社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－ARC－165 (15)
2005／12／1

統合型トレースキャッシュにおける分岐予測器のシミュレーション評価

谷川 一哉[†] 斎藤 正嗣[†] 前田 志[††] 弘中 哲夫[†]
小出 哲士[†††] マタウシュハンスユルゲン[†††]

E-mail: {tanigawa,hironaka}@ce.hiroshima-cu.ac.jp

[†]広島市立大学情報科学部 [††]広島市立大学大学院情報科学研究科
[†††]広島大学ナノデバイス・システム研究センター

我々はトレースキャッシュと命令キャッシュを統合することにより，キャッシュ容量を効率的に使用する統合型トレースキャッシュを提案している．統合型トレースキャッシュでは，トレースデータをマルチバンクメモリに効率的に格納する手法を導入しており，本稿ではその手法とマルチ分岐予測機構を効率的に組み合わせる手法について提案する．本稿ではマルチ分岐予測機構として TMP を使用し，シミュレーション評価により，マルチ分岐予測機構を 16 命令発行プロセッサに導入した場合の分岐予測ヒット率と IPC を評価した．その結果，分岐予測ヒット率は 89.6%～99.8%と通常の単一の分岐予測並に高いヒット率を達成した．IPC は 1.82～4.92 という結果になった．

Evaluation of Branch Predictor for Unified Instruction Trace Cache

Kazuya Tanigawa[†] Tadashi Saito[†] Moto Maeda[††] Tetsuo Hironaka[†]
Tetsushi Koide[†††] Hans Jurgen Mattausch[†††]

[†] Faculty of Information Sciences, Hiroshima City University
[††] Graduate School of Information Sciences, Hiroshima City University
[†††]Research Center for Nanodevices and Systems, Hiroshima University

We proposed a unified trace cache that enables to unify trace cache and instruction cache. The unified trace cache includes in a multi bank memory to store instructions and trace data effectively. In this paper, we propose a method to combine a multi branch prediction and the multi bank memory. In this implementation, we use *Tree Based Multi Branch Prediction* as a multi branch prediction. We evaluates its predition hit ratio and nstructions per cycle (IPC) on the superscalar processor with 16 instruction issues by using SPECint95 benchmark programs. From the results, the branch prediction method achieves the prediction hit ratio from 89.6% to 99.8%, that is a comparable result to a single branch prediction method. The IPC achieved by the presented method was 1.82 to 4.92.

1 はじめに

スーパスカラプロセッサでは，複数命令を同時に実行するため高い命令フェッチバンド幅を提供できるキャッシュが必要となる．しかし、命令フェッチバンド幅を増加させるためには分岐命令による命令列の分断を解決することが必要となる．このために，現在トレースキャッシュによる命令フェッチ機構が提案されており，本研究室ではそのトレースキャッシュと命令キャッシュの両方を効率的に使用する統合型トレースキャッシュを提案している [1].

統合型トレースキャッシュでは命令とトレースデータを効率的に格納・利用する手法としてプログラムを固定長のブロック単位で管理することでマルチバンクメモリを利用したキャッシュシステムを実現している．しかしトレースデータは過去 1 回分の分岐履歴結果を元に前回と同じ分岐方向に分岐すると予測した結果であるので，トレースデータからフェッチは性能の悪い分岐予測器を利用しているのと等価であるといえる．

また，命令のフェッチに関しては，論文 [1] では 100% ヒットする分岐予測を利用している点で十分な評価であるとはいえず，また分岐予測機構の拡張に関して十分な検討がなされていない．そこで，本稿では本格的なマルチ分岐予測器を導入することにより，さらならフェッチバンド幅の向上を実現し，プロセッサの性能向上を目指す．

本稿の構成を以下に示す．まず 2 節において，統合型トレースキャッシュの構造について説明する．次に 3 節において，マルチ分岐予測機構として TMP を導入し，それをトレースキャッシュに組み込む手法について説明する．4 節において SPECint95 ベンチマークテストを用いて性能評価を行う．5 節でまとめる．

