを優先的に実行することで効率を上げる DEE（Disjoint Eager Execution）を提案した．累積分岐確率計算は，分岐命令が解決されたときと分岐命令をフェッチしたときに行う必要があるので，計算の頻度が高い．また，分岐確率は分岐命令によって異なるため，正確な累積分岐確率計算を低レイテンシで行うことは困難である．そこで，累積分岐確率計算を単純化するため，すべての分岐の分岐確率を同一と見なしている．そのため正確な両パス実行判定を行うことはできない．

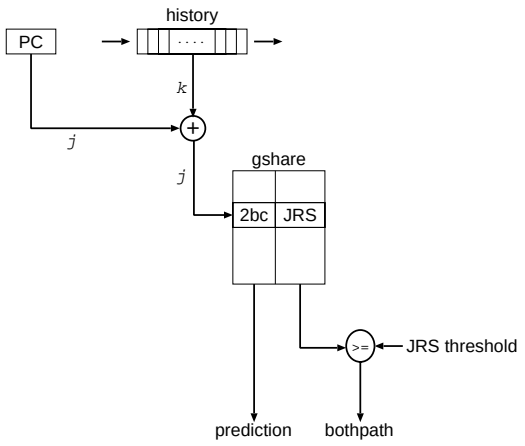


図 1 両パス実行判定機構の構成例
Fig. 1 Organization example for a both-path execution decision mechanism.

Heil ら[6)]は同時に両パス実行可能な分岐の数を 1 に制限した SDPE（Selective Dual Path Execution）モデルを提案し，レジスタ・ファイルやリオーダ・バッファをパス数に応じて多重化する必要性を述べた．また，分岐予測の信頼性を測定するリセッティング・カウンタを PHT のエントリごとに設け，両パス実行の判定を動的に行うことにより，判定精度を向上させた．

図 1 に両パス実行判定機構の構成例を示す．この例では，分岐予測機構として gshare を用いている．他の分岐予測機構でも同様に構成できる．両パス実行は分岐予測の信頼性が低い場合に行う．PHT の各エントリに予測の信頼性を測定するリセッティング・カウンタ（図では JRS）を付加する．分岐の結果が判明し，PHT を更新するときに，予測が成功していれば JRS を 1 増加させ，失敗していれば JRS を 0 にリセットする．両パス実行の判定は，あらかじめ定められた閾値（図では JRS threshold）以上であれば，両パス実行をしないと判定し，閾値を下回れば両パス実行を行うと判定する．この機構は，両パス実行に関するこれ以降のほとんどの研究において，両パス実行判定機構として用いられている．

SDPE において，Farrens ら[7)]は現実的なハードウェ

アで評価を行った．しかし，同時に両パス実行可能な分岐の数を 1 に制限しているため，一般に複数の分岐について両パス実行を許した場合の有効性については分かっていない．

Wallaceら[11]は SMT（Simultaneous MultiThreading）のアイドル状態のスレッドを両パス実行に用いる TME（Threaded Multipath Execution）を提案し，現実的なハードウェアにおける評価を行った．TME では，同時に両パス実行可能な分岐の数を 8 にまで緩和している．しかし，両パス実行は予測パスにしか許されておらず，一般性に欠ける．

Klauser ら[8),9)]はリセッティング・カウンタによる分岐予測の信頼度に基づき，同時に両パス実行可能な分岐の数を制限しない SEE（Selective Eager Execution）モデルを提案した．SEE では，**コンテキスト・タグ**と呼ばれるパス情報を，すべての命令に付加することにより，命令が含まれているパスをハードウェアが認識することを可能にした．しかし，ハードウェアの複雑さを決定する大きな要因である，プロセッサ内で同時に扱うことができるパスの数（コンテキスト数）に関する評価を行っていないなど，評価が十分でない．

3. 両パス実行

分岐命令において，後続の Taken パスと Not-Taken パスの両方向の実行を行うことを両パス実行と呼ぶ．予測を誤る分岐に対して両パスを実行すれば，予測ミスペナルティを被ることを回避でき，性能を向上させることができる．しかし両パス実行を行ううえで多くの問題も存在する．

本章では，まず本論文で仮定する両パス実行を行うプロセッサ構成を示し，両パス実行を行ううえで考慮すべき問題について述べる．

3.1 両パス実行アーキテクチャ

両パス実行を行うプロセッサ構成を**図 2** に示す．プロセッサ構成は SEE[8)]とほぼ同じである．スーパスカラ・プロセッサから，両パス実行のために追加された主な機構は，コンテキスト管理機構（Context Manager）と両パス実行判定機構（Both-Path Execution Decider）と複数のマップ表（Map Table）である．また図には示していないが，命令ウィンドウ（Register Update Unit）とロード/ストア・キュー（Load/Store Queue）にそのエントリの命令が属するパスを識別するためのタグ（以降コンテキスト・タグと呼ぶ）を格納するフィールドが追加されており，不要となったパスに属する命令の無効化やメモリ依存の検出などに用いられる．ここでパスについて**図 3** を用いて説明する．円は分岐を表し，白いものは両パス実行しなかった分岐を，黒いものは両パス実行した分岐である．矢印は，分岐間の制御フローを表す．両パス実行した分岐の次の基本ブロックから，次に両パス実行する分岐までをパスと呼ぶ．

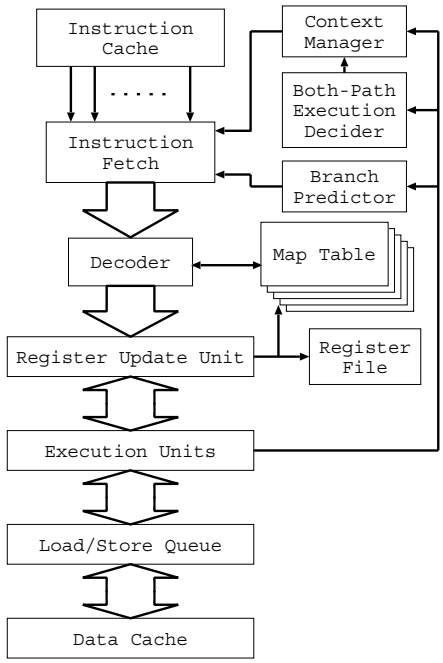


図 2 両パス実行アーキテクチャの構成
Fig. 2 Structure of both-path execution architecture.

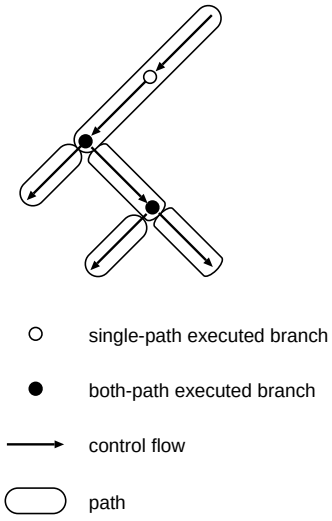


図 3 パスの例
Fig. 3 Example of paths.

