2007−ARC−175 (18)
2007/11/22

異種命令セット同時実行プロセッサの実装に向けた評価

田端 猛一[†] 塩田 耕太郎[††] 北村 俊明[†]

近年，携帯機器などの機能向上が激しく，これに用いられるプロセッサは，高性能化と低消費エネルギーの両立を要求されている．我々は，これに応える1方式として VLIW バックエンド上に異なる命令セットの SMT(Simultaneous Multi-Threading) を構成する OROCHI プロセッサを提案し開発を行っている．プロセッサを実装するにあたり，その独特な構造による性能と面積のトレードオフが多数存在しており，それらを適切に実装しなければ，本来のパフォーマンスを得ることができない．本稿では，これら性能と面積のトレードオフについてまとめ，その中の特徴の異なるスレッド間の競合によるキャッシュヒット率の評価を行うとともに，他のスレッドのパイプラインのストールがどの程度影響を与えているか評価を行う．

Evaluation of a Simultaneous Multiple Instruction-sets Execution Processor towerd the Implementation

Takekazu Tabata,[†] Kotaro Shiota[††] and Toshiaki Kitamura[†]

Recently, mobile devices have many functionalities and these applications require high performance and low power consumption to processors used in them. As one method to answer these situations, we are developing the OROCHI processor which composes SMT (Simultaneous Multi-Threading) of a different instruction sets on the VLIW backend. This processor has many trade offs of the performance and the area by the unique structure. To get the maximum performance, they need appropriately implementation. In this paper, we describe these trade offs. And, we evaluate the data cache hit rate by influence of conflict between threads which have the different feature and the effect by pipeline stalls of another thread.

1. はじめに

近年，組み込みプロセッサには，高解像度の映像処理や音楽の再生などといった処理をプログラムにより実現するため，性能向上が求められている．

しかし，これらの要求を満たす高性能な組み込みプロセッサではクロック周波数の上昇やスーパースカラ化による並列化機構の実装などに伴い消費電力の増加が引き起こされている．組み込みプロセッサは PDA や携帯電話などといった，バッテリ駆動の機器にも用いられることが多く，消費電力の増大は機器の駆動時間低下に直結してしまうため，高性能と低消費電力の両立が求められている．

これらの要求に答えるため，我々は VLIW プロセッサに着目し評価を行ってきた．結果として，コンパイル時に並列性を引き出せるようなプログラムにおいては，効率の良い実行が可能であることが分かったが，システム構築時には，システム制御のような VLIW プロセッサに向かないプログラムも存在している．そこで，我々は VLIW のバックエンド上に命令セットの異なる SMT(Simultaneous Multi-Threading) を構築し，それぞれのスレッドに汎用プロセッサ，アプリケーションプロセッサの役割を割り付けることにより仮想的なマルチプロセッサ環境を提供する，OROCHI プロセッサ[1)2)] を提案しており，現在その研究・開発を行っている．

OROCHI プロセッサの実装作業を進める上で，VLIW のバックエンド上を異なる特徴を持つスレッドが同時に実行されることによる，独特な制御を行う部分が存在する．これらの制御には性能・面積に影響を与えるようなトレードオフが存在しており，性能を引き出すためには，トレードオフを適切に見極め，制御部の実装を行う必要がある．

本稿では，これら OROCHI プロセッサを実装する上での性能と面積のトレードオフについて述べ，その中で，特徴の異なるスレッド間でのデータキャッシュの競合によるヒット率の低下と他のスレッドの影響によるパイプラインのストールに着目し，実装方法につ

† 広島市立大学大学院情報科学研究科
Graduate School of Information Sciences, Hiroshima City Universit
†† 広島市立大学情報科学部
Faculty of Information Sciences, Hiroshima City Universit

