計算機アーキテクチャ 152－14
（２００３．３．１０）

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

平川　泰[†]　弘中哲夫[†]　上口光[††]
Hans Juergen Mattausch[††]　小出哲士[††]

現在，高い命令フェッチバンド幅を実現するキャッシュの構成方式として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いる．しかし，この方式では２つのキャッシュの実装を必要とし，キャッシュ全体としての有効利用率が低いという問題点がある．この問題を解決するために，この２つのキャッシュを統合させた統合型トレースキャッシュを提案した．統合型トレースキャッシュでは，バンク構成を利用することにより分岐予測に従った命令フェッチを可能とした．SPEC95 ベンチマークテストを用いて評価を行い，最大 101%, 平均 21% の性能向上を実現した．また，バンク構造におけるアクセス衝突の回避方法も検討し，バンク衝突の起こらない理想的な場合に対してシングルポートのキャッシュで 4% の性能低下に収めることに成功した．

The proposal of integrted trace cache which combined instruction cache and trace cache

Tai Hirakawa,[†] Tetuo Hironaka,[†] Koh Joguti,[††]
Hans Juergen Mattausch[††] and Tetusi Koide[††]

Recently, the trace cache fetch mechanism is proposed as a method which realizes high instruction fetch band width. However, there is a problem that the effective use rate of the cache is low, since the mechanism needs to implement the instruction cache and the trace cache. For a solution of this problem, we proposed the integrated trace cache system which unified these caches. As a result of the simulation using the SPEC95 benchmarks, the integrated trace cache improved the performance 101%over at the maximum, and 21% over at the average.

1. はじめに

スーパスカラプロセッサに代表される複数命令同時実行を行うプロセッサでは，複数の命令を同時発行するために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．従来の方式では，分岐命令による命令列の分断により分岐予測を正確に行ったとしても十分な命令実行ができない．このため，トレースキャッシュによる命令フェッチが提案されている[1)2)3)]．トレースキャッシュは一度実行した命令列を格納し，再利用することで命令列の分断に対応している．トレースキャッシュを実装するためには既存の命令キャッシュと動的に命令列を格納するトレースキャッシュが必要となる．２つのキャッシュのタグを検索し，選択することで命令発行を行っている．しかし実行履歴と異なるパスが実行される場合には対応できない．また、トレースキャッシュを実装した場合，トレースキャッシュと命令キャッシュ間で重複する命令列が生じ、キャッシュ全体としての有効利用率が減少するという問題点がある．

また，トレースキャッシュと命令キャッシュでは、時系列で必要となるキャッシュ容量が変化するが、従来の２つのキャッシュを分離している方式ではトレースキャッシュと命令キャッシュの容量が静的に決まっているため，容量を動的に変化させることが出来ない．これらの問題を解決するために本稿ではトレースキャッシュと命令キャッシュを統合させた統合型トレースキャッシュを提案する．統合型トレースキャッシュではバンク構成を利用し，分岐命令による命令列の分断に対応することを可能とした．また，バンク構成を実現するにあたってバンクのアクセス衝突の問題が存在する．本稿ではバンク衝突回避ロジックを加えることによりアクセス衝突を減少させる方式も提案し，評価を行った．以下ま

† 広島市立大学
Hiroshima City University
†† 広島大学
Hiroshima University

ず2章ではトレースキャッシュの関連研究について述べる．そして3章で統合型トレースキャッシュの構成について述べ，4章でバンク衝突回避方法を示す．そして，5章で評価結果を示す．

2. 関 連 研 究

従来のトレースキャッシュは命令キャッシュとトレースキャッシュという 2 つの命令フェッチのためのキャッシュが必要となる．このために，

(1) メモリからのデータを直接格納する命令キャッシュと，実行順序に整列されたデータを格納するトレースキャッシュでは命令データは両キャッシュ共に格納されるため，2 つのキャッシュ間で重複する命令が存在する．