3.1.1 コンテキスト・タグ

コンテキスト・タグとは，前述したように，パスを識別するタグである．すべてのパスについて，依存す

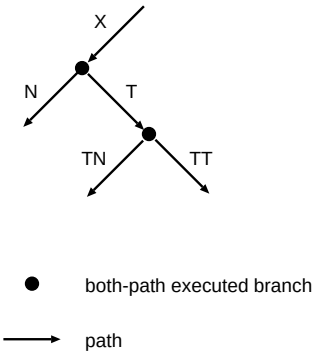


図 4 コンテキスト・タグの例
Fig. 4 Example of context tags.

る分岐がすべて解決するまで付加される．本論文で仮定するプロセッサでは，コンテキスト・タグを，T，N または null のいずれかのシンボル列で表する．両パス実行するある分岐 b を考える．今，その分岐が属するパスのコンテキスト・タグを C とする．その分岐に続く 2 つのパスのコンテキスト・タグは，Taken 方向のパスは C に T を，Not-Taken 方向のパスは N を連結したものである．これらの連結した各シンボルを，分岐 b に対応するシンボルと呼ぶ．両パス実行した分岐が解決したら，すべてのパスのコンテキスト・タグについて，その分岐に対応するシンボルを無効化し，null とする．すべてのシンボルが null となったコンテキスト・タグは空であり，そのパスはコンテキスト・タグを持たない状態となる．

図 4 に，図 3 に示したパスにコンテキスト・タグを付加した例を示す．図のように先行パスのコンテキスト・タグに T または N を連結したものをそれぞれコンテキスト・タグとする．図において，X はコンテキスト・タグが空であることを明に表している．

このエンコード法を用いれば，不要となったパスに属する命令を破棄する回路を単純に構成できる．**図 5** に context tag0 と context tag1 という 2 つのコンテキスト・タグを持つ 2 つのパスの間に子孫関係があるか（context tag0 が先祖で context tag1 が子孫）を判定する回路[8]を示す．図ではコンテキスト・タグが 6 ビットの例を示している．図中の valid と dir は，それぞれ有効ビットと方向ビットを表している．以下に不要となったパスに属する命令の破棄について説明する．不要となる命令は，解決された分岐命令の分岐しなかった方向に属するすべての命令である．したがって，解決された分岐のコンテキスト・タグに分岐しなかった方向のシンボルを連結したコンテキスト・タグを context tag0 とし，パイプライン中の命令のコンテキスト・タグを context tag1 として図 5 の回路を用いて子孫関係の判定を行う．その結果，子孫関係にあると判定された命令は不要なパスに属する命令であるので，破棄される．

3.1.2 コンテキスト管理機構

コンテキスト管理機構はフェッチするパスを決定する機構であり，コンテキスト数と同じエントリ数のテーブルで構成される．各エントリは，Valid，Context Tag，PC，Status の 4 つのフィールドからなる．Valid はそのエントリが有効かどうか示すフィールドである．Context Tag はそのエントリに割り当てられたパスのコンテキスト・タグを格納する．PC には当該パスの PC を格納する．Status にはフェッチするパスの決定に用いる当該パスの情報（フェッチすべき命令が存在するかどうか，キャッシュミスを起こしているかどうかなど）が格納される．

3.1.3 マップ 表

複数のレジスタ・コンテキストを保持するために次のような機構を用いる．まず，1 つの物理レジスタ・ファイルと各物理レジスタが各コンテキストの論理レジスタにどのように対応しているかを表すマップ表を用意する．物理レジスタ・ファイル（RUU とレジスタ・ファイル）には複数のコンテキストの内容が混在するが，マップ表をコンテキストごとに用意することにより，識別する．この構成では，単一パス実行の場合に比べ，コンテキスト数倍のマップ表が必要となる．また，性能の向上のためにはより多くの物理レジスタが必要となる．

ある分岐で両パス実行を開始する際，その分岐に続く 2 つのパスのそれぞれに対し，マップ表を新たに割り当て，現在のマップ表の内容をコピーする．これによりコンテキストを継承する．各パスの実行では，割り当てられたマップ表を独立に更新し，パスが破棄されるときは，対応するマップ表を解放する．性能向上のためには，マップ表のコピーは広いバンド幅が必要である．

3.2 両パス実行における問題

両パス実行の実現には以下の問題がある．

- パスごとにコンテキストを維持することによる複雑さの増加
- 複数のパスから命令をフェッチするため要求される命令フェッチバンド幅の増加
- フェッチする命令数の増加による命令キャッシュミスの増加
- 複数のパスからのメモリ参照により要求されるデータキャッシュバンド幅の増加

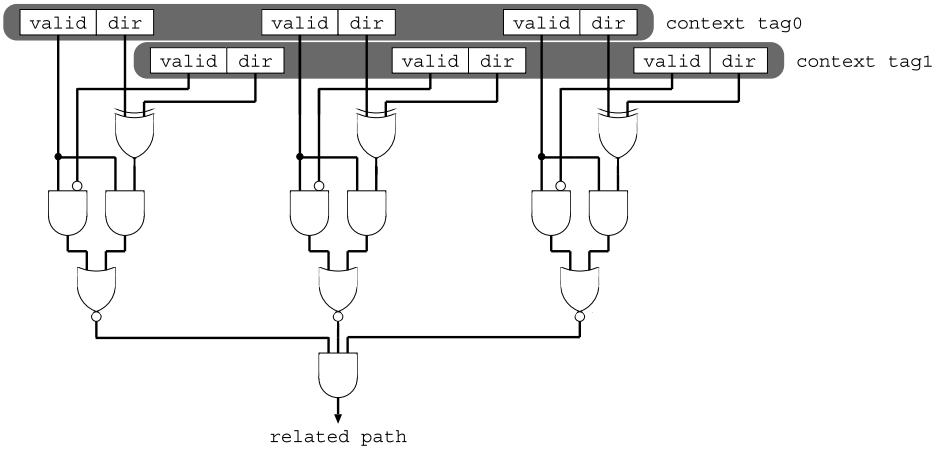


図 5 コンテキスト・タグ間の子孫関係の判定回路の例
Fig. 5 Example of a logic circuit to find offspring relation between two paths.