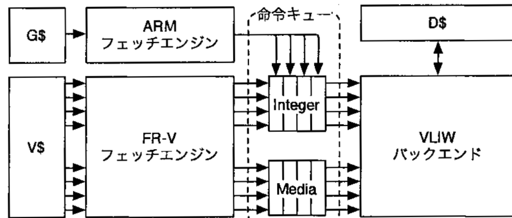

図 1 OROCHI ブロック図

いて検討を行う．

2. OROCHI プロセッサ

本章では，我々が提案している異なる命令セットを SMT に実行する OROCHI プロセッサの概要を述べる．

SMT を構成する 2 種類のスレッドを，汎用プロセッサの命令を実行する汎用スレッド，アプリケーションプロセッサの命令を実行するアプリケーションスレッドと定義した．それぞれのスレッドは以下のような特徴を持つ．

アプリケーションスレッドには FR550 命令セットのサブセット（以下 FR-V 命令）を採用した．FR550 は同時に 8 命令発行可能な FR-V ファミリーの VLIW プロセッサで，整数演算・浮動小数点演算・メディア演算の命令を持つ．今回のターゲットとするアプリケーションとして高い性能を要求するマルチメディア処理を考えたため，浮動小数点演算を除いた，整数演算とメディア演算命令のサブセットを命令セットとして採用し，最大 8 命令を同時に発行可能な仕様とした．

一方，汎用スレッドには ARMv4 命令セット（以下 ARM 命令）を採用した．特権モードを含めた基本的な命令セットを実装し，組み込み用 Linux である μ clinux などの制御プログラムの実行を想定している．ソフトウェア割り込みや外部割込みの処理はすべて汎用スレッドで行う．そのため，アプリケーションスレッド側には特権モードを用意しておらず，ソフトウェア割り込みや例外処理が必要な場合，汎用スレッド側に処理を委託する．

図 1 に OROCHI プロセッサのブロック図を示す．OROCHI プロセッサは FR-V フェッチエンジン，ARM フェッチエンジン，命令キュー，バックエンドエンジンとそれに付随するキャッシュ G$, V$, D$ というモジュールで構成される．

アプリケーションスレッドの FR-V 命令は FR-V フェッチエンジンにより，汎用スレッドの ARM 命令は ARM フェッチエンジンによりそれぞれフェッチされ，デコードなどの処理を経た後，命令キューにエンキューされる．命令キューにエンキューされた命令は，In-order にバックエンドエンジンに送り出され実行される．バックエンドエンジンに接続されるデータキャッシュ(D$) は 2 つのスレッドで共有して利用する．

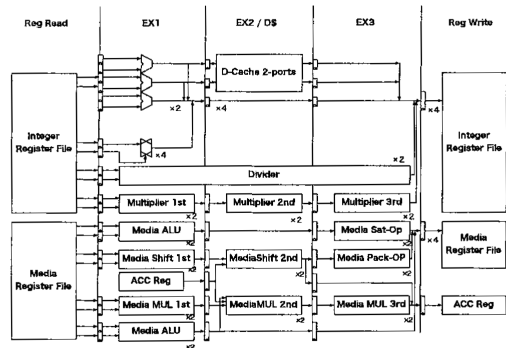

図 2 バックエンドエンジンブロック図

以下に各モジュールの動作と構成について述べる．

FR-V フェッチエンジン　アプリケーションスレッドが実行する FR-V 命令を V$から同時に 8 命令フェッチし，デコードを行い命令キューにエンキューするモジュールである．

ARM フェッチエンジン　汎用スレッドが実行する ARM 命令を G$からフェッチし，命令を分解後[3]，デコードを行う．命令分解というのは，ARM 命令セットには 1 命令でシフトを行い，その結果を用いた演算を行うものやレジスタ複数本をメモリに転送するものが存在しており，このような命令をバックエンドエンジンで実行できる単純な命令に分け，中間レジスタを介し実行を行えるようにするものである．その後分解後命令はデコードされ，同時に命令キューの空きを見て，キュー内に空きがあればエンキューする．

命令キュー　フェッチエンジンとバックエンドエンジンを繋ぐモジュールで，各フェッチエンジンから送られてきた FR-V 命令と ARM 命令をバックエンド部に送り出す働きを持っている．命令キューは整数演算命令が入る整数演算キューとメディア演算命令が入るメディア演算キューとに分かれている．FR-V 命令は整数演算キューとメディア演算キュー両側にエンキューされるが，ARM 命令は整数演算命令のみであるため，整数演算キューのみにエンキューされる．

