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

2008－ARC－177 (13)
2008－HPC－114 (13)
2008／3／5

異種命令混在実行プロセッサにおけるプロセススケジューリング手法

山原幹雄† 中田尚† 中島康彦†

近年の組み込み機器では，高度なマルチメディア処理性能が要求されており，性能向上の為に汎用プロセッサとは別にDSPなどの専用プロセッサを搭載している．しかし，複数のプロセッサを搭載する事は省電力の点において短所となる．そこで，我々はマルチスレッド実行を拡張して1つのプロセッサで異種命令セットを同時に混在実行するプロセッサOROCHIを提案している．OROCHIではメディアアプリケーションのQoS(Quality of Service)保障が重要な課題である．混在実行中の他のアプリケーションの予期しないキャッシュミスの発生により，メディアアプリケーションの実行が阻害されることがあるからである．従来は，ハードウェアでキャッシュを均等に分割し，互いに干渉させない方法が提案されてきたが，ハードウェアでの解決は回路コストの増加を招く．そこで本研究ではOSスケジューラによる方法を提案する．提案手法をμClinuxに実装しMiBenchで評価した結果，従来のスケジューラと比較して優先アプリケーションのIPCが6.3%向上した．

The process scheduling method for an SMT processor OROCHI

Mikio YAMAHARA,† Takashi NAKADA† and Yasuhiko NAKASHIMA†

Recently, embedded equipments are required to execute advanced multimedia applications. A general-purpose processor equipped with DSPs can achieve high performance. However, such multiple processor structure leads to power consumption problem. Thus this paper proposes a heterogeneous SMT processor OROCHI that can execute mixed instruction sets simultaneously. It is an important problem that QoS (Quality of Service) of the media application cannot be guaranteed in the OROCHI with straightforward ways. Unexpected cache misses of other applications interferes against the multimedia application. Previous work proposed some modifications of the hardware, for example, fair cache allocation or cache miss prediction. However,these solution leads to increasing hardware cost. This paper, proposes a solution by enhancing scheduling algorithm. The scheduler fluctuates the CPU allocation of the other application to decrease cache misses, and guarantees the QoS of the media application. This paper implemented the mechanism in μClinux, and evaluated using MiBench benchmark programs. As a result, it is found that the media applications gain by 6.3% in contrast to the conventional solution.

1. はじめに

近年，携帯電話などの組み込み機器では，命令レベル並列度を期待できないOSなどの制御プログラムと，高い命令レベル並列度を期待できるマルチメディア処理を行うプログラムを同時に実行する環境が一般的になっている．このような状況において，最近では，汎用プロセッサコアの他に，用途に応じた複数種類のDSP(Digital Signal Processor)コアを併置するような構成をとるプロセッサが登場してきている．このような構成のプロセッサが登場してきたのは，OSを含む様々なソフトウェア資産の有効活用や，命令レベル並列度が高い一部のアプリケーションについては専用ハードウェアを活用して高速化と低電力化を図るためである．多くの場合，各プロセッサごとにOSを用意して動作させる方法や，DSPをコプロセッサとして用いる方法が採用される．しかしこれらは一長一短がある．前者の場合は，個々のプロセッサが独立して動作する事ができ，過去のソフトウェア資産を有効活用できる利点がある．しかし，そもそもOSを2つ用意する事は機能的に冗長であり，省電力の点からみても欠点と言える．また後者については，OSが1つで済み機能が冗長にならない点において有効な手段である．しかし，それぞれが独立して動作する事はできず，ソフトウェア資産もそのまま活用する事はできない．さらにどちらの場合にも言える事は，コアを併置する場合，回路規模が増大し，消費電力の増加，コア間での調停機構などが必要になる事である．以上から，理想的には過去のソフトウェア資産をそのまま活用でき，