- 同時実行命令数の増加による要求される機能ユニット数の増加
- 同時実行命令数の増加による要求される物理レジスタ数の増加

以上のように，両パス実行には多くのハードウェア資源が必要なため，性能とのトレードオフを見極める正確な評価が必要である．特に，ハードウェアの複雑さを大きく左右するコンテキスト数に関する評価が必要である．

また，資源要求自体を低減させ，効率良く両パス実行を行うことが必要である．このためには，分岐予測の信頼性をできるだけ正確に推定し，両パス実行を行うべきかどうかを的確に判断する必要がある．

4. 分岐フィルタを付加した両パス実行判定機構

本章では，分岐フィルタを導入した両パス実行判定機構の提案を行い，その構成を示す．

4.1 分岐フィルタによる両パス実行判定精度の向上

分岐フィルタとは，分岐方向が強く偏っている分岐を，分岐予測器の PHT に登録しないことにより，PHT での競合による予測精度低下を防ぐものである．図 1 に示した両パス実行判定機構の判定精度も競合により低下するので，分岐フィルタは有効と考えられる．

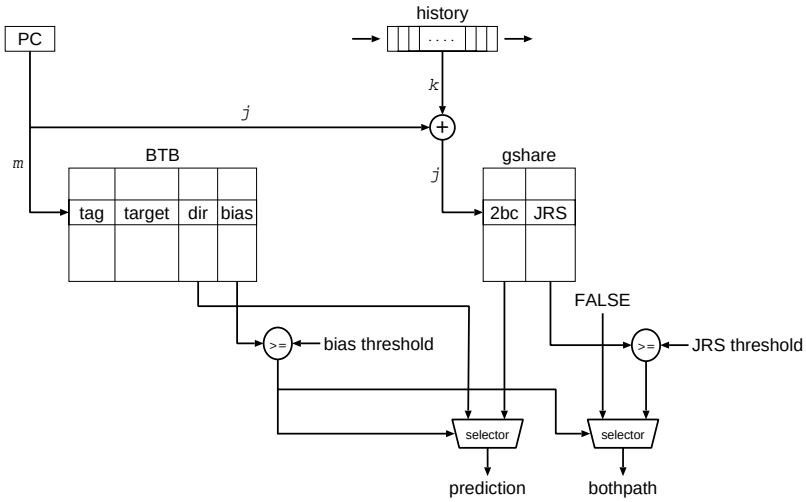
しかし分岐フィルタを用いることには，次のような欠点がある．分岐フィルタで除かれる分岐は非常に偏ってはいるが，偏りの方向と異なる方向にまったく分岐しないわけではない．現在の分岐予測は非常に高い精度を達成しており，このような稀な場合も正しく予測が行えるようになっている．分岐フィルタを使用しない従来の両パス実行では，PHT のエントリの JRS を判断に使用するので，1 つの分岐に対しても履歴に応じて，両パス実行すべきかどうかを決定する．これに対し，分岐フィルタで判断すると，このような細かい判断ができなくなるという問題がある．

4.2 両パス実行判定の変更

分岐フィルタを用いた両パス実行判定機構の構成を**図 6** に示す．BTB（Branch Target Buffer）の各エントリに対応する分岐の分岐方向の偏りを検出するカウンタ bias と静的分岐予測を格納する dir というフィールドを追加する．

まず，予測時の動作を以下に示す．bias があらかじめ定められた閾値（図では bias threshold）以上であれば，分岐フィルタによる予測 dir を分岐予測結果として採用し，gshare から得られる JRS による両パス実行の判断にかかわらず，両パス実行は行わない．一方，bias が bias threshold を下回った場合，gshare による予測が分岐予測結果として採用され，gshare の JRS による両パス実行の判断が採用される．両パス実行しないと判断された場合，予測に従って単一パス実行を行う．両パス実行すると判断された場合は，両パス実行を開始する．分岐予測結果は必要なら，両パス実行効率向上のために利用する（後述）．

次に，更新時の動作を以下に示す．分岐の結果が判明し，その分岐に対応するエントリが BTB にない場合，その分岐を BTB に登録する．その際，最後の実行結果である分岐方向を dir に記録し，静的予測とする．BTB に対応するエントリがあった場合，そのエントリの dir と実行結果を比較する．一致していれば bias を 1 増加させ，そうでなければ dir を判明した方向にセットし，さらに bias を 0 にリセットする．また，bias が bias threshold 以上であれば，gshare の



2bc : 2-bit counter
conf : confidence counter
target : target address
dir : static prediction
bias : hit counter

図 6 分岐フィルタ用いた両パス実行判定機構の構成
Fig. 6 Organization for our both-path execution decision mechanism with branch filtering.

表 1 ベンチマーク・プログラム
Table 1 Benchmark programs.

プログラム	入力	静的分岐数	動的分岐数の 90%を占める静的分岐数	動的分岐数	動的命令数	分岐頻度 [%]
compress95	30000 e 2231	527	31	10.5M	94.7M	11.1
gcc	genoutput.i	16,560	3,657	13.0M	84.2M	15.4
go	6 9 2stone9.in	6,039	1,175	9.2M	75.3M	12.2
ijpeg	specmun.ppm	1,496	50	23.9M	450.0M	5.3
li	train.lsp	677	59	24.2M	183.3M	13.2
m88ksim	ctl.in	1,062	89	12.6M	100.3M	12.6
perl	scrabbl.in	2,134	175	10.4M	80.3M	13.0
vortex	vortex.in	1,428	389	8.8M	80.0M	11.0

PHT の更新は行わない．

5. 評価結果

本章では，最初に評価環境について述べる．次に，両パス実行の問題点の重要性と評価項目について述べ，両パス実行の性能および両パス実行と分岐フィルタを組み合わせた両パス実行の性能を評価する．

5.1 評価環境

SimpleScalar Tool Set [12] Version 3.0a の中のスーパスカラ・プロセッサ用シミュレータをもとに，両パス実行を行うシミュレータを作成し，測定を行った．命令セットは MIPS R10000 [13] を拡張した SimpleScalar/PISA である．ベンチマーク・プログラムは，SPECint95 の全 8 種類を使用した．ベンチマーク・プログラムのバイナリは，GNU GCC Version 2.7.2.3（コンパイルオプション：-O6 -funroll-loops）を用いて作成した．

表 1 に入力セット，静的分岐数（コミットされたもののみ），動的分岐数の 90%を占める静的分岐数，動的分岐数，動的命令数，分岐出現頻度を示す．シミュレーション時間が過大にならないようにするために，関数の出現頻度をほぼ維持しつつ，入力のパラメータを調整している．また，ijpeg では最初の 50M 命令をスキップした後，450M 命令を実行し，vortex では最初の 100M 命令をスキップした後，80M 命令を実行した．ijpeg，vortex 以外のベンチマーク・プログラムでは，命令のスキップを行わず，最後まで実行した．なお，以後のグラフでのベンチマーク・プログラム名

表 2 プロセッサモデル
Table 2 Processor model.

