Vol.2016-ARC-219 No.14
Vol.2016-SLDM-175 No.14
Vol.2016-EMB-40 No.14
2016/3/24

# フロントエンド実行方式における エネルギー効率向上のための インオーダ実行モード切り替えアルゴリズム

地代 康政[†1]　出岡 宏二郎[†1]　塩谷 亮太[†1]　安藤 秀樹[†1]

**概要**：特性の異なる複数の実行系を単一コア内にそなえ，それらを使い分けることによってエネルギー効率を向上させる Tightly-Coupled Heterogeneous Cores（TCHCs）が提案されている．TCHCs の一つであるコンポジット・コアでは，インオーダとアウト・オブ・オーダのバックエンドを備えており，両者を切り替えて使用する．しかしコンポジット・コアはバックエンド切り替えペナルティや切り替えアルゴリズムの問題により，十分に消費エネルギーを削減できていない．この問題を解決するため，本研究では我々が提案してきた TCHC である Front-end Execution Architecture（FXA）をベースとして，低消費電力な実行モードを追加した Dual-Mode Front-end Execution Architecture（DM-FXA）と，そのためのモード切り替えアルゴリズムを提案する．提案手法を評価した結果，通常のアウト・オブ・オーダ・スーパスカラ・プロセッサと比較して 96.8%の性能を維持しつつ，平均 38.8%のエネルギー削減を達成した．

## 1. はじめに

特性の異なる複数の実行系を単一コア内にそなえ，それらを使い分けることによってエネルギー効率を向上させる **Tightly-Coupled Heterogeneous Cores（TCHCs）**が提案されている [4], [5], [6]．**コンポジット・コア** [4], [5] は，そのような TCHC の 1 つであり，インオーダとアウト・オブ・オーダの 2 つのバックエンドを持つ．以下ではこれらを **InO バックエンド**と **OoO バックエンド**と呼ぶ．コンポジット・コアでは，OoO バックエンド で実行した場合からの性能差が一定以内におさまるよう，OoO と InO バックエンドの切り替えを行いながら命令を実行する．これにより，コンポジット・コア ではアウト・オブ・オーダ・コアに近い性能を維持しながら消費エネルギーを削減することができる．

しかし コンポジット・コア には以下のような問題があり，インオーダ実行の機会を大きく損なっている：**P1) 切り替えペナルティが大きい．** これは コンポジット・コア ではバックエンドの切り替え時に，パイプライン内の全ての命令が実行されるのを待つ必要があるためである．**P2) 切り替えが局所最適となる．** コンポジット・コア の切り替えアルゴリズムでは，直近の将来の実行性能しか考慮しないため，ある程度遠い将来に現れるインオーダ実行すべき電力削減上より有益な区間を逃すことがある．**P3) 予測困難なイベントに対応できない．** InO バックエンドで実行中に L2 キャッシュ・ミスなどのイベントが発生した場合，OoO バックエンドと比べて大きな性能差が生じてしまう．このようなイベントは突発的に発生するため推定や予測が難しい上，一度発生した場合の影響が極めて大きい．

別の TCHC として，**Front-end Execution Architecture（FXA）** [6] がある．FXA は コンポジット・コア と同様に，インオーダ実行系（IXU: in-order execution unit）と，アウト・オブ・オーダ実行系（OXU: out-of-order execution unit）の 2 つの実行系を持つ．コンポジット・コア との違いは，IXU が OXU へのフィルタとして働く点である．IXU はプロセッサのフロントエンドに置かれ，ソース・オペランドが揃っており実行可能な命令をインオーダに実行する．このとき IXU で実行できた命令はパイプラインから取り除かれるため，OXU の消費エネルギーを削減することができる．FXA には切り替え実行の問題は存在しないものの，**P4) 消費エネルギーの削減が十分でない**．FXA では IXU によりフィルタできなかった命令を OXU で実行するために，リネーム・ユニットなどが常に稼働しており，それらが大きなエネルギーを消費し続けてしまう．