(2) 2 つのキャッシュは静的に容量が決まっているため，両キャッシュ間で動的に容量を変化させることが出来ない．

(3) トレースキャッシュでは実行履歴の順序に従い命令を物理的に連続して結合している．このため，実行履歴と一致する命令列しかフェッチ出来ない

の 3 つの問題点が存在する．

問題点 3 について， B. Black らによりバンク構成を利用し，実行履歴と異なるパスを実行可能とした Block-based Trace Cache[3] を提案している．バンク構成を利用したトレースキャッシュでは，基本ブロック単位で命令を格納し，フェッチの際にはバンクに振り分けられた複数の基本ブロックを分岐予測結果に従って結合させることでパスを分岐予測に従い発行可能としている．

3. 統合型トレースキャッシュの提案

トレースキャッシュを利用した命令フェッチ機構ではコアループ部分ではより多くのトレースをトレースキャッシュ内に格納することが必要とされ，メモリから新たな命令列を取り込んで来る場合は命令キャッシュ内の容量が必要とされる．この時系列によって必要とされるキャッシュが変化する．従来の分離しているトレースキャッシュでは静的に容量が決定しているためにこの変化に追従出来ない．

本稿で提案する統合型トレースキャッシュでは従来分離されていた命令キャッシュとトレースキャッシュの 2 つのキャッシュを単一のキャッシュとして管理するため，動的にキャッシュ容量を振り分けることが可能となる．

また統合したトレースキャッシュでは分岐予測に従って命令フェッチを行うために，キャッシュにバンク構造

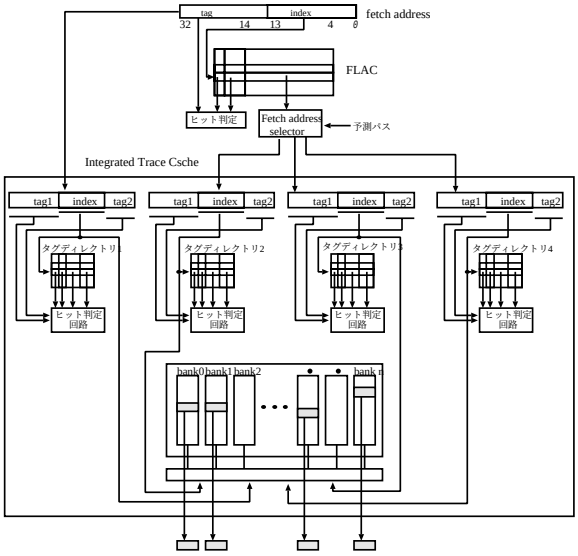

図 1 統合型トレースキャッシュの全体構成

を適用することでラインを微細化する．これにより、命令フェッチ時には分岐予測結果に従い複数のバンクから命令列を読み出すことにより命令列をフェッチする．

図 1に統合型トレースキャッシュの全体構成を示す．

統合型トレースキャッシュではバンクを 4 読み出すためにタグディレクトリを 4 多重化して実装する．フェッチアドレスはまず Fetchd Line Adress Cache（FLAC:3.3.2節で示す）へアクセスされ，対応するバンクのアドレスを発行する．発行されたアドレスはそれぞれヒット判定を行い，ヒットであればそれぞれ対応する命令列がフェッチされる．

統合型トレースキャッシュを実現するためには以下の 3 つの要素が存在する．

(1) 命令キャッシュとトレースキャッシュでのヒット判定方法の統一 (3.1節に示す)

(2) キャッシュメモリのバンク構成の利用 (3.2節に示す)

(3) 複数のバンクにアクセスするためのフェッチアドレスの生成（3.3節に示す）

以下，それぞれの要素と詳細動作について説明する．

3.1 命令キャッシュとトレースキャッシュでのヒット判定方式の統一

統合型トレースキャッシュでは命令キャッシュとトレースキャッシュでのアクセス方法の統一，およびアクセス時にどちらのキャッシュから受け取ったデータなのかを識別することが必要となる．まず，命令キャッシュでは連続した命令が格納されているため，ライン内のデータに自由にアクセスすることが可能である．それに対し，トレースキャッシュでは動的命令流の順序で命令