† 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

かつハードウェアやOSなどの機能は1つに集約できる事が望ましい．

我々は，異種命令セットを同時に混在実行するプロセッサ OROCHI[1),2)] を提案している．OROCHI では OS 向けなどの汎用命令セットとマルチメディア処理用の命令セットの各々に対応するフロントエンドと VLIW 型の命令キューを備えたバックエンドで構成されている．フロントエンドでは例えば ARM[3)] と FR-V[4)] といった組み合わせを用意する事で異種命令を混在実行する．VLIW 型バックエンドはスーパースカラのような複雑な命令発行機能を必要としないため，低消費電力化や動作周波数の向上が期待できる．ARM 命令セットを VLIW 型に変換して実行する事については先行研究[5)] により性能低下はわずかであり，動作周波数の向上により全体性能は向上することが明らかになっている．

異種命令混在実行ではメディアアプリケーションを優先的にスケジューリングするが，これだけではメディアアプリケーションの QoS(Quality of Service) を保障できない場合がある．これは予期しないキャッシュミスの発生に起因する．そこで，本研究では OS によるメディアアプリケーションの QoS を保証する手法について提案する．

本論文では，第2章で異種命令混在実行方式を採用しているプロセッサ “OROCHI” について説明する．第3章で提案手法ついて述べた後．第4章で評価，最後にまとめとする．

2. OROCHI の構成

本章では異種命令混在実行プロセッサである OROCHI について説明する．

2.1 概　　要

OROCHI は同時マルチスレッド (Simultaneous Multithreading:SMT) を拡張したモデルを適用したプロセッサである．SMT を構成する2種類のスレッドは汎用プロセッサの命令を実行する汎用スレッド，メディア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である．

汎用スレッドには ARMv4 命令セット (以下，ARM 命令) を採用する．特権モードを含めた基本的な命令セットを実装し，μCLinux[6)] を制御プログラムとして実行する．ソフトウェア割り込みや外部割り込みはすべて汎用スレッドで行う．そのため，メディアアプリケーション側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はすべて汎用スレッドに処理を委託する．

メディアアプリケーションスレッドには FR550 命令セット (以下，FR-V 命令) を採用する．FR550 は 32 ビット長整数命令，単精度浮動小数点演算命令，メディア処理演算命令の最大 8 命令同時実行のメカニズムを持つ高性能 VLIW プロセッサである．この命令セットのうち，整数演算とメディア処理演算命令のサブセットを本研究ではメディアアプリケーションスレッドの命令セットとして採用する．

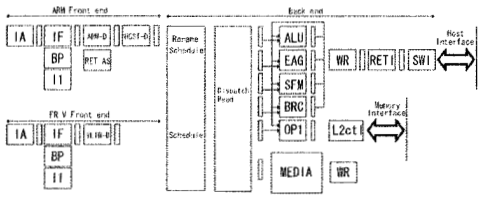

図1　OROCHI 概要

OROCHI のパイプラインモデルを図 **1** に示す．各々の命令セットに対してのフロントエンドと，VLIW 型演算器を持つバックエンドから構成される．ARM 命令により記述された過去のソフトウェア資産をスーパースカラ方式により高速実行し，コンパイラ等によりスケジューリングされた FR-V 命令を VLIW 方式により効率よく実行する．レジスタと演算器からなるバックエンド部分を共有する事により全体の回路規模を小さくするとともに，レジスタリネーミング機構やリタイア機構はスーパースカラ部分のみに実装し，マルチメディア命令を実行する際には，パイプライン段数の少ない VLIW 部分のみの稼働により消費電力の抑制をはかっている．

2.1.1 フロントエンド

命令フェッチからレジスタ読み出しの直前までは，ARM 命令と FR-V 命令のそれぞれに対してパイプラインステージが必要である．以降では各々を ARM フロントエンド，FR-V フロントエンドとする．バックエンドは VLIW 型であるため，命令列が1サイクルで実行可能な単純な命令 (ALU 演算，シフト演算，単純なロード／ストア演算) のみを取り扱う必要がある．ARM 命令はマルチプルロード／ストア命令や第2オペランド内にシフト操作を指定する事により，一般的な RISC 命令は複数命令かかるところを 1 命令に記述できるなど，複雑な機能を 1 命令に記述可能な CISC 型命令といえる．共通の命令キューと演算器により実行するためには，命令を単純な形に分解する必要がある[7),8)]．ARM フロントエンドの各パイプラインステージは次のような構成となる．まず，IA ステージではプログラムカウンタを生成する．IF ステージでは命令キャッシュを読み出すと同時に分岐予測を行う．その後，ARM-D，HOST-D ステージにおいて命令分解を行う．一方，FR-V 命令は ARM フロントエンドと同様に IA，IF ステージを設け，VLIW-D ステージにおいて VLIW 型命令のデコードを行うが，既に単純な命令となっているので，命令の分解は行う必要はない．