2 統合型トレースキャッシュ

本節では本研究室で提案している統合型トレースキャッシュについて述べる．

2.1 統合型トレースキャッシュの全体構成

統合型トレースキャッシュでは，最初に命令アドレスの確認を行う．命令アドレスに対する実行履歴が存在しない場合には通常の命令フェッチを行うために，マルチバンクキャッシュに対し連続した命令列をフェッチするようにアドレスを生成する．

命令アドレスに対する実行履歴 (トレースデータ) が存在するならば，トレースキャッシュとしての動作が必要になる．その場合，対応する命令列のアドレスを複数発行する．そして，これらのアドレスに対してマルチバンクキャッシュ上にデータがあるかどうかのヒット判定を行い，対応する命令列を各バンクから読み出す．これらの読み出された命令列を結合し，トレースとしての命令列の供給を行う．

図 1 に統合型トレースキャッシュの構成図を示し，構成図内の各ユニットについて説明する．その後で，上記の動作について詳細に説明する．

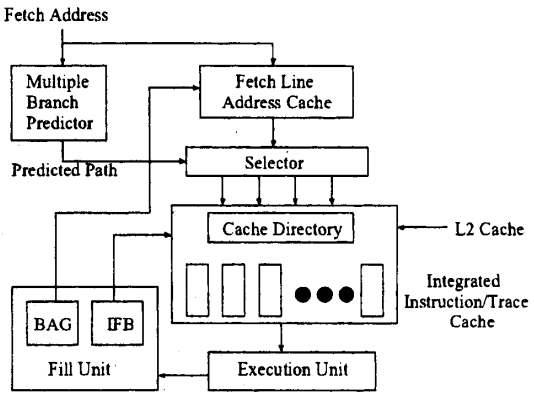

図 1: 統合型トレースキャッシュの全体構成

Cache Directory 従来型のトレースキャッシュでは生成されたトレースデータのみがキャッシュ内に格納されるが，統合型トレースキャッシュでは入力されるデータとしてトレースデータと命令列の 2 つが存在する．これらを格納しているのが Cache Directory である．Cache Directory はマルチバンクメモリ構造をとっており格納されるトレースデータや命令列はバンクの幅に合わせたブロックという単位で格納される．このブロックの大きさとして現在は仮に 4 命令分の大きさとしている．また，トレースデータと命令列が重複する場合，優先的に命令キャッシュ用のデータから置換えを行う．また，トレースキャッシュ用のデータの置換えは，そのエントリ内で最も古いデータから置換えを行うので，Cache Directory 内には常に最新の最も有用なトレースデータが格納されている．

Fetched Line Address Cache(FLAC) トレースを生成するアドレスの集合を格納するキャッシュである．統合型トレースキャッシュにおいて，あるフェッチアドレスに対して複数の Cache Directory 内のバンクからブロックを呼び出し，連結することで長いトレースデータを生成することを実現するために，FLAC ではフェッチアドレスとそのアドレスに対応するフェッチされる可能性のある Cache Directory 内のバンクアドレスグループ (BAG) を管理している．

Multiple Branch Predictor 従来型のトレースキャッシュと同様に，複数の分岐命令による命令の分断に対応するために，トレース結果に基づいたマルチ分岐予測機構により 1 サイクルで複数の分岐予測を行う．

Selector Selector では，Multiple Branch Predictor から予測したパスの情報と，FLAC から BAG に格納されているブロックの情報を受けとり，それら 2 つの情報を照らし合わせ，Cache Directory から呼び出すアドレスを決定し，Cache Directory に供給する．

Fill Unit 実行終了した命令列を結合し，トレースを生成する．そのトレースされたデータを固定長のブロックに分割し Cache Directory に送信するのが IFB である．また Fill Unit では実行された命令列を基に BAG の作成を行い，生成された BAG を FLAC に送る．