列が並んでいるため，先頭のアドレスからのアクセスのみ可能である．このヒット判定方法を統一するため，本稿ではデータ識別ビット，及びアクセス用に 2 つの tag，tag1 と tag2 を付化した (図 2)．この方式では、命令キャッシュによるアクセスでは tag1 しか必要とせず，トレースキャッシュ用のデータではアドレスの下位ビットを比較するために tag1 に加えて tag2 を用いてトレースデータの開始位置の比較，判定を行う．トレースキャッシュ識別ビットは格納されているデータがトレースキャッシュのデータであれば 1，命令キャッシュからのデータであれば 0 とし，tag の比較が終った後に使用される．次に，それぞれのデータについてヒット判定方法の詳細を示す．

(1) 命令キャッシュとしてのヒット判定：命令はメモリからの順序通りに並んでいるため従来の命令キャッシュのヒット判定方法と同様にアドレスの上位ビットのアドレスと tag1 を比較し，一致するか判定する．また，トレースキャッシュ識別ビットを確認し，命令キャッシュからのデータであればヒットと判断する．

(2) トレースキャッシュとしてのヒット判定：トレースキャッシュとしてのデータの場合，キャッシュのライン内には分岐先ターゲットを先頭とした命令列が格納されている．このため，トレースキャッシュのヒット判定ではアドレスの下位ビットの比較も必要となる．トレースの先頭のアドレスが一致するかを判定するためアドレスの下位ビットと 2 つ目の tag2 も比較を行う．上位ビットは命令キャッシュと同様に tag1 と比較する．両方の tag が一致し，トレースキャッシュ識別ビットがトレースキャッシュのデータであればトレースキャッシュのデータがヒットとなる．

このようなアクセス方法を採用することにより，トレースキャッシュのデータエントリと命令キャッシュのデータエントリを同一キャッシュ内に共存でき，キャッシュメモリの有効利用を実現する．

この方式により，統合型トレースキャッシュでは命令キャッシュ，トレースキャッシュのデータを同一のインデックスで管理できる．このため，トレースキャッシュとしての命令列と命令キャッシュとしての命令列が同一のインデックスで管理されることになるため，重複命令が削減できる．

3.2 バンク構造の利用

マルチポートメモリを通常のマルチポートメモリとして構成する他，マルチポートメモリの構成方法の 1 つとしてバンク構成メモリを利用する方式が考えられる．バンク構造を効率よく利用する方式として HMA 方式[6] がある．バンク構造を利用しラインを細かく分割することにより，命令キャッシュからのデータ，トレースキャッシュからのデータをより細かく分割することが可能となり、重複する命令数の削減がより効率的に行えることが考えられる．これはラインのサイズが大きい時，1 つ必要な基本ブロックがあればその付近の基本ブロックも保持しなくてはならない．また、従来のトレースキャッシュの問題点である実行履歴と一致しない場合の命令フェッチも解決できる．これは基本ブロック単位で分断し，バンクのライン内に格納することにより，各バンクに振り分けられた命令列を分岐予測に従ってフェッチ出来るからである．

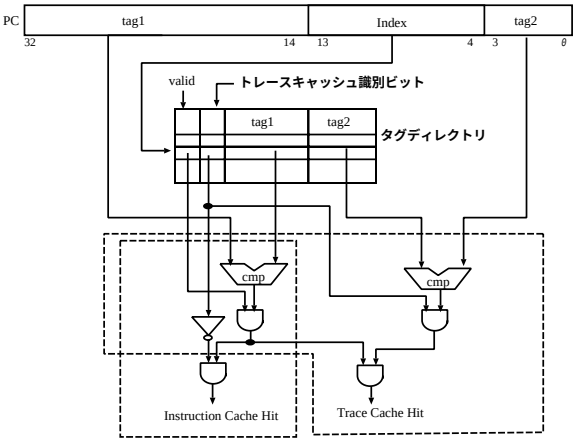

図 2 統合型トレースキャッシュのヒット判定

3.3 複数のバンクにアクセスするためのフェッチアドレスの生成

バンク構成を利用する場合，トレースデータは基本ブロック単位で格納されており，複数のバンクから分岐予測に従ったバンクをフェッチすることによりトレースを生成する．このため，それらのバンクのアドレスを生成することが必要となる．この節では，統合型トレースキャッシュでのアクセスされるバンクのアドレスを生成する方法を示す．