バックエンドエンジン　バックエンドエンジンのパイプラインは，図 2 に示すようにメディア演算や乗算にあわせて 5 段の構成となっており，命令キューから送られてきた命令を In-order に実行する．

3. 関連研究

OROCHIプロセッサのデータキャッシュメモリのように，SMTプロセッサに搭載されるキャッシュメモリでは，複数スレッドが並列に実行されることにより，破壊的干渉が発生することが知られている．破壊的干渉は，スレッド間でのキャッシュラインの競合により，キャッシュのあるスレッドのデータを他のスレッドのデータが上書きすることによって発生するキャッシュミスが多発する状況を指す．この破壊的干渉が発生する場合，各スレッドの性能が大きく低下することが問題となっており，これまでも，破壊的干渉を抑制するためのキャッシュ置き換えアルゴリズムや制御方式が提案されている．

山崎らの研究[4]は，セットアソシアティブ方式のキャッシュのマルチスレッドアーキテクチャ下でスレッドの処理数に応じて，キャッシュラインのリプレースメントを制限する動的スレッドアソシアティブ方式により，スレッド間でのキャッシュの破壊的干渉削減を目指したものである．

小笠原らの研究[5]は，キャッシュのリプレースメント方式として，論理スレッド番号を用いるLTN(Logical Thread Number)方式を提案している．この方式では論理スレッド番号に着目しスレッドのリプレース可能なウェイを制限し，スレッド間の干渉低減を目指している．さらに論文[6]では，キャッシュのリプレース方式を破壊的干渉の発生率により，擬似LRU方式からLTN方式動的に切り替える方式を提案し，破壊的干渉の削減を目指している．

4. 実装に向けた検討

本章では，OROCHIプロセッサを実装する上で検討を行う，性能と面積のトレードオフとなる点についてまとめる．

OROCHIプロセッサの最終的な評価を行うにあたり，比較対象は汎用プロセッサ＋アプリケーションプロセッサの環境である．この環境と比較し，OROCHIプロセッサにおける一番大きな違いは，フェッチ部はARMフェッチエンジンとFR-Vフェッチエンジンに分かれているが，バックエンドエンジンはVLIW型で1つしかないことである．

バックエンドエンジン及びデータキャッシュメモリはそれぞれのスレッドで共有して使用するため，同一ステージにそれぞれの命令が混在して実行されることとなる．しかし，各スレッドによって求められる性能用件も実行されるアプリケーションも異なる．アプリケーションスレッドはマルチメディア処理の実行を想定しているため，高いスループットを必要とし，汎用スレッドはOSなどの制御プログラムの実行を想定しているため，リアルタイムに割り込みに応答するレスポンスを必要とする．さらにスーパースカラプロセッサ上のSMT構成と比較し，バックエンドエンジンがIn-orderであるという点も異なっている．

これらのことから，以下のような性能と面積のトレードオフが考ええられ，OROCHIプロセッサを実装するにあたり詳細な検討と評価が必要である．

- スレッド間でのライン競合によるキャッシュヒット率の低下
  OROCHIプロセッサでは，データキャッシュはそれぞれのスレッドが共有して使用するため，2つのスレッドの間でのライン競合によるキャッシュミスの頻発で，性能低下が引き起こされる可能性がある．関連研究では，破壊的干渉が発生しやすいデータキャッシュの使用率が高いスレッドが複数実行される場合の評価を行っている．これらの研究におけるスレッドのモデルと，汎用スレッドとアプリケーションスレッドは特徴が異なっている．汎用スレッドは，制御プログラムの起動と割り込みに対する処理が実行され，これらのプログラムにおいては，キャッシュの使用頻度はそこまで高くない．一方，アプリケーションスレッドでは，動画処理などといった，データサイズが大きな処理を行うことを想定しているため，キャッシュの使用頻度は高くなると考えられる．
  この環境下で，どのようなキャッシュ構成で実装を行えば，キャッシュのヒット率をあげることが出来るか評価・検討を行う．
- 他のスレッドの影響によるパイプラインのストール
  バックエンドエンジン内は，In-orderで実行されており，汎用スレッド，アプリケーションスレッドのどちらかが何らかの要因でストールした場合，パイプライン全体がストールする．本来はそれぞれのスレッド間にはパイプラインストールによる影響はないはずであるが，OROCHIプロセッサにおいては，どちらかのスレッドでストールが発生した場合，ストールが発生していないスレッドにおいても影響を受け，ストールが解消するまで待ち続けなければならない．このような状況が頻発する場合，両スレッドの性能が共に大きく低下する可能性がある．そのような場合，ストールしていないスレッドを先のステージに進めるような制御を行うことが可能であれば，スレッド間の影響を減らすことができると考えられる．
  そのため現在のモデルを用い，どの程度他のスレッドのストールによる影響を受けているか調査を行うと共に，影響が大きい場合は，どのような制御が効率的か評価を行う．
- 汎用スレッドの実行頻度低下
  OROCHIプロセッサのモデルにおいて，汎用スレッドはアプリケーションスレッドが利用しない