命令デコード幅	最大 8 命令
命令発行幅	最大 8 命令
命令コミット幅	最大 8 命令
命令ウィンドウ	RUU (Register Update Unit) 128 エントリ，LSQ (Load/Store Queue) 64 エントリ
機能ユニット	iALU 8，iMULT/DIV 4，Ld/St 4，fpALU 4，fpMULT/DIV/SQRT 4
分岐予測機構	BTB 2048 エントリ 4 ウェイセットアソシアティブ，gshare 9 ビット履歴 16K エントリ PHT，RAS (Return Address Stack) (各パス) 16 エントリ，分岐予測ミスペナルティ 10 サイクル
コンテキスト数	5 パス
命令キャッシュ	128 KB 2 ウェイ・セットアソシアティブ，ライン幅 32 バイト，2 ポート，各ポート最大 4 命令，ヒットレイテンシ 1 サイクル，ミスペナルティ 10 サイクル
データキャッシュ	128 KB 2 ウェイ・セットアソシアティブ，ライン幅 32 バイト，4 ポート，ヒットレイテンシ 2 サイクル，ミスペナルティ 10 サイクル
二次キャッシュ	統合，2 MB 8 ウェイ・セットアソシアティブ，ライン幅 64 バイト，ミスペナルティ 32 サイクル

の表記において，compress95，m88ksim，vortex は，それぞれ comp，m88k，vort と表すこととする．

5.2 評価項目とプロセッサの構成

3.2 項で両パス実行の実現における問題を述べたが，そのうち，特に重要と思われる以下の項目について評価を行う．

- コンテキスト数が両パス実行性能に与える影響
- 命令キャッシュポート数が両パス実行性能に与える影響
- 両パス実行が命令キャッシュミス率に与える影響
- 分岐フィルタによる両パス実行判定精度の改善

評価においては，**表 2** に示すプロセッサを仮定し，評価項目となるパラメータのみをふって測定した．プロセッサの各パラメータは，現在のハイエンド・スーパスカラ・プロセッサと同程度から，近い将来に実現可能と考えられる程度とした．評価項目であるコンテキスト数，命令キャッシュポート数，命令キャッシュサイズは，後の測定で分かるコスト性能比の良い値としている．また，命令キャッシュポートのパスへの割当てについては，1 つのポートは分岐予測により予測されたパスのフェッチに与え，もう 1 つのポートは，それ以外のパスのフェッチにラウンドロビンで与えるとした．また，命令キャッシュポート数がコンテキスト数を上回った場合には，1 つのパスに対して複数の命令キャッシュポートが割り当てられ，Taken 分岐を越えて命令をフェッチできるとした．

また，不要なパスの命令の破棄は SEE[8] と同じく，分岐方向が判明すると，その方向とは逆の方向に属するすべての命令が無効化されるとした．

これらの条件の下で単一パス実行での各ベンチマーク・プログラムにおける分岐予測精度と IPC と分岐がフェッチされてから解決されるまでの平均サイクル数を**表 3** に示す．

表 3 単一パスでの分岐予測精度と IPC
Table 3 Prediction accuracy and IPC on single-path execution.

プログラム	分岐予測精度 [%]	IPC	分岐が解決されるまでの平均サイクル数
compress95	91.30	2.18	6.24
gcc	90.03	1.58	6.25
go	81.03	1.38	8.47
ijpeg	90.21	2.86	6.13
li	93.37	1.90	6.13
m88ksim	98.14	2.77	9.28
perl	95.38	2.00	6.33
vortex	97.77	2.77	6.95

以下では，まず，表 2 に示す構成での結果を示し，その後，評価項目としたパラメータをふった評価結果を示す．

5.3 両パス実行の性能

図 7 に単一パス実行に対する両パス実行による性能向上率を示す．横軸はベンチマーク・プログラムで，縦軸は性能向上率である．各ベンチマーク・プログラムにつき，2 本の棒グラフがある．左の棒グラフは分岐フィルタのない従来の両パス実行方式（JRS）であり，右の棒グラフは両パス実行と分岐フィルタを組み合わせた方式（sfilter+JRS）である．なお，bias threshold は 15 とし，JRS threshold は 7 とした．これらの値は本研究の測定によって求めた最適な値である

JRS を見ると，すべてのプログラムで両パス実行によって性能が向上していることが分かる．特に go における性能向上が 20.5%と非常に大きい．go では分岐予測精度が 81.03%と特に低いために両パス実行が分岐予測ミスを回避するように効果的に作用したためである．次に compress95，gcc，li の性能向上が大きい．これは比較的分岐予測精度が低いためである．ijpeg も分岐予測精度は高くないが，表 1 を見れば分かるように，分岐の出現頻度が少なく，改善の機会が

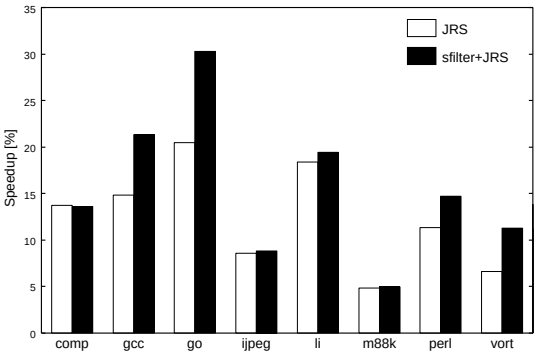


図 7 分岐フィルタと両パス実行による性能向上率
Fig. 7 Speedup with both-path execution and branch filtering.

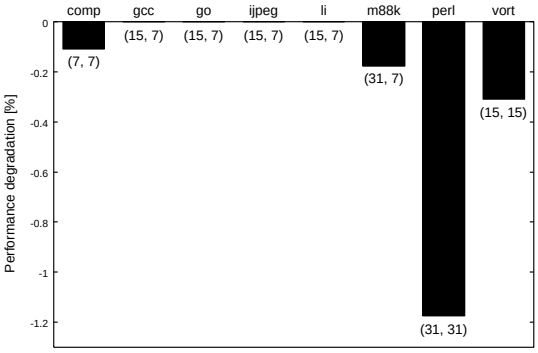


図 8 ベンチマークごとの最適パラメータからの性能低下率
Fig. 8 Performance degradation from optimum parameters case.

表 4 両パス実行判定精度の改善
Table 4 Decision accuracy improvements for both-path execution.