命令キャッシュとしての動作 まず，フェッチアドレスを用い FLAC(Fetch Line Address Cache) へアクセスする．FLAC に実行履歴が存在しない場合，FLAC はミスとなる．この場合，統合型トレースキャッシュは命令キャッシュとして振る舞い，FLAC は単純に先頭のアドレスから連続する命令のアドレスをフェッチするように統合型トレースキャッシュにアドレスを出力する．

トレースキャッシュとしての動作 まず先頭のフェッチアドレスを用いて FLAC へアクセスする．FLAC に実行履歴が存在する場合，FLAC はヒットする．その場合に統合型トレースキャッシュはトレースキャッシュとして振る舞う．この場合，FLAC は，入力されたフェッチアドレスから実行される可能性のある複数の分岐先の分岐アドレスが格納された BAG(Bank Address Group) を Selector に送る．正確には BAG には分岐アドレスではなく分岐アドレスをキャッシュ内のブロックアドレスに変換したものが使用される．つまり FLAC は BAG を格納するためのキャッシュの役割をする．Selector は BAG の中から分岐予測結果に基づき，適切なブロックアドレスを選択し，Cache Directory に送信する．Cache Directory は送信されたブロックアドレスを基に命令列のフェッチを行い，最終的にトレースデータとして出力する．

上記のような動作を実現するために，Fill Unit が IFB(Instruction Fill Buffer) を用いてトレースデータを作成し，Cache Directory にストアする．さらに Fill Unit はトレースデータを生成しながら，あるアドレスから実行される可能性のある分岐先アドレスの情報を格納し，それをキャッシュ内のブロックアドレスに変換した後，BAG という形式にまとめ，FLAC に送信する．

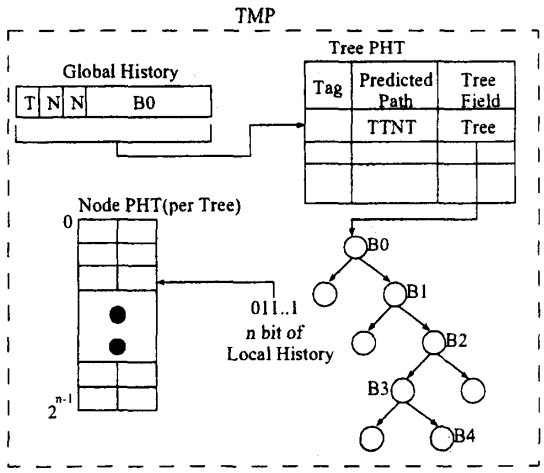

図 2: TMP の分岐予測方法

2.2 問題点

現在の統合型トレースキャッシュの問題点は以下の 2 点である．

- マルチ分岐予測器の検討が行われていない．
- トレースデータがない場合，単純な連続アドレスをフェッチしている．

まず 1 つ目の問題点であるが，統合型トレースキャッシュの中で使用するマルチ分岐予測器について十分な検討が行われていない．そのため実際にマルチ分岐予測器を実装した場合にどの程度の分岐予測ヒット率を達成し，性能がどのような結果になるのか不明である．次に 2 つ目の問題点であるが，マルチ分岐予測器を内包しているにも関わらず，それを通常の命令フェッチに利用していないため，命令フェッチしてきた命令の多くが利用されない場合が多かったと考えられる．そのため通常の命令フェッチが発生する度に性能が低下していたと考えられる．

3 マルチ分岐予測器の実装

本稿ではマルチ分岐予測器として TMP(Tree Based Multiple Branch Predictor)[2] を使用する．本節では TMP について述べ，その後で TMP を統合型トレースキャッシュに実装する方法について説明する．