表 1 osim パラメータ

A Cache	4Way, 16KB, line 64B ミスペナルティー 10cycle
V Cache	4Way, 16KB, line 64B ミスペナルティー 10cycle
L2 Cache	4Way, 2MB, line 64B ミスペナルティー 100cycle

隙間を利用して実行を行う．さらに，ハードウェア量の削減のため，命令キューにエンキューする際に制限を加えることを考えている．

具体的には，FR-V フェッチエンジンがフェッチした整数演算命令は内容・数にかかわらず，スロット番号の若い順から並列に実行できる命令数分エンキューする．さらに，ARM フェッチエンジンは演算の種類とスロットを 1 対 1 で対応付ける．そのため，FR-V スレッドが整数演算命令を発行し続ける限り ARM 命令でスロット 0 を利用できない．

このような制限下で汎用スレッドはどの程度の性能を得られるのか評価を行い，汎用スレッドの実行頻度が低い場合や汎用スレッドが待たされる時間が長く発生する場合，制限を緩和することで，汎用スレッドの実行頻度とハードウェア量のトレードオフを評価する．

また，制限の緩和を行った場合においても，アプリケーションスレッドが常に発行制限の最大まで整数演算命令を発行し続けた場合，演算器を占有し続ける可能性がある．その場合，リアルタイム性を保障する必要のある汎用スレッドをどの程度の頻度で実行しなければならないか評価を行い，その結果に基づき最低性能保障を行う．

- 命令キューのサイズ

  命令キューはバックエンドエンジンに対応したデコードされた信号を持つ．そのため，キューサイズを大きくする場合，多量のラッチが必要になるとともに，ARM フェッチエンジンからの命令のエンキュー機構も複雑化してしまう．逆にキューサイズが小さすぎると，ARM フェッチエンジンからの命令をエンキューできる可能性が小さくなってしまう．そのため，命令キューのサイズを変更してハードウェア量の変化と性能への影響の評価を行い最適な命令キューのサイズを確認する．

本稿ではこれらの中で，特にスレッド間でのライン競合によるキャッシュヒット率の低下と，他のスレッドの影響によるパイプラインのストールについて評価を行う．

5. 評　　価

評価項目の測定には，OROCHI モデルのプロセッサシミュレータ osim を用いた．両評価において共通のパラメータを表 1 に示す．

今回の評価では，OS と FR-V スレッドを利用するアプリケーションを並列に動作させる環境が整わなかったため，両スレッドを同時に開始し，アプリケーションスレッドが終了するまで計測を行う．

評価用アプリケーションとして，汎用スレッドに MiBench の bitcount を，アプリケーションスレッドに，Stanford ベンチマーク Queens を除く各種，MiBench の susan 各種，patricia，adpcm 各種を用いた．MiBench の bitcount はキャッシュのアクセス頻度が低いが，ある一定間隔でキャッシュに対して要求を出す特徴を持っており，今回の評価では，制御プログラムの代替として用いることとした．MiBench のデータサイズは，汎用スレッドは large を，アプリケーションスレッドは small を使用し，各スレッド間でデータの共有はない．汎用スレッドの bitcount においてデータサイズを large としたのは，評価を取るベンチマークすべてにおいて，汎用スレッドが終了する前にアプリケーションスレッドが終了するという条件をそろえるためである．bitcount においてはデータサイズによる，キャッシュのアクセスに影響を与えないことを事前に調べている．