プログラム	改善量 [%point]
compress95	0.14
gcc	1.07
go	1.05
ijpeg	3.13
li	1.73
m88ksim	0.49
perl	3.86
vortex	7.48

少ないためである．また m88ksim，vortex でも性能向上が高くない．これは go とは逆に分岐予測精度が非常に高く，改善の機会が少ないからである．

次に，JRS と sfilter+JRS を比べてみると，ベンチマークによっては性能向上を達成しているものがあるが，まったく性能向上していないものもある．これは，両パス実行判定精度，分岐予測精度，分岐出現頻度，複数パスを実行するために必要な資源の空き状況の複合的要因によるものと思われる．**表 4** に sfilter による判定精度の改善量を示す．ここで判定精度とは，両パス実行すると判定した分岐の中で，予測ミスした分岐（コミットされたもののみ）の割合である．

まず，go と gcc に関しては，判定精度向上，低精度な分岐予測，高い分岐出現頻度，低 IPC から推測される資源競合の低さという要素が加わり，高い性能向上を得ている．perl と vortex は，分岐予測精度が高いが，判定精度が非常に大きく改善しており，性能向上を得ることができたと思われる．これに対して，compress95 は，両パス実行判定精度の改善が非常に少ないので，性能向上も見られない．m88ksim は，分岐予測精度が非常に高いにもかかわらず，判定精度の向上が少いため，性能向上がほとんどなかったと思われる．ijpeg は，判定精度が大きく改善しているが，前述のように分岐の出現頻度が低く，また IPC が高いことから複数のパスを実行する資源が十分になく，性能向上が得られなかったものと思われる．li は，判定精度は向上しているが，分岐予測精度が比較的高く，判定精度向上が十分でなかったため，性能向上が得られなかったと思われる．

今回の測定で得られた性能向上率が，理想の両パス実行が行われた場合に対し，どの程度の達成度であるかを知ることは，両パス実行機構を今後改良するにあたっての目標となる．理想の両パス実行とは，分岐予測ミスする分岐についてのみ両パス実行することをいう．我々のシミュレータで現在は，これを実現することはできないが，文献 9) にその場合の測定結果が示されている．これによると，単一パス実行に対して平均 36%まで性能向上の余地があることが示されている．

次に，性能が bias threshold と JRS threshold の 2 つのパラメータによって，どの程度影響を受けるかについて考察する．**図 8** は，各ベンチマーク・プログラムについて最適なパラメータの場合に対し，ベンチマーク・プログラムの平均で最適なパラメータ（それぞれ 15，7）の場合，どの程度性能が低下するかを測定した結果である．なお，bias threshold，JRS threshold の 2 つのパラメータは，それぞれ 1，3，7，15，31 とふって測定を行った．横軸はベンチマーク・プログラムで，縦軸は各ベンチマーク・プログラムごとの最適パラメータからの性能低下率である．棒グラフの先端につけた数字の組 (b, J) は，それぞれ bias threshold，JRS threshold の最適値である．図より perl において最大 1.2%性能が低下し，他のベンチマーク・プログラムでは 0〜0.3%の性能低下にとどまっている．ここで最も性能低下の大きかった perl に対して，2 つのパ

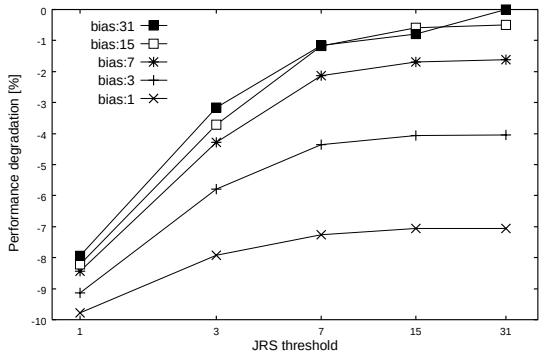


図 9 最適パラメータからの性能低下率（perl）
Fig. 9 Performance degradation from optimum parameters case (perl).

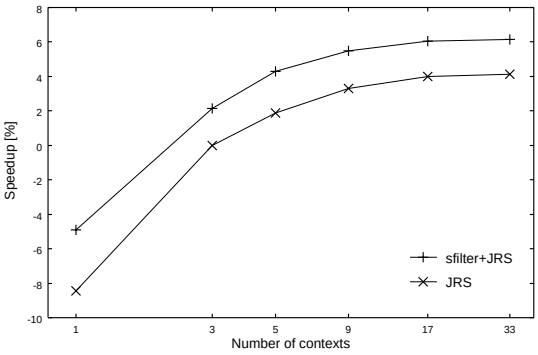


図 11 コンテキスト数を変化させた場合の性能向上率（平均）
Fig. 11 Speedup with the different number of contexts (mean).

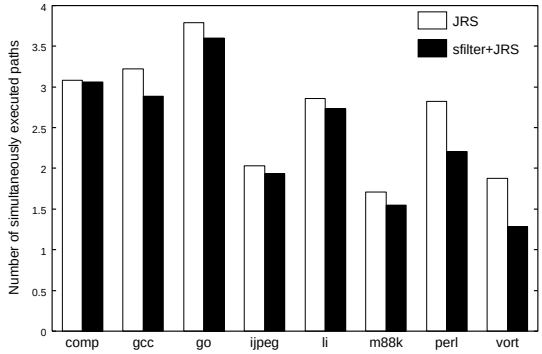


図 10 同時実行された平均パス数
Fig. 10 Number of simultaneously executed paths.

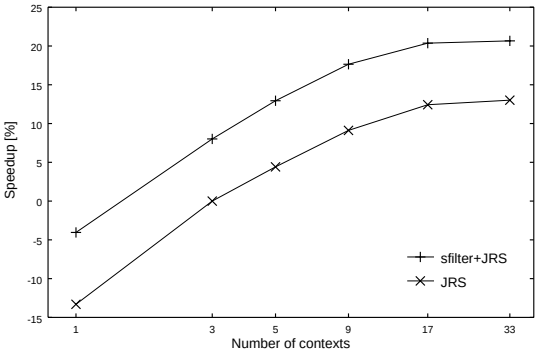


図 12 コンテキスト数を変化させた場合の性能向上率（go）
Fig. 12 Speedup with the different number of contexts (go).