2.1.2 バックエンド

OROCHI のバックエンドは各命令共通であり，依存関係の解消，命令スケジューリングを行う Rename/Schedule ステージ，各演算を行う Execute ステージ，命令発行を行う Dispatch/Read ステージからなる．ARM フロントエンドで単純な内部命令に分解された ARM 命令は，Rename/schedule ステージにおいて命令間の依存関係を検査し，必要があればレジスタリネーミングされる，その結果を用いてスケジューリング機構が，VLIW 型命令キューの適切な位置に命令をキューイングする．FR-V 命令に対してはコンパイラ等ですでに依存関係は解消しているため，レジスタリネーミングは行なわず，そのまま命令キューにキューイングする．演算機は ALU，シフト演算を行なう SFM，アドレス計算と乗算の補助を実行する EAG，ロード／ストアを実行する OP1 から構成される．これらの演算器は 1 サイクルで実行される．また，EAG，SFM，ALU の演算結果は各演算器にバイパスする機能を設けている．ただし，OP1 からのバイパスはキャッシュミスによって結果を反映できない場合があるため，設けていない．

2.2 VLIW 型命令キュー

VLIW 型命令キューは演算器ごとに別れており，各エントリはデコーダ側から演算器側に向かうシフトレジスタである．命令発行は常に最右端からのみ行われる．そのため，複雑な命令発行機構は必要とせず，動作周波数や回路面積において有利である．一方，ARM 命令のキューイングは任意の位置に可能であり，依存関係がなければ，既に命令が入っている位置よりも右側，すなわち命令を追い越してキューイングすることができる．このようにスケジューリング時に命令の順序を動的に並び替えることができるため，スーパースカラ方式と同等の命令レベル並列度が達成できる．ロード命令がキャッシュミスを起こした場合は，後続命令がロードユニットを使用するか，ロード結果を使用するまで命令発行を継続できる．またロード命令とロード結果を使用する命令を離してスケジューリングすることにより，影響の軽減が可能である．同一サイクルでスケジューリングされる命令間に依存関係がある場合は命令順に先頭命令から見て後方となるようスケジューリングすることが必要である．このために必要な情報は，従来型スーパースカラもレジスタリネーミング機構が生成しているため，新たな機構はない．

VLIW 型命令は 1 ワードに含まれる複数の命令が同時に命令キューの左端に同一列に格納され，演算器の入力バッファのエントリに向かって右シフトされていく．左端のエントリが空いていない場合はそのまま待機する．

2.3 課 題 点

FR-V 命令において問題点となるのは前述の VLIW 型命令キューにある．VLIW 型であるため，命令は

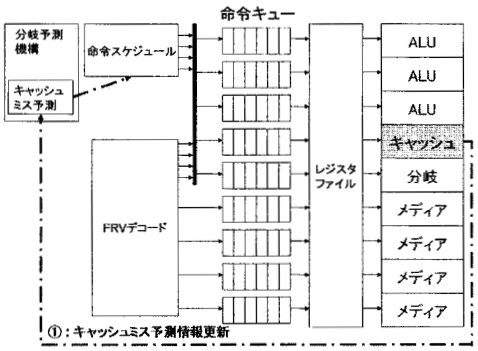

図 2 ハードウェアにおける解決方法

キューの右端一列が揃って発行される．仮に，ARM 命令のロード命令が予期せぬキャッシュミスを起こし，後続命令がロード結果を使用する場合を考える．キャッシュミスをしたデータをメモリからキャッシュへ到達するまで，後続命令を発行することはできない．そうなれば同時に発行される FR-V 命令もストールする事になる．従って FR-V 命令の性能が ARM 側のキャッシュミスの影響を受け低下してしまう．これを解決するためのハードウェアによる手段としては以下の 2 つが考えられる．