上記の P1〜4 までの問題を解決するため，本論文では FXA をベースとして低消費エネルギーな実行モードを追加した **Dual-Mode Front-end Execution Architecture（DM-FXA）**と，そのためのモード切り替えアルゴリズムを提案する．DM-FXA では OXU スリープ・モード と呼ぶ IXU のみで命令を実行する実行モードを追加し，適宜このモードへ実行を切り替える．OXU スリープ・モード ではアウト・オブ・オーダ実行に必要な処理を省略することでエネルギー消費を削減し，上記 P4 による FXA

[†1] 現在，名古屋大学大学院工学研究科
Presently with Graduate School of Engineering, Nagoya University

の問題を解決する．また，DM-FXA では 通常モードから OXU スリープ・モード へは切り替えペナルティが発生する一方で，OXU スリープ・モード から通常モードへはペナルティなく切り替えられるため，上記 P1 のレイテンシの問題を低減できる．

提案する DM-FXA のための切り替えアルゴリズムでは，ある程度将来を見越した切り替えを導入することで，上記 P2 の 切り替えが局所最適になる問題を解決する．また，大きく性能が低下するイベントの発生時に，即座に OXU スリープ・モード から脱出するイベント駆動の切り替えを導入する．イベント駆動であるため予測ミスは本質的に存在せず，上記 P3 の予測の困難さに起因する問題を解決する．

本論文の以降の構成は以下の通りである．2 節では既存研究について述べた後，3 節で コンポジット・コア の問題について述べる．4 節では提案する DM-FXA について説明し，5 節ではそのための切り替えアルゴリズムについて述べる．6 節では評価を行い，その後 7 節でまとめる．

## 2. 既存研究

本節では既存の TCHC として, コンポジット・コア [4], [5] と FXA[6] について説明する．

### 2.1 コンポジット・コア

図 1 にコンポジット・コアのブロック図を示す．以下ではコンポジット・コアのアーキテクチャ構成について説明し，その後切り替えアルゴリズムについて説明する．

#### 2.1.1 構成

コンポジット・コアは実行コアの高速な切り替えを実現するために，両バックエンド間でデコードまでのフロントエンドと，L1 キャッシュを共有する．各バックエンドは独立したレジスタ・ファイルをもつため，実行の切り替え時にはそれらのデータのマイグレーションが必要になる．

#### 2.1.2 バックエンド切り替え時の動作

実行バックエンドの移行時にはまずフェッチを停止し，アクティブなコアのパイプラインからの命令の排出を待つ．このとき，並行して投機的にレジスタ・ファイル内のデータを転送する．これらによる転送が終了すると，他方のバックエンドでの実行を開始できる．以上の動作に必要な時間が切り替え時のペナルティとなる．

#### 2.1.3 切り替えアルゴリズム

切り替えは，OoO バックエンドで実行した場合からの性能差が一定以内におさまるよう，適宜 InO バックエンドへと実行を移行する．性能低下の小さい区間でインオーダ実行することで，性能を維持しながらアウト・オブ・オーダ実行による消費エネルギーを削減する．切り替え判断は一定の命令数を実行する毎に行う．この一定の命令数，すなわち切り替え判断から次の切り替え判断までの区間を今後「命令区間」と呼ぶ．