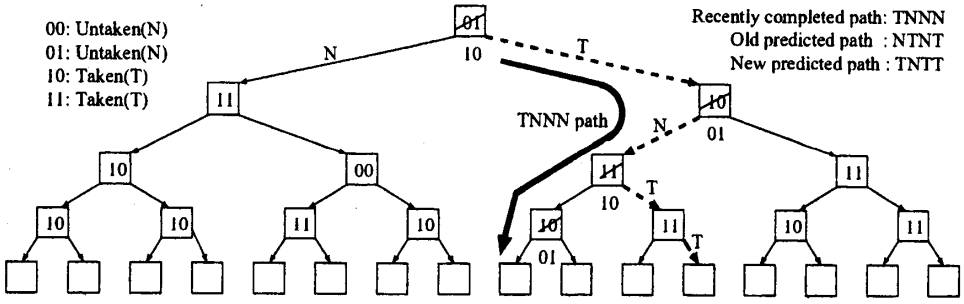

図 3: ツリーノード予測器に 2 ビット飽和カウンタを用いた分岐予測方法

3.1 TMP の特徴

TMP の特徴には以下のようなものがある．1 つ目の特徴として命令完了時に複数の分岐予測を実行することがある．これによりフェッチステージの複雑さ，および分岐のレイテンシを縮小することができる．

2 つ目の特徴として，ある基本ブロックから始まる全てのパスをまとめたものを 1 つのグループとし，そのグループ毎に予測や更新が管理されていることがある．そのため，統合型トレースキャッシュの BAG と同じような構成を取り，導入しやすいと言える．

さらに TMP は木構造をしており，各ツリーの各ノード毎に分岐の履歴を保持しているので，これを用いて各ノード毎に分岐予測を行う．このため従来型の分岐予測器とは異なり独立した更新，および予測ができる．これは，複数の分岐予測器がツリーの中にあることと同じである．これにより，TMP は正確に複数の分岐予測を同時に行うことができる．

3.2 TMP の動作

本節では TMP を用いた分岐予測結果の生成方法について説明する．下記の動作の説明では，図 2 に TMP の分岐予測方法を図示し，図 3 にツリーノード予測器として 2 ビット飽和カウンタを用いた場合の具体的な分岐予測方法について説明する．

1. ツリーノード予測器の更新．まず，過去の分岐の履歴を格納したグローバルヒストリと Fill Unit で作成したブロック集合の先頭のアドレスを使用し，Tree PHT の中のエントリの選択から始まる．その後，選択されたエントリのツリーフィールドにアクセスし，完了したブロックおよび，関連する分岐方向に基づいて，対応するツリーノード予測器を更新する．

   例えば図 3 では，最近完了したブロックを表す TNNN パスに沿ったツリーノード予測器は，それらの 2 ビット飽和カウンタをインクリメントするかデクリメントすることにより更新する．ここでは，最近完了したブロックのパスが TNNN で表されており，T は分岐成立，N は分岐不成立を表している．パスの最初は T，すなわち分岐成立なので，そのノードの保持する 2 ビット飽和カウンタの値 01 をインクリメントして 10 とする．次にパスの 2 番目は N であり，分岐不成立である．2 番目に選択されたノードの保持する 2 ビット飽和カウンタの値は 10 であったが，分岐不成立のためデクリメントして 01 となる．以下この動作を繰り返す．

2. 新しく予測されたパスの生成分岐予測はツリーノード予測器に基づき新しく予測されたパスを生成する．まず，最初に先頭のノードの 2 ビット飽和カウンタの値を確認すると，今回の場合，その値は 10 である．この結果から，分岐予測結果は分岐成立であることが分かる．次に最初のノードから分岐した時のノードの 2 ビット飽和カウンタの値を確認すると，値は 01 であり，分岐予測結果は分岐不成立であると分かる．この動作を繰り返し，今回，図 3 において予測されたパスは TNTT である．