(1) プロセス毎にキャッシュの割り当てを分割する
(2) キャッシュミスの発生を予測する

上記のうち，(1) についてはキャッシュ分割によりお互いのキャッシュミスによる悪影響を減らすことができるが，各プロセスが利用できるキャッシュ容量が半減するため，更なるキャッシュミスを誘発してしまうため本研究では考慮しない．以降では (2) のキャッシュミスの発生を予測する場合の動作について説明する．

図 2 にハードウェアによる解決方法を示す．キャッシュミスを動的に予測することができれば，依存する命令をあらかじめ離して命令キューに投入することにより，キャッシュミスが原因のストールの発生を回避する (図 2 中の①)．ARM 命令は追い越しスケジューリングが可能であるため，キューの中の空いた部分には後続の命令が入ることができる．これにより，キャッシュミスが発生した場合にも依存命令は離れて配置されているため，命令キューのストールは発生せず，その後依存命令が右端に達したときにはキャッシュアクセスは完了しているので，ペナルティ無く実行を行われることが期待できる．ここで，ロード命令実行時のキャッシュミスを予測することが目的であるため，予測対象は 1 次データキャッシュとなる．キャッシュミス予測機構分岐を予測機構と類似の構造にすることにより，容易に実装を行うことができる．

しかし，ハードウェアでの解決は，回路の増大を招き柔軟性も低下するという問題点が挙げられる．そこで，本研究ではハードウェア構成は何も変更しない場

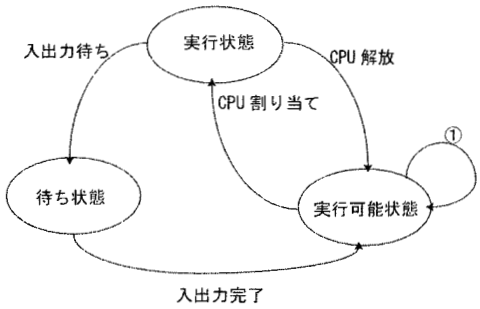

図 3　提案手法の状態遷移

合と，キャッシュミス予測機構を搭載した場合の両者に対して，提案するスケジューリング手法を用いてメディア命令の QoS 保障を目的とする．

3. 提案手法

FR-V 命令の性能が落ちる要因となるのは ARM 命令のキャッシュミスに起因していることは 2.3 節で述べた．キャッシュミスが頻繁に発生するほど，その影響は大きくなると考えられる．そこでキャッシュミスの回数をできる限り抑えるために，ARM 命令の CPU 時間を意図的に変更することで対処する．すなわち，FR-V 命令に並走している ARM 命令の性能をある一定のレベルまで下げる事でキャッシュミスの発生を抑え，FR-V 命令の性能を保障する ARM 命令の性能の変動は OS スケジューラで行う．提案手法では，ARM 命令の性能を落とすために，スケジューリングの際の CPU の割り当て割合を変化させる．スケジューラはタイマーなどの割込みの発生時に実行され，次に実行すべきプロセスを決定している．従来の方法では，プロセスが実行可能状態にあるとすると．スケジューラによって CPU が割り当てられると実行状態に遷移し，プロセス自身の持つ CPU 時間を使い切る，あるいは入出力待ちで待ち状態になるまで実行される．実行可能状態に遷移すると，再びスケジューラがそのプロセスを選択するまで実行可能状態でいる．通常の OS では，同じプロセスへの CPU の割り当てが特定回数続く場合，そのプロセスへの CPU の割り当てを一旦行わず，実行可能状態にする．提案手法の状態遷移を図 **3** に示す．スケジューリングをあえて回避させることで，ARM 命令を一時停止させている (図 3 中の①)．ただし，カーネルタスクは必ず動いており，あくまで停止するのはアプリケーションのプロセスだけである．また，その他のスケジューリングポリシーには変更は加えず既存の方法で行う．カーネルは実行するプロセスが無いため，arch_idle 関数に入り，アイドルループを繰り返す．次のタイマーの割り込みが発生すると，スケジューラは適切なプロセスを選択し，実行を再開する．このとき，どれくらいの頻度で CPU の割り当てを止めるかで性能の低下率を制御可能である．例えば，スケジューリングが 5 回に 1 回の割り合いで CPU を割り当てないとすると，ARM 命令の性能は 5 分の 4，すなわち 80%となる．