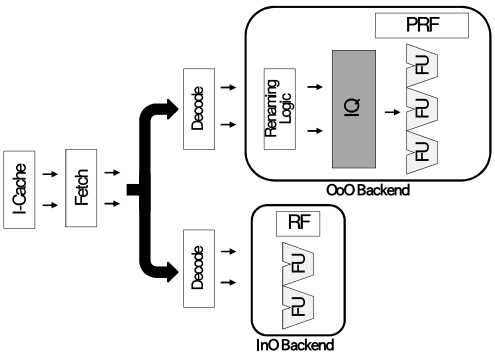

図 1: コンポジット・コア

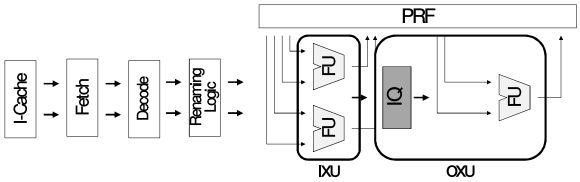

図 2: FXA

切り替え判断の動作を以下の 4 つの段階に分けて述べる．

(1) **両バックエンドにおける実行サイクル数測定及び推定：**実行を終えた命令区間について，OoO バックエンドと InO バックエンドのそれぞれで実行した場合にかかる実行サイクル数 $Cycle_{OoO}$ および $Cycle_{InO}$ を求める．実際に実行したバックエンドについては，その時の実行サイクル数を測定すればよい．使用されてないバックエンドでの実行サイクル数は推定によって導く．推定には線形回帰モデルを用い，実行中に得られるキャッシュ・ミス回数や分岐予測ミス回数などのメトリックから導出する．

(2) **目標とする総実行サイクル数の算出：**目標とする総実行サイクル数 $Cycle_{target}$ を算出する．全命令区間を OoO バックエンドで実行した場合の総実行サイクル数は，各命令区間の $Cycle_{OoO}$ の累積で得られる．この最小の総実行サイクル数と指定された性能低下許容量 $Slowdown$ から， $Cycle_{target}$ を求める（式（i））．

$$Cycle_{target} = \sum Cycle_{OoO} \times (1 + Slowdown) \quad \text{(i)}$$

(3) **閾値の計算：**目標総実行サイクル数と実際の総実行サイクル数の差 $Cycle_{error}$ から，バックエンドの選択判断に用いる閾値 $Cycle_{thr}$ を計算する．この閾値は，$Cycle_{target}$ を目標量， $Cycle_{actual}$ を制御量とした PI 制御のフィードバックとして式（ii）から導出される．ここで $K_p$ は比例定数である． 積分定数 $K_i$ にかかっているのは，過去の $Cycle_{error}$ の累計である．

$$Cycle_{thr} = K_p Cycle_{error} + K_i \sum Cycle_{error} \quad \text{(ii)}$$

(4) **バックエンドの選択：**次命令区間におけるバックエンド間の実行サイクル数の差が，閾値 $Cycle_{thr}$ より大きければ OoO バックエンドを使用し，そうでなければ InO バックエンドを使用する（式（iii））．実際には，次命令区間における実行サイクル数の差は得られないた

め，手順（1）で導出した $Cycle_{OoO}$ および $Cycle_{InO}$ のサイクル差を用いる．これには，直前の命令区間の実行性能と同様の特性を持った命令列が次も現れるという仮定がおかれている．コンポジット・コアの後続の研究 [5] では，トレース・ベースのフェーズ予測器を用いることで次区間の性能差を予測している．

$$\begin{cases} Cycle_{InO} - Cycle_{OoO} \geq Cycle_{thr} \\ \qquad Select\ OoO\ Backend \\ Cycle_{InO} - Cycle_{OoO} < Cycle_{thr} \\ \qquad Select\ InO\ Backend \end{cases} \tag{iii}$$

上記の切り替えによって，コンポジット・コアではインオーダ実行への切り替えを行うことで，総実行サイクル数の増加と引き換えに消費エネルギーを削減する．

ここで上記の切り替えアルゴリズムにおける閾値 $Cycle_{thr}$ の値は，ある種の余剰性能と見ることができる．これは，閾値が $Cycle_{error}$ をもとに導出されるためである（式（ii））．目標総実行サイクル数に対して現在の実行が短時間で実行できているほど閾値は大きくなり，InO バックエンドへ切り替えが行われやすくなる．

### 2.2 FXA

図 2 に FXA のブロック図を示す．以下では FXA の構成について説明した後，その基本的な動作について説明する．

#### 2.2.1 構成

FXA は以下の二つの実行系を持つ．

(1) **OXU**: 通常のスーパスカラ・プロセッサの実行コアと同様のものである．
(2) **IXU**: 主に演算器とそれらを接続するバイパス・ネットワークからなる．図 2 に示すように， IXU はフロントエンドのリネーム・ステージとディスパッチ・ステージの間に配置される．FXA では，物理レジスタ・ファイル（PRF: Physical Register File）読み出しステージは，OXU とは別にリネーム・ステージの後にも存在する．読み出されたソース・オペランドは IXU に入力される．

