2003－ARC－155　(5)
2003／11／28

非数値演算を効率良く実行する統合型トレースキャッシュの評価

平川　泰[†]　上口　光[††]　弘中哲夫[†]
マタウシュ ハンスユルゲン[††]　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実行終了した命令列を格納し，再利用するトレースキャッシュが提案されている．しかし，この方式では命令データを格納するために実行履歴を格納するトレースキャッシュと、メモリからのデータを格納する命令キャッシュの 2 つの異なるキャッシュが命令フェッチのために必要となる．この 2 つのキャッシュ容量はプログラムの実行過程や，ワークロードによって必要とされる容量が変化する．この変化に追従するために，本稿では従来 2 つ必要であったキャッシュを 1 つに統合することにより，動的にキャッシュ容量を変化させ実行過程やワークロードの変化に追従可能な統合型トレースキャッシュを提案している．提案する統合型トレースキャッシュを SPEC95 ベンチマークに含まれているプログラムを用いて評価を行い，特に gcc などの非数値演算プログラムにおいて平均命令フェッチ数は最大 15%, 平均 8.8% の性能向上を実現した．

Evolution of Non-Numerical Computation performance by Integration of Instruction and Trace Cache

Tai Hirakawa,[†] Koh Johguchi,[††] Tetsuo Hironaka,[†]
Hans Jürgen Mattausch[††] and Tetsushi Koide[††]

Recently, the trace cache mechanism is proposed as a method which realizes high instruction fetch band width. However, normal implementation use two separate cache memories with fixed storage capacity. But, the optimum size of the two caches changes during program execution. To overcome this problem, we proposed an integrated instruction/trace cache system. Simulations using the SPEC95 benchmarks, we show that the proposed integrated instruction/trace cache improves the performance, especially for non-numerical computations.

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．例えば 16 命令同時実行を考えた場合，分岐命令は 4 ～ 5 命令に 1 つといわれているため，16 命令の中に 3 ～ 4 の分岐命令が存在することとなる．命令フェッチバンド幅を増加させる方式の 1 つとしてキャッシュのラインサイズを大きくする方式があるが，この方式では分岐命令による命令列の分断により十分な命令供給を行うことができない．このため，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いる．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する．しかしながら，トレースキャッシュを実装するためにはメモリからの命令データを格納する既存の命令キャッシュと実行終了した命令列を格納するトレースキャッシュの 2 つのキャッシュが必要となる．

トレースキャッシュを利用した命令フェッチ機構では命令フェッチ時に 2 つのキャッシュのヒット状況を確認し，トレースキャッシュがヒットしていればトレースキャッシュから，トレースキャッシュがミスしていれば命令キャッシュから命令フェッチを行う．このため，コアループ部分ではトレースキャッシュに多くの容量が必要となり，新たな命令列を実行する際には命令キャッシュに多くの容量が必要とされる．実際に実行するプログラムは，一定のパスを繰り返し実行し，プログラムの実行範囲が小さい様な数値演算系のプログラムと，パスが一定ではなく，プログラムの実行範囲が大きい数値演算系のプログラムが存在する．故に，トレースキャッシュは数値演算の様なプログラムでは性能を十分に生かすことができ，多くの命令キャッシュ容量を必要としない．しかし，非数値演算の様にワークロードが大きく，パスが一定ではない様なプログラムでは命令キャッシュのサポートが必要不可欠となる．このような場合，従来

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

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は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．

この問題点の解決のために，本稿ではトレースキャッシュと命令キャッシュを統合した統合型トレースキャッシュを提案する．2 つのキャッシュを統合させることで時系列に応じて動的にトレースキャッシュ，命令キャッシュとして必要な容量の変化に対応する．

本稿の構成を以下に示す．第 2章で従来型トレースキャッシュについて述べ，第3章で統合型トレースキャッシュの提案を行い，第 4章で統合型トレースキャッシュを構成するために必要となるバンク構成の利用，第 5章で統合型トレースキャッシュでのヒット判定方法，第 6章でシミュレーションによる評価を行い，第 7章でまとめる．

2. 関 連 研 究

従来型トレースキャッシュの構成方法としてこれまで以下の 2 つの代表的な方法が研究されてきた．

(1) 実行履歴を 1 つのラインに結合し格納するトレースキャッシュ[1)]