表 1　ベンチマークプログラム

	ARM	FR-V
Dijkstra&FFT	Dijkstra (small)	FFT
Quicksort&FFT	Quicksort (small)	FFT
SHA&FFT	SHA (small)	FFT

表 2　シミュレーション条件

分岐予測 (gshare)	pht : 2bit ×8K entry(gshare)
RET-AS	8entry
物理レジスタ	32entry
ストアバッファ	8entry
キャッシュライン	64byte
ARM I1 キャッシュ	4way, 16Kbyte
FR-V I1 キャッシュ	4way, 16Kbyte
D1 キャッシュ	4way, 32Kbyte
共用 L2 キャッシュ	4way, 2Mbyte

4. 評　価

提案するスケジューリング手法と各命令の混在実行機能を μClinux に実装し，ARM 命令と FR-V 命令を混在実行するパイプラインシミュレータで評価を行った．シミュレータではキャッシュミス予測器の有無の組み合わせとした．今回は，通常の OS スケジューラでの実行した時の ARM 命令の性能を 100%とし，以下 80%，60%，40%，20%の 5 段階で変化させている．スケジューリングにより ARM 性能をそれぞれに変化させた場合の ARM 命令，FR-V 命令の IPC とキャッシュミスの回数を測定した．データの取得範囲は混在実行を開始してから 1 億サイクル経過時までとし，これを 50 万サイクル間隔に分割し，それぞれの区間の平均 IPC とキャッシュミス回数を測定しその遷移を比較対象とする．

4.1 評価環境

評価に用いたベンチマークプログラムは MiBench の中から採用した．用いたベンチマークと組み合わせを表 **1** に示す．μClinux の扱えるファイル容量の制限から使用するベンチマークのサイズはすべて small とした．ただし，それでは実行時間が短いので，実行時間を十分に確保するために，内部で無限にループさせている．コンパイルオプションは ARM 命令のロードモジュールでは (-O2, -msoft-float) とする．FR-V 命令のロードモジュールでは (-msoft-float mcpu=fr550 -O2) とする．シミュレータの実行環境は，FreeBSD6.2R，XEON3.8GHz，2 次キャッシュ 1Mbyte，主記憶 2Gbyte とした．シミュレーション