### 2.3 基本的な動作

IXU は OXU に対するフィルタとして働く．すなわち，IXU で実行された命令は OXU で実行されず，命令パイプラインから取り除かれる．IXU を用いた命令処理の流れを以下に記す．ここでは命令は全て 1 サイクルで実行できる整数演算命令とする．

(1) フロントエンドのレジスタ読み出しステージにおいて，PRF を読み出す．
(2) 命令がレディであるかを判断する．ここで「命令がレディ」とは，ソース・オペランドが全て得られ，実行可能であることを意味する．ソース・オペランドは PRF からの読み出しか，IXU 内の実行結果のバイパスによって得られる．
(3) 命令がレディかどうかに応じて，IXU では以下のように処理する：
  (a) レディな命令は IXU で実行し発行キュー（IQ: issue queue）にディスパッチしない．実行結果は IXU を出た後に PRF に書かれる．
  (b) レディでない命令は，NOP として IXU をそのまま通過する．その後，命令は IQ にディスパッチされ，実行される．

通常のインオーダ・プロセッサでは，レディでない命令のデコード時は，依存が解消されるまでパイプラインをストールさせる．これに対し IXU では，レディでない命令は NOP として通過し，パイプラインはストールされずに命令は流れ続ける．

FXA では IXU によって平均 5 割近くの命令が実行される [6]．これにより FXA では性能を落とすことなく IQ の消費エネルギーを削減できる．

## 3. コンポジット・コアの問題

本節ではコンポジット・コアのもつ 3 つの問題について述べる．

### 3.1 切り替えペナルティが大きい

コンポジット・コアは異なる 2 つのバックエンドを持つ構成である以上，それらの切り替えには必ずペナルティが発生する．バックエンドの切り替えのためには，パイプラインからの命令の排出と投機転送に失敗したレジスタ・ファイル内のデータの転送を待つ必要がある．

### 3.2 切り替えアルゴリズムの問題

コンポジット・コアの切り替えアルゴリズムは，切り替え判断の視野が短期的である点と，予測困難なイベントの発生に対応できない点で問題がある．

#### 3.2.1 長期的視野を持たない切り替え判断

コンポジット・コアのアルゴリズムは短期的な視点しか持たないため，切り替え判断は局所最適に陥る可能性がある．切り替え判断時に考慮されているのは次命令区間の実行サイクル数差のみである．この差が閾値よりも小さければインオーダ実行を選択する．

余剰性能を温存しない切り替え判断によって，遠い将来に得られるはずであった電力上有益な区間でのインオーダ実行の機会が失われる．このような切り替えの例を，図 3 を用いて説明する．図 3 は，gobmk ベンチマークをコンポジット・コアで切り替え実行した際の，閾値の変動と判断の様子を示した図である．横軸は実行時の命令区間数を示す．グラフの赤線は各命令区間におけるバックエンド間の実行サイクル数差を表し，青線は閾値を表す．背景が白い区間は OoO バックエンドで実行されたことを，灰色の区間は InO バックエンドで実行されたことを示す．コンポ