(2) バンク構成を利用することで，実行履歴を分割し格納するバンク構成型トレースキャッシュ (Block Based Trace Cache[3)])

本稿では以下，(1) を 1 ライン型トレースキャッシュ，(2) をバンク構成型トレースキャッシュと呼ぶ．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 については，バンク構成型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

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．このため，トレースキャッシュ，命令キャッシュとしての容量を動的に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，バンク構成型トレースキャッシュを適用し複数のラインを読み出して結合することでトレースを生成する．

本稿では統合型トレースキャッシュは 16 命令フェッチとした．バンクのラインサイズ，読み出しバンク数は可変であるが，本研究では各バンクのラインサイズは 4 命令とし，4 バンク読み出しを行うことにより 16 命令フェッチを可能とする．このため，この条件では統合型トレースキャッシュの読み出しポート数は 4 とする．以降すべてこの条件を用いる．

3.1 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，最初にフェッチアドレスの確認を行う．命令の実行履歴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となるため，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．この統合型トレースキャッシュの構成図を図 1に示し，各ユニットの動作を以下に示す．

3.2 統合型トレースキャッシュ（Integrated Instruction/Trace Cache)

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は以下の 2 つが存在する．

(1) L2 キャッシュからの命令キャッシュ用のデータ．

(2) Fill Unit からのトレースキャッシュ用のデータ．

また，複数のラインをフェッチし，それらを結合しトレースを生成するため，マルチポートのキャッシュが必要となる．このため，本稿ではバンク構成を利用しマルチポートのキャッシュを実現する．このため，以下の機能が必要となる．

(1) バンク構成の利用（4章に示す）

(2) データの配置方法と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 (5章に示す)

(3) ポート毎のアドレスのヒット判定を行うため，読み出しポート数に合わせたタグディレクトリの多重化（4 ポート読み出しならば 4 つ多重化）

3.3 Fetched Line Address Cache

Fetched Line Address Cache (FLAC) は，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ではバンク構成を利用し，複数のバンクから命令を読み出してトレースを

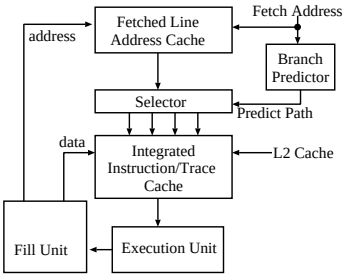

図 1 統合型トレースキャッシュの全体構成

生成する．このため，トレースを生成するアドレスの集合を保持する必要が生じる．FLAC では Fill Unit で結合された命令列のアドレスの集合を格納し，命令フェッチ時にはその対応するアドレスの集合を統合型トレースキャッシュに供給する．また，FLAC は Branch Target Buffer (BTB) としての働きを兼ねるため，統合型トレースキャッシュでは BTB を必要としない．

3.4 Fill Unit

Fill Unit は従来型トレースキャッシュと同様に実行終了した命令列を結合しトレースを生成する．従来型との相違点は，基本ブロック単位で管理を行い，命令のデータとアドレスを別々に管理を行う．

3.5 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列の分断に対応するため，分岐予測機構は 1 サイクルで複数の分岐予測を行うことが必要となる．[4)]

3.6 統合型トレースキャッシュの動作概要

統合型トレースキャッシュの動作は以下の流れで行われる．

(1) フェッチアドレスで FLAC へアクセス．
(2) FLAC から対応するアドレスの集合を読み出しセレクタへ送る．
(3) セレクタで分岐予測に応じてアクセスするアドレスを選択．
(4) 統合型トレースキャッシュで対応するアドレスに対してヒット判定．
(5) ヒットなら，統合型トレースキャッシュ内の対応する命令列を読み出し，供給する．ミスならば L2 キャッシュへ必要なデータのアクセスをする．

これらの動作は，命令キャッシュとして動作する場合，トレースキャッシュとして動作する場合の 2 つの状況に分けることが出来る．以下，これらの 2 つの状況について示す．