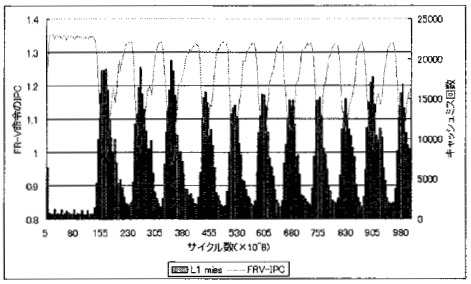

図 4　Dijkstra&FFT : ARM 性能 100%

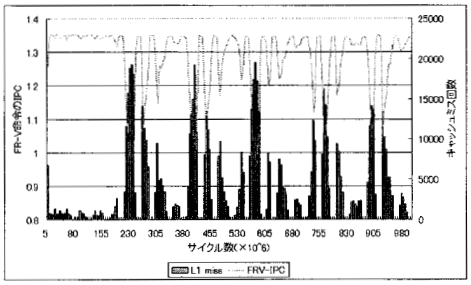

図 5　Dijkstra&FFT : ARM 性能 60%

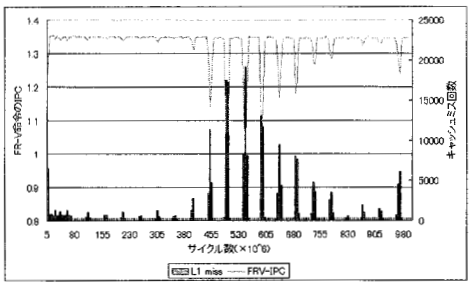

図 6　Dijkstra&FFT : ARM 性能 20%

表 3　ARM 性能 100%時の FR-V 命令の平均 IPC

Dijkstra&FFT	Quicksort&FFT	SHA&FFT
1.21	1.22	1.34

条件を，表 **2** に示す．

4.2　評価結果

4.2.1　キャッシュミスの影響

まず，ARM 命令が引き起こすキャッシュミスが FR-V 命令の実行にどれほど影響を与えているかを比較する．シミュレータはキャッシュミス予測器を搭載しない構成で行った．合わせて，ARM 命令の性能を変動させた場合のキャッシュミスの発生頻度も比較する．Dijkstra&FFT について ARM 命令の性能を変動させたときの FR-V 命令の IPC とキャッシュミス回数の測定結果のうち 100%，60%，20%の結果を図 **4** から図 **6** に示す．図中の棒グラフは単位時間あたりのキャッシュミス回数を，折れ線グラフは FR-V 命令の IPC の遷移を表す．また，各ベンチマークごとの FR-V 命令の平均 IPC を表 **3** に，1 億サイクル経過時のキャッシュミスの回数を表 **4** に示す．Dijkstra&FFT の場合，初期化フェーズ以外の区間での FR-V 命令の IPC の最大値と最小値の差は 0.38 であった．これは IPC の最大値の約 28%に当たる．この変動と連動して，キャッシュミス回数が 2000 回程度の頻度の時と 15000 から 20000 回の頻度の時が交互に発生している．ARM 性能 60%や 20%の結果を見ると ARM 性能を変動させた場合では，ARM アプリケーションを停止させている区間はカーネルの動作による 10 数回程度のごくわずかなキャッシュミスしか発生しておらず，その間 FR-V 命令の IPC が上昇していることがわかる．この結果からキャッシュミスの原因は ARM アプリケーションであり，このキャッシュミスを削減することにより，FR-V 命令の QoS を保障することができることがわかる．ARM 性能を低下させるにつれ，キャッシュミスの発生の低い区間の割合が増加しており，表 4 にあるように，測定区間中で発生したキャッシュミス回数は減少している．一方，ARM アプリケーションが中断された間に，FR-V アプリケーションによって中断前のデータがキャッシュから追い出されてしまう場合があるため，一時的にキャッシュミスが単位区間で見れば増加する場合があることも確認できた．また，一度に割り当てられる CPU 時間は変化させていないため，一旦 ARM アプリケーションプロセスに CPU が割り当てられて実行を開始すると，特定量のキャッシュミスを引き起こしてしまう．すなわちミス回数の頻度は幅は減少するが，高さの変化はわずかである．FR-V 命令の IPC を高くかつ比較的安定した遷移をさせるには，単位区間あたりのキャッシュミスの回数の削減も必要と考えられる．そうなれば，プロセスの持つ CPU 時間の変化も行う必要があるため，Alexandra らの手法[9] と複合した方法，例えば FR-V 命令の IPC が低下している場合，ARM 命令の性能を 50%まで低下させ，それでも FR-V 命令の QoS が保障できない場合はプロセスの CPU 時間を削減するといった方法の検討も今後の課題である．

4.2.2　FR-V 命令の IPC による性能比較

キャッシュミス予測器を用い予測器を搭載したシミュレータを作成し，その IPC を測定した．ここで ARM 性能は 100%とした．キャッシュミス予測器を用いた時の FR-V 命令の IPC を基準に，キャッシュミス予測器を搭載せずに ARM 性能を変動させるスケジューラで実行した場合の FR-V 命令の IPC の比率を測定した．その結果を各ベンチマークごとに図 **7** に示す．予測器を搭載していない場合は ARM 性能を 20%にすると，キャッシュミス予測器を用いたハードウェアと