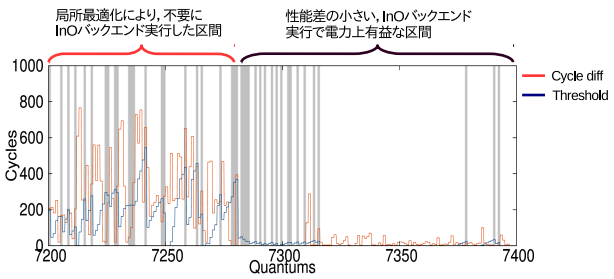


図 3: コンポジット・コアによる切り替え実行

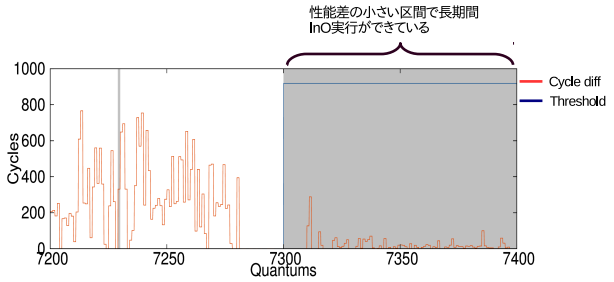


図 4: 粗粒度な閾値更新の導入

ジット・コアのアルゴリズムでは，図 3 上部の赤い括弧で示した区間のように，実行サイクル数差が大きい命令区間でも十分に余剰性能があれば InO バックエンドへ切り替える．余剰性能の大量消費により，図の後半の性能差の小さいフェーズでも十分に InO バックエンドを使用することができなくなっている．

#### 3.2.2 予測困難なイベントへの対応

コンポジット・コアのアルゴリズムは予測困難なイベントに対応できない．InO バックエンドで実行中に L2 キャッシュ・ミスなどのイベントが発生した場合，OoO バックエンドで実行した場合と比べて大きな性能差が生じる場合がある．推定や予測に基づく コンポジット・コア の切り替えアルゴリズムにおいては，このような突発的なイベントの発生は予測することが難しい．

文献 [5] では，トレース・ベースのフェーズ予測器によって次区間の性能差を予測し，切り替え判断の正確さを向上しているが，上記の問題は依然として大きい．誤ったインオーダ実行による余剰性能の大量消費は極力回避すべきであるが，予測のアプローチには本質的に限界がある．

## 4. Dual-Mode Front-end Execution Architecture

我々は，FXA をベースとして低消費エネルギーな実行モードを追加した **Dual-Mode Front-end Execution Architecture（DM-FXA）** を提案する．DM-FXA では OXU スリープ・モード と呼ぶ IXU のみで命令を実行するモードを追加し，モードを切り替えつつ実行する．区別のため，以降では従来の FXA の動作を **OXU アクティブ・モード**と呼ぶ．以下ではまず OXU スリープ・モード について説明し，その後モード間の切り替え動作を説明する．

### 4.1 OXU スリープ・モード

OXU スリープ・モード では，全ての命令を IXU のみでインオーダに実行する．OXU スリープ・モード では IXU でソース・オペランドが全て揃わず実行できない命令が現れた場合，パイプラインをストールさせて依存が解決されるのを待つ．これは通常のインオーダ・プロセッサの動作と同じである．OXU スリープ・モード での実行中に浮動小数点命令のような IXU で実行できない命令に遭遇した場合は，即座に OXU アクティブ・モード に移行する．

全ての命令を IXU のみでインオーダに実行するため，OXU スリープ・モード では OXU を含むアウト・オブ・オーダ実行に必要な処理を完全に省略する．このときは，次節以降で詳しく述べるように，レジスタ・リネームも省略し，論理レジスタ番号を用いて PRF へアクセスする．

### 4.2 モード切り替え

本節ではモード間の切り替えの動作と，それに伴って発生する切り替えペナルティについて説明する．

OXU アクティブ・モード から OXU スリープ・モード への移行は，以下の 2 つの操作を経て行われる．1) まず，コンポジット・コア の場合と同様に，IXU を停止し，既に OXU にある命令の実行終了を待つ． 2) 次に，論理レジスタ番号で PRF にアクセスができるよう， PRF 内のデータを論理レジスタ番号の順に並べ替える．モード切り替えの際は以上の 2 つの動作を終える必要があり，これが切り替え時のペナルティとなる．

OXU スリープ・モード から OXU アクティブ・モード への移行は，レジスタ・リネームなどのアウト・オブ・オーダ実行に必要な処理を単純に再開するのみである．このため，切り替えペナルティは存在しない．

## 5. 切り替えアルゴリズムの提案

本節では，DM-FXA のためのモード切替アルゴリズムを提案する．3 節で述べた コンポジット・コア の切り替えアルゴリズムをベースとし，そこに 1) 粗粒度な閾値の更新と，2) イベント駆動による強制切り替えを導入する．これにより，3.2 節で述べた問題を解決する．