3.3 TMP の統合型トレースキャッシュへの実装

図 4 に統合型トレースキャッシュに TMP を実装した時の図を示す．図 4 の斜線部分は，図 1 の統合型ト

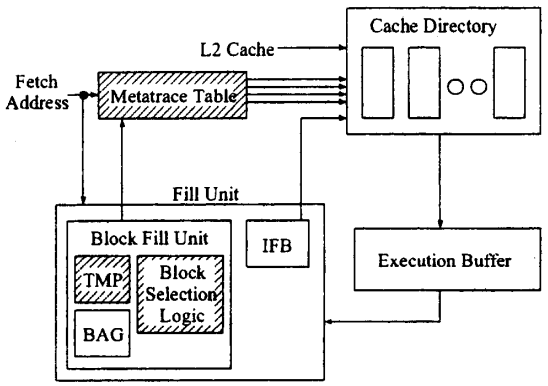

図 4: 統合型トレースキャッシュへの TMP の実装

レースキャッシュの全体構成から新たに追加した部分である.

まずフェッチ動作としては, フェッチアドレスがMetatrace Table に転送される. Metatrace Table は入力されたフェッチアドレスを基に Metatrace Table 内のキャッシュから予測された実行パスに対応する命令ブロックのアドレスを予測された分岐先の個数分だけ呼び出し, Cache Directory に送信する. Cache Directory は入力されたアドレスを基にマルチ分岐予測結果に対応した命令列をプロセッサに送信する.

分岐予測は実行結果からブロックを生成する Fill Unit で行われ, Fill Unit で実際に実行されたパスを基にブロックを生成しながら, TMP を用いたマルチ分岐予測が行われる. Fill Unit で行われたマルチ分岐予測結果は Metatrace Table に送られ, Metatrace Table 内のキャッシュに格納される. 以下に TMP の実装により新たに追加されたユニットの説明を行う.

Metatrace Table 統合型トレースキャッシュの FLAC, Selector の役割を担い, 予測されたパスを構成するブロックへのアドレスを格納する部分である. Metatrace Table は, フェッチアドレスに対して予測されたパスを構成するブロックアドレスのシーケンスを TMP から受け取り Metatrace Table 内のキャッシュに格納する. 各予測パスは次に実行されるであろうと思われる予測結果を 1 つだけ格納している. そのためフェッチ時には分岐予測結果を参照するだけなので, フェッチにかかるレイテンシを縮小している.

TMP TMP は前述のように木構造に基づいた PHT を持っている. 木構造の 1 つのノードは 1 つの基本ブロックを表す. TMP はツリーノード予測器と呼ばれる個々の分岐予測器を木構造にすることにより, 複数の分岐についての予測を効果的にすることができる.

Block Selection Logic ここでは, Metatrace Table に格納されるべきブロックのシーケンスを決め, Metatrace Table に予測パスを構成する情報としてブロックのブロック ID を送る.

4 評価

本節では, TMP を実装した統合型トレースキャッシュの性能を評価するために, C 言語で TMP を実装したスーパスカラプロセッサモデルのシミュレータを作成した. テストプログラムには SPECint95 ベンチマークを用いて, IPC の評価を行う. トレースデータの生成には Simplescalar 2.0[3] を使用している. トレースデータは 5000 万命令実行した中で, 最初の 1000 万命令はプログラムの初期化部分の影響を排除するため評価を取らず, 残りの 4000 万命令の実行結果から評価を取る.

4.1 評価環境

本節ではマルチ分岐予測機構として TMP を用いた統合型トレースキャッシュをスーパスカラプロセッサに搭載した場合の IPC の評価を行う. 評価に用いるスーパスカラプロセッサは, 16 命令同時発行でフェッチ, デコード, レジスタリネーム, レジスタアクセス, 命令スケジューリング, 演算実行, データメモリアクセス, 命令完了の 8 段のパイプラインで構成され, 投機的実行ができるスーパスカラプロセッサとした. また, 分岐の正誤判定の早期判断を促すため本稿では 6 段目の演算実行ステージで分岐の正誤判定ができるようにした. 表 1 に評価パラメータを示す. 同表において, コミットバッファサイズについて事前に評価を行なった結果, サイズが 4096 以上になると IPC が収束していくことが分かった. このため, コミットバッファの影響を排除するため 4096 に定めた.