表 4　キャッシュミス回数

ARM 性能	Dijkstra&FFT[回]	Quicksort&FFT[回]	SHA&FFT[回]
100%	1437360	1911059	168323
80%	1092864	1589732	136218
60%	744817	1091710	101884
40%	466257	639713	77790
20%	216924	368281	53830

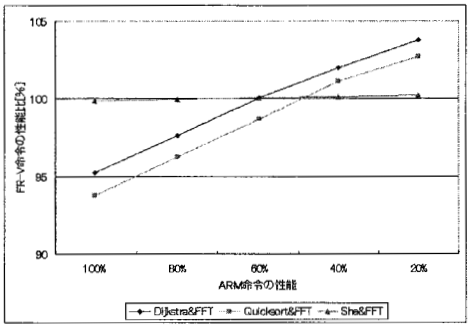

図 7　FR-V 命令の IPC 比

比較して FR-V 命令の IPC は平均で 2.27%向上することがわかる．特に Dijkstra&FFT では，3.8%の向上が見られた．また，ARM 命令の性能が 100%の場合と比較して，ARM 命令の性能を 20%とした場合の FR-V 命令の IPC は平均で 6.3%向上することが確認できた．特に Dijkstra&FFT では，9.5%の向上が見られた．SHA&FFT は向上率が 1%未満であったが，これはそもそも SHA のキャッシュミスの回数が他の 2 つに比べ非常に少ないため，ARM 性能を下げる効果が低かったと考えられる．また，いずれの場合においても ARM 性能を 40%まで低下させると，ハードウェアによる方法より FR-V 命令の IPC は高くなることが確認できた．

5. ま と め

本稿では，一般的な SMT を拡張して異種命令を同時に混在実行するプロセッサ OROCHI において，メディアアプリケーションの QoS を維持するためのスケジューリング手法について述べた．メディアアプリケーションに並走するアプリケーションのキャッシュミスの影響により，メディアアプリケーションの QoS 保障できない場合がある．そこでキャッシュミスの発生を抑制するために，優先する必要のないアプリケーションの CPU 割り当てを意図的に変動させる方法を提案した．μClinux に実装しパイプラインシミュレータを用いて提案手法の評価を行った．その結果，提案手法により単位時間あたりのキャッシュミス回数は抑制されることが確認できた．それに伴い，ARM 性能を 20%まで下げた場合には FR-V 命令の IPC が従来のスケジューラと比較して平均 6.3%向上する事を確認した．同様に，ハードウェアによる方法と比較して平均 2.27%向上する事を確認した．

6. 謝　　辞

本研究の一部は科学研究費補助金（基盤研究 (B) 課題番号 19300012），および，半導体理工学研究センターとの共同研究による．

参 考 文 献

1) 片岡晶人, 中西正樹, 山下茂, 中島康彦: VLIW 型命令キューを持つ OROCHI の命令スケジューリング機構, 情処研報 2007-ARC-172, pp. 25–30 (2007).
2) 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, 情処研報 2006-ARC-170, pp. 55–60 (2006).
3) ARM Limited: *ARM Architecture Reference Manual, ARM DDI0100E* (2000).
4) 富士通株式会社: *FR550 Series Instruction Set Manual Ver.1.1* (2002).
5) Shimada, H., Shimada, T., Tabata, T., Kojima, T., Kise, K., Nakashima, Y. and Kitamura, T.: Outline of OROCHI: A Multiple Instruction Set Executable SMT Processor, *IWIA* (2007).
6) : μClinux. http://www.uclinux.org/.
7) 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, 情処研報 2006-ARC-168, pp. 77–82 (2006).
8) 小島和也, 中島康彦: OROCHI 評価用集中命令ウィンドウ型スーパースカラの設計, 情処研報 2006-ARC-170, pp. 61–66 (2006).
9) Fedorova, A., Seltzer, M. and Smith, M. D.: Improving Performance Isolation on Chip Multiprocessors via an Operating System Scheduler, *16th International Conference on Parallel Architecture and Compilation Techniques*, pp. 25–38 (2007).