トレースキャッシュを利用する場合，一度実行された命令列を Fill Unit で結合し，格納していくことにより分岐の飛び込み先から分岐命令までの一連の命令列をキャッシュに格納する．以降、この一連の命令列を「擬似的な基本ブロック」と呼ぶ．擬似的な基本ブロックのサイズを確認することにより，過去の履歴から連続してアクセスされると予測される部分を抽出できる．統合型トレースキャッシュではこの擬似的な基本ブロック単位で命令を格納する．このため 1 つのラインには最大 1 つの分岐命令しか存在しない．各ラインは分岐を行う先は

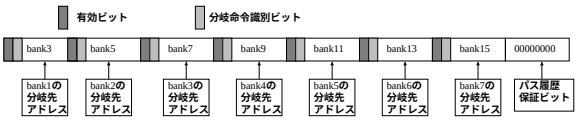

図 3　AFB の詳細

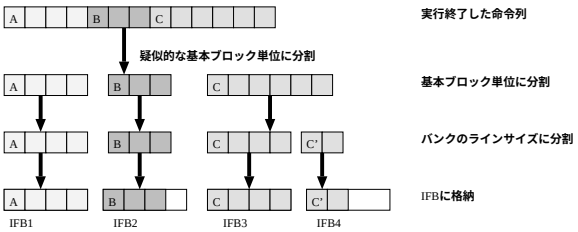

図 4　命令の結合：IFB

1 つとなり，分岐を行わないもしくは分岐命令が存在しない場合連続したバンクにアクセスする．

この方式を実現するために，実行終了した命令列を結合する Fill Unit を改良する．従来は，命令のデータ、アドレスともにセグメントという単位に結合されていたが，アドレスと命令データを分割するため Fill Unit 内部に Instruction Fill Buffer（IFB）と Address Fill Buffer（AFB）を実装する．IFB は実行終了した命令のデータを格納するものであり，バンクのラインは 4 命令なので 4 命令ごと 4 つ用意される．AFB はトレースを生成する一連の疑似的な基本ブロックのアドレスの集合である．AFB はバンク構成を利用したキャッシュのためにフェッチされる可能性のある複数のフェッチアドレスを一つに結合するものであり，命令のアドレスが入力される．トレースキャッシュではバンク単位で命令の結合を行う場合，各バンクのラインには分岐命令は 1 つもしくは存在しないため，あるバンクから分岐するバンクは連続したもの，もしくは分岐先のターゲットとなる．次に AFB の詳細を図 3に示す．IFB は実行された命令のデータを格納するため，AFB は、Fill Unit によって命令が結合されるときに同時に結合される．

3.3.1　命令結合の動作詳細

AFB の結合は，プログラムの実行終了時に行われる．実行終了した命令列は 1 つのパスとなっているため，1 度の結合では十分なアドレスの集合を生成することはできない．図 5で実行されたパスの場合の AFB の動作を示す．図 5より，今回結合された命令列は，1 つ目の分岐命令が分岐を行い，2 つ目の分岐命令は分岐を行っていないため，太線で示すようなパスを実行している．bank6 から bank12 へは分岐命令は存在しないため，連続したバンクにアクセスされる．