ラメータを変化させたときの性能低下を**図 9** に示す．グラフの横軸は JRS threshold で，縦軸は性能低下率である．グラフには折れ線グラフが 5 本あり，それぞれ bias threshold が 1，3，7，15，31 を表している．図より bias threshold，JRS threshold が 7 以上ならば，どちらの値についても性能が敏感に変化することはないことが分かる．他のベンチマーク・プログラムについても同様の傾向がある．表 1，表 3 から分かるように，SPECint95 には性質の大きく異なるプログラムが含まれている．このことと，図 8，図 9 の結果より，これらのパラメータがそのアプリケーションでの最適値から大きく外れても，性能に大きな影響は与えないと考えられる．

次に，**図 10** に実際に同時実行されたパス数の平均を示す．横軸はベンチマーク・プログラムで，縦軸は同時実行されたパス数の平均である．各ベンチマーク・プログラムにつき，2 本の棒グラフがある．左の棒グラフは JRS であり，右の棒グラフは sfilter+JRS である．JRS を見ると，分岐予測精度の低い go では多くのパスが同時実行され，分岐予測精度の高い m88ksim や vortex ではあまりパスが同時実行されていないことが分かる．JRS と sfilter+JRS を比較すると，すべてのベンチマーク・プログラムで同時実行されたパス数が減少している．これは分岐フィルタによって両パス実行判定精度が向上したためである．

5.4 コンテキスト数制限による影響

図 11 に，コンテキスト数を 1〜33 と変化させたときのベンチマーク平均での性能向上率を示す．基準は，コンテキスト数が 3 の場合の性能である．図 3 を見れば分かるように，両パス実行を許す分岐数 (b) と，そのときのパスの最大数 (p) は，$p = 2b + 1$ という関係があるので，3 コンテキストとは両パス実行を許す分岐数が 1 の場合であり，最も単純な両パス実行方式である．また，**図 12** は，分岐予測精度の低いプログラムの代表として，go の場合の性能向上率を示している．分岐予測精度の高いプログラムは，平均と似た傾向を示すので，ここでは示さない．

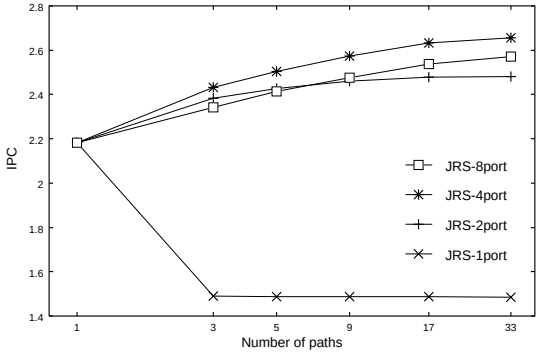


図 13 命令キャッシュポート数とコンテキスト数を変化させた場合の IPC（従来の両パス実行方式）
Fig. 13 Impact on IPC with the number of contexts and the number of L1 I-cache ports (conventional both-path execution).

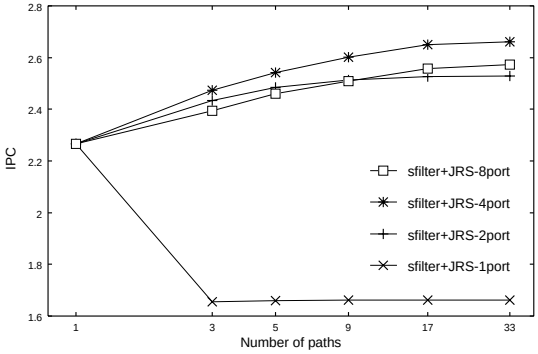


図 14 命令キャッシュポート数とコンテキスト数を変化させた場合の IPC（両パス実行と分岐フィルタを導入した方式）
Fig. 14 Impact on IPC with the number of contexts and the number of L1 I-cache ports (both-path execution with branch filtering).

図 11 より，コンテキスト数 3〜5 で性能向上は抑えられており，この程度で十分であることが分かる．一方，図 12 からは，9〜17 程度まで性能は飽和せず増加し続ける（図 11 と図 12 は縦軸のスケールが異なることに注意）．go のようにプログラムサイズが大きく分岐予測が困難なプログラムは，SPECint95 には多くは含まれていないため，平均では 3〜5 コンテキストで十分であるという測定結果を得たが，一般には，そのようなプログラムは数多く存在することが知られている．たとえば，文献 14) では，HTTP daemon, Ghostscript，mpeg，Verilog などが例としてあげられおり，SPECint95 と実使用での性能差について問題が投げかけられている．このようなプログラムの存在とマップ表の複雑さを考慮すると，最善のコスト性能比でのコンテキスト数は，SPECint95 での最適値 3〜5 より，やや多い数と思われる．

5.5 命令キャッシュポート数とコンテキスト数による影響

命令キャッシュポート数とコンテキスト数を変化させた場合の IPC を，**図 13** と**図 14** に示す．図 13 は従来の両パス実行の場合であり，図 14 は分岐フィルタを導入した場合である．横軸はコンテキスト数であり，1〜33 で変化させた．縦軸は IPC である．それぞれのグラフには 4 本の折れ線があり，命令キャッシュポート数を 1（1port），2（2port），4（4port），8（8port）と変えた場合である．

これらの図から分かるように，JRS，sfilter+JRS ともにポート数が 1 の場合には，コンテキスト数を増やしても，単一パス実行の性能より低下している．これは，命令をフェッチできるパスが 1 パスだけのため，コンテキスト数を増やすと，将来コミットされる正しいパスの命令をフェッチする機会が減少し，有効な命令フェッチバンド幅が低下するためである．また，JRS，sfilter+JRS ともに，ポート数 4 の場合が最も性能が高い．ポート数を 8 にすると性能が低下する．これは，ポート数の増加のために，コミットされない誤ったパスからの命令がプロセッサに供給され，資源競合が起きるためである．

表 5 命令キャッシュミス数の増加率
Table 5 Increase of I-cache misses.

命令キャッシュサイズ [KB]	gshare	JRS	sfilter+JRS
4	1029.82	1132.53	1100.16
8	329.92	353.81	328.46
16	80.386	85.672	76.059
32	13.771	14.919	13.008
64	1.848	2.100	1.855
128	1.00	1.098	1.016
256	0.710	0.751	0.741
512	0.647	0.685	0.678

5.6 命令キャッシュに対する影響

両パス実行が命令キャッシュに与える影響を見るために，命令キャッシュサイズを変化させて性能を調べた．

表 5 に命令キャッシュサイズを変化させた場合のミス数の変化を示す．128 K バイトの単一パス実行でのミス数で正規化している．すべての命令キャッシュサイズにおいて，単一パスより両パス実行によりミス数が増加していることが分かる．JRS と sfilter+JRS を比べると，sfilter+JRS の方がミス数が減少していることも分かる．

上記命令キャッシュミス数の違いが，性能に与える影響を調べた．**図 15** がその結果である．縦軸は，完全キャッシュの場合に対する各命令キャッシュサイズで

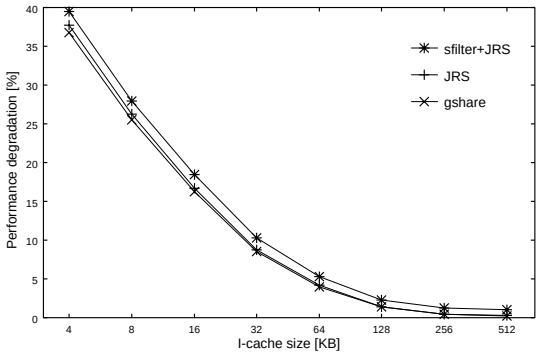


図 15 命令キャッシュサイズを変化させた場合の性能低下率
Fig. 15 Performance degradation with differnct I-cache sizes.