- 命令キャッシュとしての動作
  命令列の実行履歴が存在しない場合，命令キャッシュとしての動作が必要とされる．まず先頭のフェッチアドレスが FLAC へアクセスする．実行履歴が存在しない場合，FLAC はミスとなる．この場合，トレースキャッシュとしてのデータは存在しないため，命令キャッシュとしてのアクセスとして単純に先頭のアドレスから連続する命令のフェッチを行う．統合型トレースキャッシュでは 4 個のバンクを読み出し 16 命令フェッチを行うため，命令キャッシュとしてフェッチを行う場合，先頭のアドレスから 4 つの連続するバンクにアクセスし，命令フェッチを行う．
- トレースキャッシュとしての動作
  トレースキャッシュとして動作する場合，まず先頭のフェッチアドレスが FLAC へアクセスする．FLAC がヒットすると，実行履歴が存在するためトレースキャッシュとしての命令フェッチを行う．

4. キャッシュメモリのバンク構造の利用

キャッシュを完全な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．現在，バンク構造を効率よく利用する方式として HMA 方式[7)] が提案されている．バンク構成メモリでは，1 ポートのメモリセルを用いた 1 ポートメモリバンクを基本構造とし，1 つのメモリセルに対する配線領域を減少させる．

統合型トレースキャッシュではキャッシュメモリにバンク構成を利用することで，以下の 2 つの利点が挙げられる．

(1) 重複する命令数の削減がより効率的に行える．
  基本ブロックサイズは平均 4 〜 5 命令と言われているので 1 ラインを 16 命令とした場合，3 〜 4 程度の基本ブロックが格納されている．このため，必要な基本ブロックが 1 つあった場合，16 命令のラインでは付近の基本ブロックも保持しなくてはならない．しかし，統合型トレースキャッシュではよりラインを細かく分割し管理するため，必要となる基本ブロックをより細かな範囲で管理することが可能となる．
(2) 実行履歴と一致しない場合の命令フェッチの実現．
  統合型トレースキャッシュでは細かく分割した複数のラインを同時に読み出し，それらの命令列を 1 つのトレースとして結合を行う．バンク構成を利用しマルチポートのキャッシュを実現することで各バンクに振り分けられた命令列を同時に読み出すことを可能とし，分岐予測に従ってフェッチすることが可能となる．

しかし，各バンクは 1 ポートとなっているため，同時に同一のバンクへアクセスが生じた場合，アクセス衝突が生じ，1 つのデータしか取り出すことが出来ない．

5. ヒット判定方法

統合型トレースキャッシュでは，命令キャッシュとし

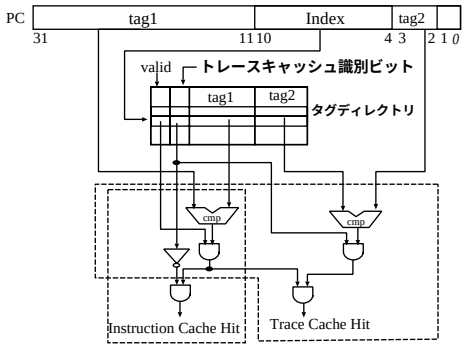

図 2 統合型トレースキャッシュのヒット判定

てのデータと，トレースキャッシュとしてのデータを識別するヒット判定方法の統一が必要となる．また，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．本稿ではデータ識別ビット，及びアクセス用に 2 つの tag として， tag1 と tag2 を付加する (図 2)．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．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．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．

5.1 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（トレースキャッシュ識別ビットが 0 であれば）ヒットと判断する．

5.2 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識別ビットが 1）トレースキャッシュのデータをヒットと判断する．

5.3 フェッチアドレスの発行

Fill Unit で生成されたアドレスは，毎サイクル 4 つのバンクをフェッチするために統合型トレースキャッシュ内へ保存される必要がある．このために統合型トレースキャッシュでは 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．

6. 統合型トレースキャッシュの性能評価

提案した統合型トレースキャッシュの評価を行うために 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．トレースキャッシュは以下の 3 つの構成方式の評価を行った．