提案アルゴリズムでは不適切なインオーダ実行を回避することで OXU スリープ・モード実行率の向上を図る．不適切なアウト・オブ・オーダ実行に比べ，不適切なインオーダ実行による悪影響は非常に大きい．不適切なアウト・オブ・オーダ実行のデメリットは 1 命令区間分の消費エネルギー削減の機会を失うのみであり，性能低下も起こらない．一方で不適切なインオーダ実行では，余剰サイクル数が大きく消費される．これは後に訪れる，余剰サイクル数の消費が少ない有益な区間を OXU スリープ・モード実行する機会を多数失うことを意味する．

### 5.1 粗粒度の閾値更新

2.1.3 節 で述べた コンポジット・コア の切り替えアルゴリズムにおいて，閾値の更新のみを粗粒度にする．具体的には，命令区間毎に切り替え判断を行うのに対し，閾値の算出は 100 程度の命令区間毎に行う．

図 4 は，切り替えアルゴリズムに粗粒度な閾値更新を導入し，図 3 と同一の区間を実行した例である．粗粒度な閾値更新では，余剰サイクル数が溜まっても閾値は変化しないので，性能差が大きい区間が連続した場合にもインオーダ実行は行われなくなる．余剰サイクル数の消費を防ぎ，後の性能差が小さい区間で存分にインオーダ実行ができている．このように粗粒度な閾値更新によって，ある程度遠い将来の性能変動を見越した切り替えが可能になる．

### 5.2 イベント駆動の切り替え

提案するアルゴリズムでは，OXU スリープ・モード実行で大きく性能低下を引き起こすイベントに注目し，イベント発生時に即座に OXU スリープ・モードを中止する．このようなイベント駆動の切り替えにより大幅な性能低下を確実に回避する．OXU アクティブ・モードへの切り替えにはペナルティがないため，DM-FXA はこのイベント駆動の切り替えに適している．

発生時に OXU スリープ・モード実行を中止するイベントとしては，1）L2 データ・キャッシュ・ミスと 2）IXU と相性の悪いループの 2 つがある．1）L2 データ・キャッシュ・ミスが発生した場合，OXU スリープ・モードではインオーダ実行であるため，アウト・オブ・オーダ実行と異なりキャッシュ・ミス・レイテンシを隠蔽できないため，大きな性能低下が生じる．2) IXU と相性の悪いループとは，データ依存関係に起因して OXU スリープ・モードでストールを発生するコードが繰り返し現れるループを指す．OXU アクティブ・モードでは IXU で依存関係が解決できない命令はストールせずに OXU に送られるため，このようなループ実行はモード間で性能差を生じる．

2)IXU と相性の悪いループの検出については，PC 情報を使うことで学習する．学習にはバックエッジ PC を用いる．PC の巻き戻りが発生した際の PC($PC_{now}$) と飛び先の PC($PC_{target}$) の組が，連続で現れる限りループ内にいると判断する．実装のために，テーブルを 1 つ使用する．このテーブルはバックエッジを検出した時の $PC_{now}$ をインデクスとする，3bit の飽和カウンタをエントリとしてもつ．エントリは全て $100_2$ で初期化されている．

IXU と相性の悪いループの検出動作について説明する．

(1) 現在の実行がループ内にいることを検出する．
(2) ループから外れた際 ($PC_{now}$ と $PC_{target}$ それぞれの一致が取れなかった場合)，脱出したループの $PC_{now}$ についてテーブルの学習を始める．
(3) ループから脱出した命令区間の，次の区間の両モードの実行サイクル数差 $Cycle_{OoO} - Cycle_{InO}$ があらかじめ定めた閾値以上であった場合，テーブルの $PC_{now}$ に対応するカウンタを 2 減算し，そうでなければ 1 加算する．評価においてはこの閾値は 150 である．
(4) 脱出したループの $PC_{now}$ に対応するエントリの最上位ビットが 0 であれば OXU アクティブ・モードへ移行する．またループからの脱出が発生した命令区間の次の区間も OXU アクティブ・モードで実行する．