今回のパスが実行されたとき，実行結果により図 6のようにアドレスが結合されている．実行されたパスは 5

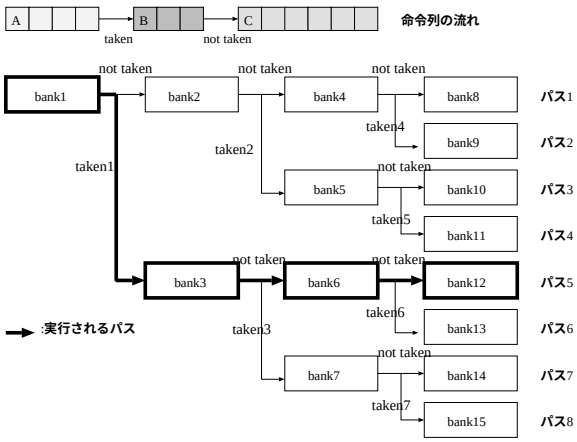

図 5　実行されるパス

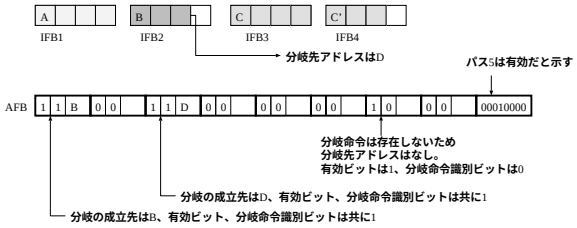

図 6　AFB のアドレス格納

なので，パス履歴情報ビットは 5 ビット目を 1 とし，そのパスは有効とする．実行されたパスでは B の基本ブロックから C の基本ブロックへは分岐不成立であるが，B の基本ブロックの分岐命令は実行され分岐先アドレスは計算される．B の基本ブロックの分岐命令が分岐は成立した場合のために分岐成立先の D のアドレスを格納する．実行されなかったバンクの分岐情報は不明なため，有効ビットは 0 とし、命令の発行を不可能としている．もし A の基本ブロックが分岐しない場合，A から連続するバンクを読み出すこととなり，次のバンクの情報は不明なため、連続したアドレスを発行し，命令をフェッチする．この動作により，1 度の実行でパス 5 に対するアドレスの集合を生成することが出来る．よって，トレースを格納する場合，トレースの命令データは IFB がキャッシュに格納され，AFB は FLAC（3.3.2項で示す）内部の対応するエントリに格納される．

2 度目以降の実行では，AFB を利用してフェッチされた命令列は，命令と共に Fill Unit へ送られ，異なるパスが実行される場合はそのパス情報を AFB に付加する．この動作により AFB に複数のパス情報を格納し，柔軟な命令フェッチを可能とする．

3.3.2　フェッチアドレスの発行

3.3.1項で生成された AFB は，毎サイクル 4 つのバンクにフェッチするために保存される必要がある．こ

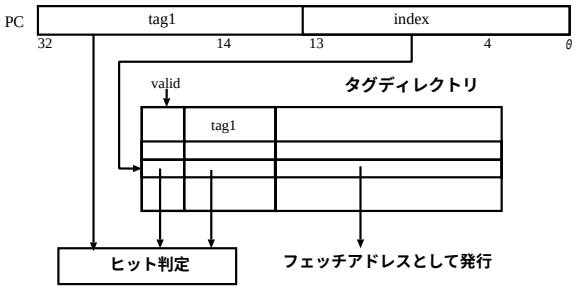

図 7　FLAC の内部構造

のために統合型トレースキャッシュでは Fetched Line Address Cache(FLAC) を実装することを提案する．

統合型トレースキャッシュでは各サイクルに 4 つのバンクをフェッチするため，そのためのアドレスを発行する回路が必要となる．このため，トレースを生成するアドレスの集合である AFB を Fetched Line Address Cache(FLAC) に格納し，4 つのフェッチのためのアドレスを生成する． FLAC の内部を図 7に示す．

統合型トレースキャッシュではまずフェッチの先頭アドレスは FLAC にアクセスされる．FLAC でヒットした場合， FLAC から対応するアドレスを 4 つ統合型トレースキャッシュに発行する． FLAC から発行されるアドレスはフェッチされる可能性のあるアドレスの集合なので，分岐予測に従い，フェッチされるアドレスを選択する．

4. バンク衝突回避手法の提案

統合型トレースキャッシュでは必ずバンク衝突が生じるフェッチのパターンが存在する．この章では，バンク衝突の生じるパターンを説明し，その回避手法について述べる．バンク衝突回避のために以下の 2 つの方法を示す．