(1) 1 ライン型トレースキャッシュ
(2) バンク構成型トレースキャッシュ
(3) 統合型トレースキャッシュ

の 3 つの構成方式の評価を行った．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．

6.1 評価環境

統合型トレースキャッシュの命令フェッチ効率を比較する．表 1にシミュレータの仕様を示す．従来型のトレースキャッシュはトレースキャッシュ，命令キャッシュ共に 4 ウェイセットアソシアティブとし，統合型トレースキャッシュは同一のインデックスに命令とトレースのデータが格納されるためウェイ数は 8 とした． L2 キャッシュレイテンシは文献[3] と同様に 8 サイクルとした．なお，評価では統合型トレースキャッシュの命令フェッチ効率のみに着目するため L2 キャッシュ，および分岐予測精度は 100% ヒットとした．また，バンク衝突についてはバンク構成型トレースキャッシュの命令格納方式によって左右されるため，今回の評価ではバンク衝突は起こらないこととした☆．

☆ 紙面の都合上割合したが，バンク衝突の回避手法についても提案しており，提案方式を用いた場合バンク衝突が起こらない場合に比べて平均 6% の性能低下に抑えることに成功している．なお，文献[3] のバンクキャッシュはバンク競合を考慮していない

表 1　シミュレータの仕様

キャッシュサイズ	8KB ～ 128KB
バンク数	32
分岐予測精度	100%
命令結合レイテンシ	5cycle
演算実行	1cycle
$L2$ キャッシュアクセスレイテンシ	8cycle

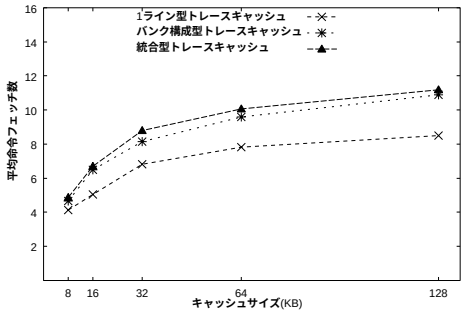

図 3　統合型トレースキャッシュと従来型トレースキャッシュの比較（gcc）

6.2　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比較

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比較を行うために，キャッシュ容量を8K ～128K まで変化させ性能の比較を行った．プログラムはSPEC ベンチマークのすべてを実行したが，gcc のみに注目し，図 3に評価結果を示す．

また，従来型トレースキャッシュでは命令キャッシュの容量が重要となる．事前評価を行った結果，従来型のトレースキャッシュで最も性能がよい命令キャッシュ容量は 1 ライン型トレースキャッシュでは総キャッシュ容量の半分，バンク構成型トレースキャッシュでは命令キャッシュは 4KB となった．このため，今回の評価では従来型トレースキャッシュの命令キャッシュの容量は以上のように定めた．

これらの結果から，非数値演算プログラムの場合，ワークロードは大きくなるため命令キャッシュの容量が多く必要となる．また，分岐が一定方向に安定していないため，1 ライン型のトレースキャッシュでは分岐予測に対応できない．従って，バンク構成型トレースキャッシュよりも性能が劣っている．

gcc では，統合型トレースキャッシュはバンク構成型トレースキャッシュよりキャッシュ容量 32KB の時最大 7%，平均 4% 性能が向上している．

この評価のため，gcc について各命令アドレスに存在する命令の実行頻度を解析した．この結果を図 4に示す．横軸にアドレス，縦軸に各アドレスの実行回数を示す．この結果より，gcc では幅広いアドレス領域の実行が必要となる．このため，命令キャッシュとしての性能が多く必要とされ，統合型トレースキャッシュの方が高い性能を示した．