4.2 評価結果

表 2 に分岐予測のヒット率, IPC の評価結果, 分岐予測が 100% ヒットした場合の理想 IPC を示す. 同

表 1: シミュレーションのパラメータ

キャッシュサイズ	*128KB*
演算実行	1cycle
L2 キャッシュアクセスレイテンシ	8cycle
L2 キャッシュヒット率	100%
データキャッシュヒット率	100%
整数演算器	16
浮動小数演算器	16
データキャッシュポート	4
コミットバッファサイズ	4096
レジスタファイル	4096
ストアバッファサイズ	1024

表 2: IPC の評価結果

ベンチマーク	分岐予測ヒット率	IPC	理想 IPC*
gcc	89.6 %	2.15	7.10
ijpeg	96.0 %	4.49	5.82
li	92.9 %	2.25	6.97
m88ksim	96.5 %	2.36	5.39
go	87.5 %	1.82	9.14
vortex	99.8 %	4.92	7.03

* 分岐予測 100% ヒットの場合

表から分かるように，分岐予測のヒット率は比較的高い値を示していると言える．しかしながら分岐予測が100%ヒットする場合と比較すると，IPC が最大 81%も低下している．これは分岐予測がミスした場合のレイテンシが平均で 10 サイクル以上と非常に大きいためである．このような分岐予測ミスの際のレイテンシが大きいのは分岐予測を用いた命令フェッチからその予測結果が正しいかどうかが分かるまでに多くのサイクルが費やされているのが大きな原因である．ただし，この分岐予測ミスの場合のレイテンシを改善するのは，現状の深いパイプラインを持つプロセッサでは非常に難しい．そこで今後はコンパイラサポートにより分岐予測ミス自体が減るような手法が必要になると考えられる．

5 まとめ

本稿では，統合型トレースキャッシュの機構を利用し，マルチ分岐予測機構を実装したスーパスカラプロセッサの構成方法の提案をした．マルチ分岐予測機構を実装する上で命令キャッシュに相当するキャッシュディレクトリはマルチバンクキャッシュが使用でき，比較的面積を抑えることが出来るが，マルチ分岐予測の予測パスを格納する Metatrace Table が必要となり，チップ上に必要なキャッシュ容量が増大する可能性がある．そのようなスーパスカラプロセッサのシミュレータを作成し，SPECint95 ベンチマークプログラムを使用して分岐予測ヒット率と IPC の評価を行った．その結果，分岐予測ヒット率は 89.6%～99.8%と通常の単一の分岐予測並に高いヒット率を達成した．しかし IPC は分岐予測 100%ヒットの場合と比較して，23%～81%低下するという結果になった．今後は分岐予測ヒット率をさらに高めるためのコンパイラサポートを用いた手法の提案などが必要になると考えられる．

参考文献

[1] 平川泰，弘中哲夫，上口光，Hans Juergen Mattausch，小出哲士．キャッシュの有効利用率を上昇させる命令キャッシュ，トレースキャッシュ統合型キャッシュの提案．情報処理学会研究報告 2002-ARC-142, Vol. 2003, No. 27, pp. 79–84, March 2003.

[2] Ryan Rakvic, Bryan Black, and John Paul Shen. Completion time multiple branch prediction for enhancing trace cache performance. *International Symposium on Computer Architecture*, pp. 45–58, 2000.

[3] Doug Burger and Todd M.Austin. The simplescalar tool set, version 2.0. *University of Wisconsin-Madison Computer Sciences Department Technical Report*, Vol. #1342, , June 1997.