(1)　インデックスビットの反転

(2)　フェッチされたバンクのコピー

4.1 インデックスビットの反転

統合型トレースキャッシュのヒット判定方法では分岐命令が分岐が不成立と予測した場合バンク衝突が起こる可能性が高い．これは統合型トレースキャッシュのヒット判定方法に依存するもので，バンクの 1 ラインを 4 命令とした場合，あるインデックスに対して tag2 は 4 つ存在する可能性がある．分岐命令だった場合，分岐の不成立先のインデックスが同一のインデックスに格納される可能性があり，この場合必ずバンク衝突が起こる．

この問題を回避するために命令を結合する際，分岐が不成立の場合の分岐先はインデックスのビットを反転させることを提案する．

表 1　シミュレータの仕様

キャッシュサイズ	16KB ～ 128KB
分岐予測精度	100%
命令結合レイテンシ	5cycle
演算実行	1cycle

分岐不成立の場合にインデックスのビットを反転させることにより，不成立先の命令列は同一のインデックスに格納される可能性は低くなる．

この方式の実現のために，統合型トレースキャッシュのラインにはビット反転を識別するための 1 ビットの情報を付加する．

4.2 フェッチされたバンクのコピー

小さなループ文が繰り返される場合，統合型トレースキャッシュでは同一バンクへのアクセスが集中し，命令のフェッチが出来ない場合が存在する．このため，小さなループ文を連続してフェッチを行う場合，そのループ文をコピーしてフェッチを行う．コピー対象となるループ文の検索には， AFB 内のターゲットアドレスにより簡単に検索を行うことができる．もし AFB 内に同一のターゲットがあれば，先の命令のみフェッチを行い，その命令をコピーすることでフェッチが可能となる．

5. 性能評価

統合型トレースキャッシュの評価として， C 言語によるトレース駆動シミュレータを作成し，性能評価を行った．ベンチマークプログラムとして SPEC CPU 95 整数ベンチマークを使用し，トレースデータの作成には Simplescalar 2.0 を使用している．なお， SPEC CPU 95 整数ベンチマークのバイナリは Simplescalar の WEB ページからダウンロードした．

5.1 評価環境

表 1にシミュレータの仕様を示す．

まず，提案した統合型トレースキャッシュと従来型トレースキャッシュの評価結果を示す．また，統合型トレースキャッシュはバンク衝突の起こらない理想的な状況として評価を行った．

統合型トレースキャッシュと従来型のトレースキャッシュのキャッシュ容量が 64KB の場合の毎サイクルでの平均命令フェッチ数を図 8に示す．統合型トレースキャッシュでは特に Compress において性能の向上が見られた．これは，基本ブロックサイズが大きな場合に統合型トレースキャッシュの命令フェッチに利点があるためだと考えられる．