表 1: 評価モデルの構成

| | BASE/CC(OoO) | FXA/DM-FXA |
|---|---|---|
| Fetch Width | 3 | ← |
| Issue Width | 4 | 2 |
| Retire Width | 3 | ← |
| IQ | 64 entries | 32 entries |
| Function Unit | ALU:2, FPU:2, MEM:2 | ← |
| Ld./St. Queue | 32/32 entries | ← |
| ROB | 128 entries | ← |
| L1 I-cache | 48KB, 2 cycles, | ← |
| L1 D-cache | 32KB, 2 cycles, | ← |
| L2 cache | 512KB, 12 cycles, | |
| Main Memory | 200 cycles | ← |
| IXU | N/A | width 3, depth 3 |

## 6. 評価

### 6.1 評価環境

シミュレーションにより提案手法を評価した．性能の評価には，鬼斬 [7] に提案手法を実装したシミュレータを用いた．評価では SPECCPU INT 2006 に含まれる全ベンチマーク・プログラムを用いた．入力セットには ref を使用し，プログラムの先頭 2G 命令をスキップした後の 100M 命令について測定した．消費エネルギーの評価には McPAT[3] を用い，22nm テクノロジを想定した．

上記の環境において，以下のモデルを実装し評価した．

(1) **BASE**: 通常のアウト・オブ・オーダ・スーパスカラ・プロセッサを実装したモデル．
(2) **FXA**: 従来の FXA を実装したモデル．
(3) **CC+PRED**: コンポジット・コア を理想化して実装したモデル．2.1.3 節 で述べた既存のアルゴリズムを理想化して適用しており，性能差予測が完璧に行われるとする．切り替え判断は 1000 命令毎に行うものとし，BASE から 5% の性能低下を許容する．
(4) **DMFXA+CGEV**: 提案する DM-FXA とその切り替えアルゴリズムによるモデル．モード切り替え判断は 1000 命令毎に行い，閾値の算出は 100 区間毎に行う．このモデルでは FXA から 10% の性能低下を許容する．FXA は BASE よりも平均で 6.8% 性能が高いため，このモデルは CC+PRED とほぼ同じ性能を持つ．

**表 1** にこれらの主な構成を示す．これらのパラメータは，ARM big.LITTLE を構成する Cortex A57[1]/A53[2] にあわせている．また，CC+PRED の InO バックエンドは，3-issue の インオーダ・スーパスカラである．

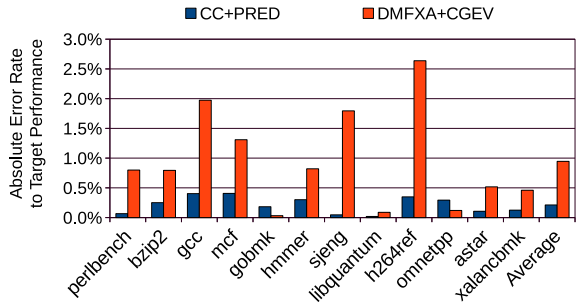


図 5: 目標性能からの絶対誤差率

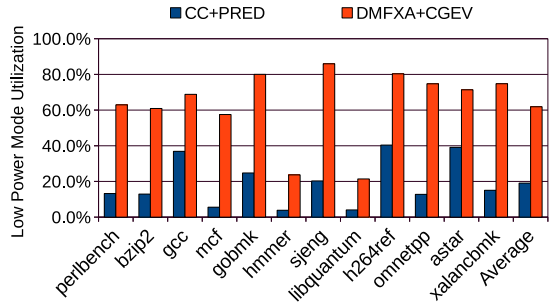


図 6: 低消費電力モードの使用率

### 6.2 評価結果

#### 6.2.1 切り替え制御の精度と性能

まず，各モデルにおける切り替え制御の精度を評価する．**図 5** に，各モデルにおける目標性能からの絶対誤差率とその平均を示す．CC+PRED における絶対誤差率は平均で 0.21%であり，高い精度で目標性能へ制御できている．DMFXA+CGEV の絶対誤差率は平均で 0.95% であり，CC+PRED よりは少し誤差が大きいものの十分な精度を達成している．これらによる制御の結果，BASE とくらべて幾何平均で CC+PRED は 95.2%，DMFXA+CGEV は 96.8% の性能となっている．