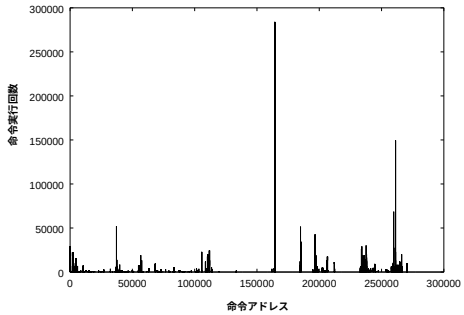

図 4　gcc のアドレス利用領域

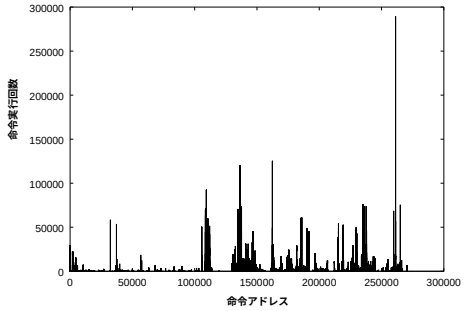

図 5　gcc -O2 のアドレス利用領域

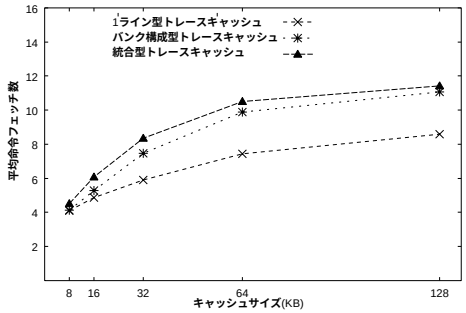

図 6　統合型トレースキャッシュと従来型トレースキャッシュの比較（gcc -O2）

また，gcc により多くの非数値演算的な振舞いをさせるため，ベンチマークプログラム gcc の入力に最適化オプション -O2 を使用し評価を行った．このプログラムの振舞いを図 5に示す．

この結果，最適化オプションをつけない場合に比べてプログラムの使用される領域は上昇した．また，この時の統合型トレースキャッシュの評価結果を図 6に示す．

この時，統合型トレースキャッシュはキャッシュ容量 16KB の時最大 15%，平均 8.8% 性能が向上した．

6.2.1　考　　察

統合型トレースキャッシュでは，特にワークロードが大きな場合性能が向上した．この時の統合型トレース

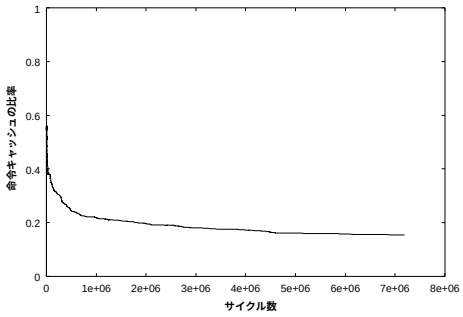

図7 統合型トレースキャッシュ内のデータ比率

キャッシュ内の命令キャッシュとしてのデータとトレースキャッシュとしてのデータの比率を図7に示す.

この結果，特に初期状態において，命令キャッシュの容量が統合型トレースキャッシュ内に多く存在している．初期状態では，トレースデータは作成されていないため，命令キャッシュとしてのデータが多く必要となる．このため，初期状態では統合型トレースキャッシュの方がより高速に動作していると考えられる．

また，キャッシュとして安定状態に入った後も，置換の対象となるデータは命令キャッシュのデータとなっている．統合型トレースキャッシュの命令配置方式ではトレースデータと命令キャッシュとしてのデータが同一のインデックスに割り振られるため，重複しているデータは自動的に置換対象となっている．このため，キャッシュの有効利用率が高く，性能が向上していると考えられる．

7. 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．ワークロードの小さなプログラムの場合では，バンク構成型トレースキャッシュと大きな差は生じなかったが，特にgccを-O2オプションを用いて実行した場合，平均8.8%，最大15%性能が向上した.

また，従来型トレースキャッシュはトレースキャッシュ内に十分に命令を格納することが出来れば高い性能を示すことが出来る．統合型トレースキャッシュはキャッシュ容量が十分な場合はバンク構成型トレースキャッシュと同等の性能であるが，キャッシュ容量が小さな場合では性能が向上している．このため，統合型トレースキャッシュは実際のプロセッサで時分割処理などを行い使用できるキャッシュ容量が減少したような場合，従来型トレースキャッシュよりも特に有利だと考えられる.

また，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，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．本論文では命令フェッチ効率を評価するために分岐予測精度は100%として評価を行ったが，今後はプロセッサの性能向上率を明確にするため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．本研究の一部は文部科学省科学研究費（若手研究（B） 15700068）の助成を得た.

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) 斉藤史子，山名早人：投棄的実行に関する最新技術動向，情報処理学会研究報告， ARC-145-11,pp.67-72,2001
6) 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)
7) 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
8) 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)