次に，アクセス衝突回避手法を付化し，性能評価を行った．図 9にバンク数は 64，キャッシュサイズは 64KB の場合の提案したアクセス衝突回避手法を用いた

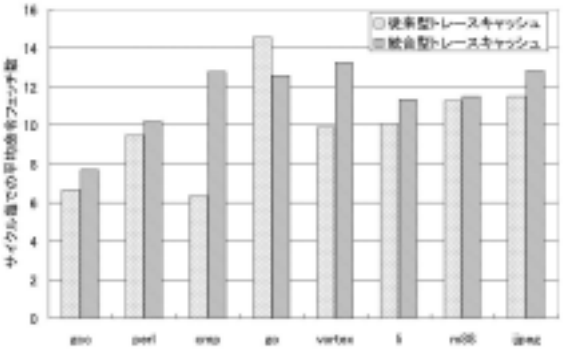

図 8　提案方式と従来型トレースキャッシュの比較

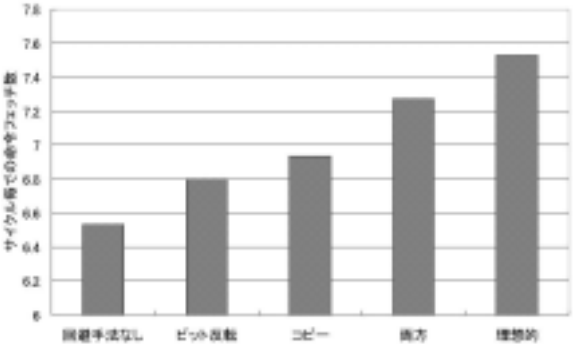

図 9　バンク衝突回避手法の比較

場合と用いない場合のサイクル毎での平均命令フェッチ数を示す．

この結果より，ビット反転，命令のコピーのどちらもバンク衝突を減少させることが出来た．さらに，両方を実装させた場合は理想的な場合よりサイクル毎での命令フェッチ数は 4% 程度の減少となった．また，従来型トレースキャッシュと比較しても約 11% 性能が向上した．バンク衝突回避手法を適応しない場合，統合型トレースキャッシュは従来型のトレースキャッシュとほぼ同程度の性能しか得られない．しかし，バンク数を減少させた場合，ビット反転のアクセス回避手法の効果は減少していった．これは，ビット反転を行いインデックスを振り分けることでバンク数が少なくなるほど衝突確率が上昇したためだと考えられる．

6. ま と め

本論文ではトレースキャッシュと命令キャッシュの統合型トレースキャッシュの構成方式について述べ，従来型トレースキャッシュとの比較を行った．その結果，バンク衝突が起こらない場合の統合型トレースキャッシュは最大 101%，平均 21% サイクル毎における平均命令フェッチ数が向上した．またバンク衝突回避手法にインデックスビットの反転，同一命令フェッチのコピーを提案し，提案手法を用いることでバンク衝突を最大 85% 減少させ，サイクル毎での平均命令フェッチ数もバンク衝突の起こらない場合よりも 4% 程度の減少に収めることに成功した．

今後の課題として，統合型トレースキャッシュの面積評価，重複する命令の削減方式の検討を行う．また命令をフェッチする際に分岐予測の精度が重要な問題になる．本論文では分岐予測精度は 100% として評価を行ったが，今後は分岐予測精度も含めた正確な評価を行っていく予定である．

謝辞　本研究の機会を頂き，御指導頂いた北村俊明教授に深甚なる謝意を表します．また本研究にご協力頂いた半導体理工学研究センター（STARC）に感謝の意を表します．

参 考 文 献

1) Eric Rotenberg, Steve Bennett, and Sanjay Jeram Patel: *Trace Cache : A Low Latency Approach to High Bandwidth Instruction Fetching* 29th Annual Tnternational Symposium on Microarchitecture(December,1996)
2) Friendly, Daniel H., Patel, Sanjay J., and Patt, Yale N: *Alternative Fetch and Issue Policies for the Trace Cache Fetch Mechanism* Proceedings of the 30th AC M/IEEE International Symposium on Microarchitecture(November, 1997)
3) B. Black, B. Rychlik and J. Shen: *The Block-based Trace cache*, In Proceedings of the 26th Annual International Symposium on Computer Architecture (May 1999)
4) Ryan Rankvic,Bryan Black and John Paul Shen: *Complention Time Multiple Branch Prediction for Enhancing Trace Cache Performance*, International Symposium on Computer Architecture(june 2000)
5) Tse-Yu Yeh,Debrah T.Marr,Yale N.Patt: *Increasing the Instruction Fetch Rate via Multiple Branch Prediction and a Branch Address Cache*, International Conference on Supercomputing(july 1993)
6) H.J. Mattausch:*Hierarchical N-Port MemoryArchitecture based on 1-Port Memory Cells*, Proc.23rd European Solid-State Circuits Conf.,Southampton, UK, 16-18 September, pp.348-351,1997
7) Kevin Skadron, Pritpal S. Ahuja, Margaret Martonosi, Douglas W. Clark:*Branch Perdiction, Instruction window size, and Cache Size: Performance Tradeoffs and Simulation Techniques*, IEEE Transaction on Computers(1999)