#### 6.2.2 スリープ・モード使用率

各モデルにおける全実行命令数内に占める低消費電力モードの使用率を**図 6** に示す．ここで低消費電力モードとは，CC+PRED では InO バックエンド による実行モード，DMFXA+CGEV では OXU スリープ・モード を意味する．CC+PRED の低消費電力モード使用率が平均 19.1% であるのに対し，提案手法である DMFXA+CGEV では 61.9% である．これは 4 節や 5 節で述べた DM-FXA やその切り替えアルゴリズムの効果による．

#### 6.2.3 消費エネルギー

**図 7** に各モデルにおける，BASE に対する相対消費エネルギーを示す．DMFXA+CGEV は CC+PRED と比較して平均で 19.3% 消費エネルギーを削減した．これは前述したように低消費電力モードの使用率が大幅に増えているためである．また BASE と比較して平均で 38.8%，FXA と比較して平均 27.4% 消費エネルギーを削減した．

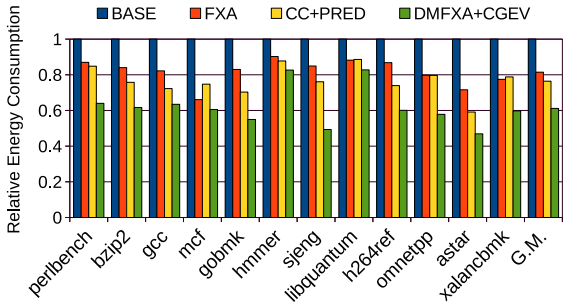


図 7: BASE に対する相対消費エネルギー

## 7. まとめ

単一のコア内に異種性を導入することで電力効率を向上させるアーキテクチャとして コンポジット・コア や FXA が提案されてきた．しかし，これらではアーキテクチャそのものやモード切り替えアルゴリズムに問題があり，十分な消費エネルギーの削減が行えない問題があった．これを解決するため，本論文では DM-FXA とそのための切り替えアルゴリズムを提案した．評価の結果，通常のスーパスカラ・プロセッサと比較して 3.2% の性能低下で 38.8% の消費エネルギーが削減できることを確かめた．

**謝辞** 本研究の一部は，日本学術振興会 科学研究費補助金 基盤研究 (C)（課題番号 25330057），および 若手研究 (A)（課題番号 24680005）による補助のもとで行われた．

### 参考文献

[1] Bolaria, J.: Cortex-A57 Extends ARM's Reach, *Microprocessor Report 11/5/12-1*, pp. 1–5 (2012).

[2] Krewell, K.: Cortex-A53 Is ARM's Next Little Thing, *Microprocessor Report 11/5/12-2*, pp. 1–4 (2012).

[3] Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M. and Jouppi, N. P.: McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures, *Proceedings of the 42nd Annual International Symposium on Microarchitecture*, pp. 469–480 (2009).

[4] Lukefahr, A., Padmanabha, S., Das, R., Sleiman, F. M., Dreslinski, R., Wenisch, T. F. and Mahlke, S.: Composite Cores: Pushing Heterogeneity Into a Core, *Proceedings of the 45th Annual International Symposium on Microarchitecture*, pp. 317–328 (2012).

[5] Padmanabha, S., Lukefahr, A., Das, R. and Mahlke, S.: Trace Based Phase Prediction for Tightly-coupled Heterogeneous Cores, *Proceedings of the 46th Annual International Symposium on Microarchitecture*, pp. 445–456 (2009).

[6] Shioya, R., Goshima, M. and Ando, H.: A Front-end Execution Architecture for High Energy Efficiancy, *Proceedings of the 47th Annual International Symposium on Microarchitecture*, pp. 419–431 (2014).

[7] 塩谷亮太，五島正裕，坂井修一：プロセッサ・シミュレータ「鬼斬弐」の設計と実装，先進的計算基盤システムシンポジウム SACSIS，pp. 120–121 (2009).