5.1 スレッド間でのライン競合によるキャッシュヒット率の低下

5.1.1 評 価 環 境

評価を行ったデータキャッシの構成を表 2 に示す．基本的なダイレクトマップ，2Way/4Way/8Way セットアソシアティブで LRU によるリプレースを行う方式に加え，4Way セットアソシアティブ方式において，スレッドごとにリプレース可能なウェイを制限する方式についても評価を行う．リプレース可能なウェイの制限は，スレッドごとに 2Way ずつ固定である 2-2WAY 方式と，キャッシュの利用履歴に基づいて動的に変更する DYN 方式を osim 上に実装した．DYN 方式のパラメータとして，初期状態でそれぞれのスレッドでリプレース可能なウェイ数を 2 とし，過去 32 回のキャッシュアクセスの履歴を利用し，すべてのキャッシュアクセスがどちらかのスレッドである場合，そのスレッドがリプレース可能なウェイ数を 3，それ以外のスレッドがリプレース可能なウェイ数を 1 と変更する．そのような状況で，リプレース可能なウェイ数が 1 になっているスレッドが 16 回以上連続してキャッシュをアクセスし始めたとき，リプレース可能なウェイ数を初期

表 2 キャッシュ パラメータ

方式	説明
DIR	ダイレクトマップを共有
2WAY	2Way セットアソシアティブを共有
4WAY	4Way セットアソシアティブを共有
8WAY	8Way セットアソシアティブを共有
2-2WAY	4Way セットアソシアティブでスレッドごとにリプレース可能なラインをそれぞれ 2Way に制限
DYN	4Way セットアソシアティブでキャッシュの使用履歴に基づき，リプレース可能なラインを動的に変更

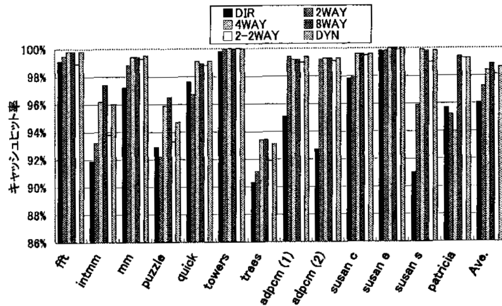

図 3　4KB データキャッシュヒット率

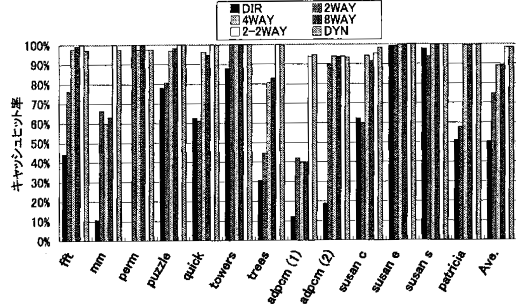

図 5　8KB 汎用スレッドデータキャッシュヒット率

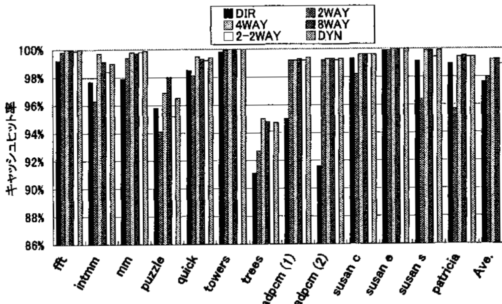

図 4　8KB データキャッシュヒット率

状態に戻すこととした．2-2WAY 方式でのキャッシュのリプレースメントはスレッドごとに 2Way の LRU により行い，DYN 方式では，4WAY の LRU で，割り当てられているリプレース可能なウェイの中から選択する方式を用いた．

キャッシュ容量はそれぞれの構成において 4KB と 8KB の場合のキャッシュヒット率を測定した．評価を行うにあたり，コールドミスを排除するため最初の 100 万サイクルをスキップした後，計測を開始し，計測終了はアプリケーションスレッド，汎用スレッドのどちらかの実行が終了した段階までとした．