の性能の低下率である．すべてのベンチマーク・プログラムで，この図と同じ傾向を示しており，違いは小さい．同図より分かるように，両パス実行が命令キャッシュに与える影響は単一パス実行のそれとほとんど変わらないといえる．つまり，表 5 に示したような違いはあるが，性能に顕著な違いを与えるほどではないことが分かる．

6. 両パス実行におけるハードウェア量

本章では，両パス実行において必要となるハードウェア量について述べる．

6.1 マップ 表

両パス実行により大きな性能向上を得るためには，マップ表コピーのレイテンシは非常に短くなければならない．可能なら 1 サイクルで行うことが望ましい．このために要求されるマップ表のバンド幅は非常に大きく，特別な回路が必要とされる．文献 15) に，このような問題に対応する回路が示されている．この回路は，各マップ表の対応するビットを格納するセルを集めてレイアウトし，隣接するセルを結合するものである．**図 16** に 5 つのコンテキストを保持する場合のマップ表のセルの構成例を示す．(a) はセルの間に完全網を形成した場合で，(b) はバス結合した場合である．各配線は 1 ビット幅である．(a) は同時に複数のマップがコピーできるが，(b) では同時に 1 つのマップしかコピーできない．扱うコンテキスト数が少ない場合，(a) の回路を用いることができ，単純な回路で，最大のバンド幅を確保できる．5 章で行った評価より，必要なコンテキスト数は 3〜5 であるから，(a) の回路を用いることができると思われる．また，コンテキスト数 5 のときのマップ表のハードウェア量はわずか 350 バイト程度であり，十分に小さい．

図 16 の回路を使ってもマップ表の全面積は大きくなる．このため，単一パス実行のスーパスカラ・プロセッサのマップ表に比べれば，アクセス時間が増加し，性能とのトレードオフが存在する．しかし，SRAM の高速化に一般に有効な回路や構成の工夫（たとえば，文献 16) にあるようなアレイの分割や文献 17) にあるようなワード線の階層化）や，パイプライン化などのアーキテクチャ上の工夫を行えば，アクセス時間の増加が性能に与える影響を十分に小さくできると思われる．

6.2 キャッシュバンド幅

5.5 節の結果より，命令キャッシュのポート数は 2 で十分であることが分かった．また，今回の測定では 8 命令発行のプロセッサに対し，データ・キャッシュのポート数を 4 と仮定しており，十分なバンド幅であると思われる．キャッシュの多ポート化には，空間的多重化（たとえば Compaq Alpha 21164[18]），時間的多重化（たとえば Compaq Alpha 21264[19]），インターリーブ（たとえば AMD Athlon[20]）など，さまざまな方法がある．2〜4 ポートは現在の商用マイクロプロセッサと同程度なので，ハードウェア量，複雑さともに問題ないといえる．

6.3 機能ユニット数

両パス実行では同時実行命令数が増加するために要求される機能ユニット数が増加する．本研究では，たとえば整数 ALU が 8 であるなど，現在のプロセッサの 2 倍程度を仮定している．しかし，1 億程度ものトランジスタを搭載する将来のプロセッサでは，機能ユニットのこの程度の増加が，チップ面積を増加させる割合は小さい．また，機能ユニットを増やすとデータパスの複雑さが増すが，複雑さの低減に関しては研究がなされており（たとえば文献 21)），本研究で仮定している程度では問題にならないと思われる．

6.4 物理レジスタ数

両パス実行では同時実行命令数が増加するために要求される物理レジスタ数が増加する．本研究では物理レジスタ数を 185 本（RUU 128 本，int 32 本，fp 32 本，その他 3 本）と仮定している．これは現在の商用マイクロプロセッサより 22%多い（たとえば Compaq Alpha 21264[19]では 152 本（int 80 本，fp 72 本）である）．この増加により，レジスタの全面積が大きくなり，アクセス時間が大きくなる．しかし，文献 22) にあるようなレジスタ・ファイルのマルチバンク化や文献 23) にあるような物理レジスタへの論理レジスタの割当て手法の改良により，アクセス時間の増加が性能に与える影響を十分に小さくできると考えられる．

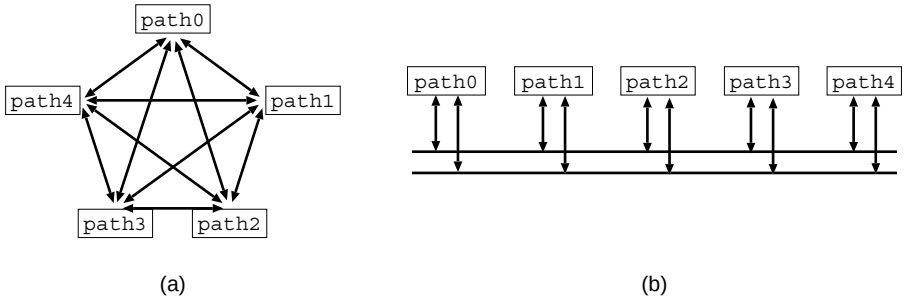


図 16 マップ表のセルの構成例
Fig. 16 Organization example for a cell of the map table.

7. ま と め

両パス実行は分岐予測ミスペナルティを原理的になくすことができ，性能向上が期待できる手法である．しかし，近い将来に実現可能と思われるハードウェア量における評価が不十分であり，どの程度有用であるかの知見は十分には得られていない．本論文では，ハードウェアの複雑さの大きな要因となると考えられるコンテキスト数やキャッシュポート数などの影響について SPECint95 をベンチマークとして評価を行った．コスト性能比の点から，命令キャッシュのポート数は 2，パス数は 3〜5 が適当であることが分かった．ただし，サイズが大きなプログラムにおいては，9〜17 パスまで性能は向上し，より多くのパスを扱うことによる性能改善の余地は大きいことが分かった．また，両パス実行は，単一パス実行に比べて多くの命令をフェッチするので，命令キャッシュの性能への悪影響が危惧されたが，それはほとんどないことが分かった．さらに，この構成パラメータにおいて，ハードウェアの複雑さの増加は大きくないことを議論し，レジスタ・リネーミングのためのマップ表へ追加すべきハードウェア量は 350 バイト程度であることを見積もった．クロック・サイクル時間への影響を軽微と仮定し，実行サイクル数の評価を行った結果，近い将来実現可能と考えられる 8 命令発行，2 ポートの命令キャッシュ，4 ポートのデータキャッシュ，5 つのコンテキスト数のプロセッサにおいて，両パス実行は単一パス実行に対し，最大 20.5%，平均 11.2%の性能向上を見込めることが分かった．また，両パス実行の判定に使用する表における競合を抑制するために，分岐フィルタを導入する機構を提案した．これにより，従来の判定機構の場合に比べ，最大 8.1%の性能向上が得られることを確認した．