5.1.2 評価結果

データキャッシュヒット率の結果をキャッシュ容量別に表 3，表 4 に示す．Stanford ベンチマークの Bubble と Perm はキャッシュヒット率がどの条件においても 99.9%以上であったため割愛している．

評価を行ったほとんどのベンチマークにおいて，4WAY 及び 8WAY のヒット率が良いという結果を得た．4WAY と 8WAY で比較を行った場合，キャッシュ容量が 4KB の場合においては，8WAY のほうが多少は良い結果が出ているが，8KB の場合はほとんどのベンチマークで差が無い結果となっており，8WAY ほど高い連想度は必要ないと考えられる．

また，キャッシュのリプレースを制限する 2-2WAY 方式では，キャッシュの容量が半分になることによる影響が，キャッシュの破壊的干渉を抑えることによる効果を超えなかったことにより，4WAY 方式に比較しキャッシュヒット率が低下した．また，DYN 方式においては，4KB, 8KB いずれの場合においてキャッシュヒット率は，4WAY より多少低下することが分かった．実験中，ほとんどのベンチマークにおいて，アプリケーションスレッドに 3Way 割り当てられたが，汎用スレッドに割り当てられた 1Way が，あまり効率的に利用できなかったのではないかと考えられる．

これらのことからシステムに実装する上で全体的なパフォーマンスを考えた場合，一番面積・性能の効率が良いと考えられるのは 4Way セットアソシアティブの構成である．

しかし，汎用スレッドのキャッシュミス率に注目すると，表 5 に示すように，2-2WAY 方式，DYN 方式の両方の方式において，4WAY 方式と比較して大幅に改善がみられる．特に，8KB のキャッシュ容量がある場合，実装が簡単な 2-2WAY 方式において全体性能も 4WAY 方式に近づいており，割り込みレスポンスの向上など，汎用スレッドの性能を上げる必要が出てきた場合は，再度，2-2WAY 方式と 4WAY 方式で評価を行う必要があると考えられる．

5.2 他のスレッドの影響によるパイプラインのストール

5.2.1 評価環境

VLIW バックエンドエンジンにおいてパイプラインのストールが発生する要因は，レジスタのフォワード待ちとキャッシュミスによるものである．

レジスタのフォワード待ちに着目した場合，FR550 命令セット及び ARM 命令セットの整数演算においては，乗算及び除算命令の発生頻度が極めて低いということにより，ほとんどのレジスタのフォワード待ちはキャッシュアクセスによって発生している．またメディア演算においても同様のレジスタのフォワード待ちは発生するが，性能を得るためにハンドコーディングを行うことから，人為的にレジスタのフォワード待ちが発生するのを防ぐことができるため考慮していない．

今回の評価では，キャッシュミス時のパイプライン

表 3 キャッシュミスに対するストールの頻度

	Stalls アプリ (回)	Stalls 汎用 (回)	Stalls 合計 (回)	L1 ミス (回)	Stalls rate (%)
bubble	5	26	31	59	52.5%
fft	175	81	256	940	27.2%
intmm	127	279	406	3039	13.4%
mm	2687	1840	4527	16115	28.1%
perm	50	6	56	268	20.9%
puzzle	199	3668	3867	37269	10.4%
quick	179	372	551	1727	31.9%
towers	7	7	14	40	35.0%
trees	846	30463	31309	53517	58.5%
adpcm(1)	8047	2439	10486	55526	18.9%
adpcm(2)	5702	6588	12290	42817	28.7%
susan c	64	6199	6263	18976	33.0%
susan e	224	6549	6773	20883	32.4%
susan s	74	11897	11971	31739	37.7%
patricia	73389	542997	616386	1236901	49.8%
Ave.					31.8%

ストールについて着目し，キャッシュミスが発生したスレッドと同一ステージに他のスレッドの命令が入っている場合がどの程度あるか評価を行う．

評価は，キャッシュミスが発生した際，どちらのスレッドによって発生したキャッシュミスかを判別し，キャッシュミスしたスレッド以外の命令が同一ステージ内に存在した回数の計測を測定するルーチンを osim 内に組み込み，どの程度の割合で片方のスレッドにおけるキャッシュミスが相手のスレッドを巻き込んでいるか評価を行った．