参 考 文 献

1) 野口良太，森 敦司，小林良太郎，安藤秀樹，島田俊夫：分岐方向の偏りを利用し破壊的競合を低減する分岐予測機構，情報処理学会論文誌，Vol.40, No.5, pp.2119–2131 (1999).
2) McFarling, S.: Combining Branch Predictors, WRL Technical Note TN-36, Digital Equipment Corporation (1993).
3) Yeh, T.-Y. and Patt, Y.: Two-Level Adaptive Branch Prediction, *Proc. 24th Ann. Int'l Symp. and Workshop on Microarchitecture*, pp.55–61 (1991).
4) Yeh, T.-Y. and Patt, Y.: A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History, *Proc. 20th Int'l Symp. on Computer Architecture*, pp.257–266 (1993).
5) Uht, A.K. and Sindagi, V.: Disjoint Eager Execution: An Optimal Form of Speculative Execution, *Proc. 28th Int'l Symp. on Microarchitecture*, pp.313–325 (1995).
6) Heil, T. and Smith, J.E.: Selective Dual Path Execution, Technical Report, University of Wisconsin-Madison (1996). http://www.ece.wisc.edu/~jes/papers/sdpe.ps.
7) Farrens, M., Heil, T., Smith, J. E. and Tyson, G.: Restricted Dual Path Execution, Technical Report CSE-97-18, University of California, Davis (1997).
8) Klauser, A., Paithankar, A. and Grunwald, D.: Selective Eager Execution on the PolyPath Architecture, *Proc. 25th Ann. Int'l Symp. on Computer Architecture*, pp.250–259 (1998).
9) Klauser, A. and Grunwald, D.: Instruction Fetch Mechanisms for Multipath Execution Processors, *Proc. 32nd Int'l Symp. on Microarchitecture*, pp.38–47 (1999).
10) Chang, P.-Y., Evers, M. and Patt, Y.N.: Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference, *Proc. Int'l Conf. on Parallel Architectures and Com-*

pilation Techniques, pp.48–57 (1996).
11) Wallace, S., Calder, B. and Tullsen, D.M.: Threaded Multiple Path Execution, *Proc. 25th Ann. Int'l Symp. on Computer Architecture*, pp.238–249 (1998).
12) Burger, D. and Austin, T.M.: The SimpleScalar Tool Set, Version 2.0, Technical Report CS-TR-97-1342, University of Wisconsin-Madison (1997).
13) MIPS Technologies, Inc.: MIPS R10000 Processor User's Manual, Version 2 (1996).
14) Gloy, N., Young, C., Chen, J.B. and Smith, M.D.: An Analysis of Dynamic Branch Prediction Schemes on System Workloads, *Proc. 23rd Int'l Symp. on Computer Architecture*, pp.12–21 (1996).
15) Hwu, W.W. and Patt, Y.N.: Checkpoint Repair for Out-of-order Execution Machines, *Proc. 14th Ann. Int'l. Symp. on Computer Architecture*, pp.18–26 (1987).
16) Wada, T., Rajan, S. and Przybylski, S.A.: An Analytical Access Time Model for On-Chip Cache Memories, *IEEE Journal of Solid-State Circuits*, Vol.27, No.8, pp.1147–1156 (1992).
17) Yoshimoto, M., Anai, K., Shinohara, H., Yoshihara, T., Takagi, H., Nagao, S., Kayano, S. and Nakano, T.: A 64kb Full CMOS RAM with Divided Wordline Structure, *Int'l Solid-State Circuits Conference Digest of Technical Papers*, pp.58–59 (1983).
18) Digital Equipment Corporation: *Alpha 21164 Microprocessor Hardware Reference Manual* (1995).
19) Gwennap, L.: Digital 21264 Sets New Standard, *Microprocessor Report*, Vol.10, No.14, pp.11–16 (1996).
20) Advanced Micro Devices, Inc.: AMD Athlon Processor Technical Brief (1999).
21) Palacharla, S., Jouppi, N.P. and Smith, J.E.: Complexity-Effective Superscalar Processors, *Proc. 24th Int'l Symp. on Computer Architecture*, pp.206–218 (1997).
22) Cruz, J.-L., González, A., Valero, M. and Topham, N.P.: Multiple-Banked Register File Architectures, *Proc. 27th Int'l Symp. on Computer Architecture*, pp.316–325 (2000).
23) Monreal, T., González, A., Valero, M., González, J. and Viñals, V.: Delaying Physical Register Allocation Through Virtual-Physical Registers, *Proc. 32nd Int'l Symp. on Microarchitecture*, pp.186–192 (1999).

(平成 13 年 2 月 8 日受付)
(平成 13 年 5 月 14 日採録)

片山 清和（正会員）

1994 年名古屋大学工学部情報工学科卒業．1996 年名古屋大学大学院工学研究科情報工学専攻博士課程前期課程修了．1999 年名古屋大学大学院工学研究科情報工学専攻博士課程後期課程満了．現在，名古屋大学大学院工学研究科在学中．計算機アーキテクチャの研究に従事．

安藤 秀樹（正会員）

1959 年生．1981 年大阪大学工学部電子工学科卒業．1983 年大阪大学大学院修士課程修了．京都大学工学博士．1983 年三菱電機（株）LSI 研究所．ISDN 用ディジタル信号処理 LSI，第 5 世代コンピュータ・プロジェクトの推論マシン用プロセッサの設計に従事．1991 年 Stanford 大学客員研究員．1997 年名古屋大学大学院工学研究科電子情報学専攻講師．1998 年同大学助教授．1998 年東京大学大学院理学系研究科助教授併任．1998 年情報処理学会論文賞受賞．計算機アーキテクチャ，コンパイラの研究に従事．

島田 俊夫（正会員）

1968 年東京大学工学部計数工学科卒業．1970 年東京大学大学院修士課程修了．同年電子技術総合研究所入所．1993 年より名古屋大学大学院工学研究科電子情報学専攻教授．人工知能向き言語，LISP マシン，データフロー計算機の研究に従事．最近はマイクロプロセッサのアーキテクチャやチップ内並列処理の研究を行っている．1988 年度市村賞，1994 年度情報処理学会論文賞，1995 年度注目発明賞受賞．工学博士．