データキャッシュには 8KB,4Way セットアソシアティブの LRU を用い，アプリケーションスレッドと汎用スレッドを同時に開始した時点から計測を開始し，計測終了はアプリケーションスレッド，汎用スレッドのどちらかの実行が終了した段階までとした．

5.3 評価結果

結果を表 3 に示す．表中の Stalls アプリはアプリケーションスレッドのキャッシュミス時に同一ステージに汎用命令が入っていた回数で，Stalls 汎用は汎用スレッドのキャッシュミス時に同一ステージにアプリケーション命令が入っていた回数である．さらにそれらを足し合わせたものが Stalls 合計で，Stalls-Rate は全キャッシュミスにおいて，どの程度の割合で他のスレッドを巻き込んだパイプラインのストールが起こるかを示している．

この結果から，平均で 31.8%のキャッシュミスにおいて，同一ステージに他のスロットの命令が存在しているということが分かる．しかし，アプリケーションごとにかなりばらつきが大きく，adpcm(1) では全キャッシュミス中 18.9% 程度の影響しかなかった．一方，trees では 58.5%という半分以上のキャッシュミスで他のスレッドを巻き込んでいる．

しかし，これらの命令の組み合わせは，プログラム実行タイミングが異なった場合は異なる結果が出る可能性が考えられ，引き続き条件を変えた検討が必要であると考えられる．

6. おわりに

本稿では，OROCHI プロセッサを実装する上での性能と面積のトレードオフを述べ，特徴の異なるスレッド間の競合によるキャッシュヒット率への影響をキャッシュ容量及び構成を変えて調べ，4Way セットアソシアティブを用いたキャッシュの効率が良いことを示した．また，片方のスレッドにおけるパイプラインのストールが，同時に実行されている，相手のスレッドも巻き込み，性能の低下が発生する可能性があり，今回，片方のスレッドのキャッシュミスが相手のスレッドをどの程度の割合で巻き込んでいるか評価を行った．今回用いたベンチマークにおいては平均で 31.8%のキャッシュミスで相手のスレッドを巻き込んでいるという結果を得たが，プログラムの実行タイミングが異なった場合，異なる結果が出る可能性があるため，今後，条件を変えた検討が必要である．今後，今回挙げたトレードオフで評価を行わなかった項目についても，順次評価を行いたいと考えている．

謝辞 本研究は，半導体理工学研究センターとの共同研究によるものである．

参考文献

1) 島田貴史，田端猛一，北村俊明: 異種命令セット同時実行プロセッサ OROCHI の構成, 情報処理学会研究会報告, Vol.2006-ARC-170, pp.77–82 (2006)

2) H.himada, T.Shimada, T.Tabata, T.Kojima, K.Kise, Y.Nakashima, T.Kitamura : "Outline of OROCHI: A Fltiple Instruction Set Executable SMT Processor", Proceedings of International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, 2007(2007)

3) 中島康彦: ARM アーキテクチャ向け命令分解型スーパースカラ，情報処理学会研究会報告, Vol.2006-ARC-168, pp.55–60 (2006)

4) 山崎真也，本多弘樹，弓場敏嗣: マルチスレッドアーキテクチャにおけるデータキャッシュ構成方式の提案，情報処理学会研究会報告，Vol.1998-HPC-93，pp.79–85 (1998)

5) 小笠原嘉泰，佐藤未来子，笹田耕一，内倉要，並木美太郎，中條拓伯: SMT プロセッサ向けキャッシュメモリリプレース方式, 情報処理学会論文誌, Vol. 47,No.SIG 12(ACS 15),pp.119–132 (2006)

6) 小笠原嘉泰，佐藤未来子，並木美太郎，中條拓伯: SMT プロセッサにおけるキャッシュメモリリプレース方式の動的切り替え, 情報処理学会研究会報告, Vol.2006-ARC-170, pp.97–102 (2006)