Vol.2017-ARC-226 No.17
2017/5/24

低電力モードを備えるプロセッサと モード切り替えアルゴリズムによる電力効率の向上

塩谷 亮太[1,a)]　地代 康政[1]　出岡 宏二郎[1]　五島 正裕[2]　安藤 秀樹[1]

概要：
Tightly-coupled heterogeneous core（TCHC）は，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事でエネルギー効率を向上させるアーキテクチャである．TCHC は典型的には低電力と高性能の二つの実行モードを備えており，プログラムのフェーズごとに両者を細粒度に切り替えて使用する．しかし，既存の TCHC では実行モードの替えペナルティや，切り替えアルゴリズムが局所最適となる問題により，十分に消費エネルギーを削減できていなかった．これに対し，本論文では dual-mode frontend execution architecture（DM-FXA）と呼ぶ新しい TCHC のアーキテクチャと，multi-scale アルゴリズム (MSA) と呼ぶ新しい切り替えアルゴリズムを提案する．これらのうち，DM-FXA ではパイプライン構造の工夫により実行モードの切り替えペナルティを削減する．また，MSA ではプログラムの粗粒度な振る舞いを利用することにより，従来のアルゴリズムにあった局所最適の問題を解決する．提案手法を評価した結果，通常の out-of-order 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

プロセッサのエネルギー効率を向上させる有力な方法として**ヘテロジニアス・マルチコア（HMC: heterogeneous multicore）**がある [2,6,7,9,13]．ARM big.LITTLE は商用化された HMC の例である [6]．HMC は一般に性能やエネルギー効率などの特性が異なる複数のコアからなり，プログラムのフェーズごとに最もエネルギー効率が良いコアに実行を切り替える．しかし HMC では各コアで独立したキャッシュや予測器を持つため，コア間でコンテキストを移動する際のオーバーヘッドが非常に大きい [11]．このため，コアの切り替えは 100M 命令程度の比較的長いインターバルで行われることが多い．

これに対し，切り替えのペナルティを減らすために**tightly-coupled heterogeneous core（TCHC）**が提案されている [11,12,15]．HMC とは異なり，TCHC は非対称な複数のバックエンドを持つシングル・コアである．これらのバックエンド間では典型的にはフロントエンドやキャッシュ，予測器などが共有される．TCHC ではプログラムのフェーズごとに，コアではなくこれらのバックエンドを切り替えながら実行を行う．TCHC では切り替えペナルティが小さいため，HMC と比べて非常に短いインターバル（たとえば 500 命令程度）でバックエンドを切り替えることができる．

代表的な TCHC として，**composite core（CC）**が提案されている [11,12]．CC は**図 1** に示すように out-of-order（OoO）と in-order（InO）の 2 つのバックエンドを持つ．以下ではこれらを **OoO バックエンド**と **InO バックエンド**と呼ぶ．CC は 2 つの実行モードを持っており，OoO バックエンドを使用する **high-performance（HP）**モードと InO バックエンドを使用する **low-power（LP）**モードを持つ．CC は，それらのモードを切り替えて命令を実行する．この時の切り替え間隔は HMC のコア切り替え間隔と比べて非常に短く，たとえば 500 命令程度である．

CC の研究ではバックエンドの切り替えアルゴリズムも提案されており，我々はこの切り替えアルゴリズムを**single-scale アルゴリズム (SSA)** と呼ぶ．SSA は（推定された）HP モードからの実行サイクル数の伸びがユーザーが設定された範囲（たとえば 5%）の範囲に入るよう制御する．すなわち，仮に HP と LP の双方のモードで実行したとして，HP に対する LP の実行時間の増加が小さいと推定された場合に LP モードに切り替える．このような LP モードによる実行時間の増加が少なく，LP モードで実行すると電力削減上有益な区間を以下では **LP-friendly**

[1] 名古屋大学
[2] 国立情報学研究所
[a)] shioya@nuee.nagoya-u.ac.jp

インターバルと呼ぶ．またそれとは逆に，LP モードによる実行時間の増加が大きく，HP モードでの実行に適している区間を **HP-friendly インターバル**と呼ぶ．

CC の先行研究により，LP-friendliness は細粒度（たとえば 500 命令程度）に大きく変化することが発見されており（5 節） [11, 12]，CC と SSA はそのような細粒度な LP-friendly インターバルを利用することで電力効率を向上させている．

しかし CC と SSA には以下のような問題があり，LP モード実行の機会を損なっている：

(***P1***) 切り替えペナルティが依然大きい．これは CC のハードウェアでは実行モード切り替え時にパイプライン内の全ての命令がリタイア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．SSA ではターゲット性能に追随するために可能な限り素早く実行モードを切り替える．このため，将来にある LP-friendly インターバルを逃すことがある．

CC とは対照的な別の TCHC として，**frontend execution architecture (FXA)** がある [15]．FXA は，構造上は TCHC だが，実行モードの切替は考慮されていない．FXA は CC と同様に，**InO execution unit (IXU)** と **OoO execution unit(OXU)** の 2 つの実行系を持つ．CC では InO と OoO バックエンドは並列に置かれていたのに対し，**図 2** に示すように IXU と OXU は直列に配置される．FXA では IXU と OXU は同時に働き，IXU で実行されなかった命令が OXU で実行される．

(***P3***) しかし，FXA では消費エネルギーの削減が CC の LP モードと比べると十分ではない．これは，FXA では OXU を常に働かせる必要があるためである．FXA では全ての命令が IXU で実行できる場合でも，リネーム・ユニットや issue queue (IQ) を動かし続けなければならない．

本論文では **dual-mode front-end execution architecture (DM-FXA)** と，**multi-scale アルゴリズム (MSA)** を提案する：DM-FXA は FXA をベースとして，LP モードを追加したアーキテクチャである．DM-FXA では CC と同様にして LP モードと HP モード（通常の FXA の実行）を切り替えて実行を行う．MSA は新しい実行モード切替アルゴリズムであり，SSA の持つ局所最適の問題を解決する．**表 1** に既存手法の問題点と提案手法の利点をまとめる．DM-FXA と MSA による利点は以下の通りである：

(***C1***) DM-FXA では LP から HP モードへはペナルティなく実行モードを切り替えられる．このため，CC における切り替えペナルティの影響を低減できる．

表 1: 既存手法の問題点と提案手法の利点

		Hardware	Algorithm
Problems	CC+SSA	*P1*: Large Penalty	*P2*: Shortsighted
	FXA	*P3*: OoO operates	N/A
Contribution of DM-FXA+MSA		*C1*: Small Penalty	*C2*: Farsighted
		C3: OoO stops	

(***C2***) MSA ではある程度長期的に将来を考慮することで，SSA において切り替えが局所最適になる問題を解決する．

(***C3***) DM-FXA では，LP モード中は OoO 実行に必要な処理を完全に省略する．このため，FXA において消費エネルギーの削減が不十分である問題を解決する．

評価の結果，提案手法は CC と FXA と比べてそれぞれ 35% と 20% 高い性能エネルギー比（ED 積の逆数）を達成した．この結果，MSA では理想的なオラクル・アルゴリズムに近い LP モードの使用率を達成した．

本論文では先にアーキテクチャに関する問題と提案を述べ，次にアルゴリズム面のそれらを述べる．以降の具体的な構成は以下の通りである．2 節では背景として CC と FXA のアーキテクチャを説明する．3 節では提案する DM-FXA のアーキテクチャについて説明する．4 節では SSA について述べ，その後 5 節でその問題点について解析する．6 節では MSA について述べる．7 節では評価を行う．

2. 既存のアーキテクチャ

2.1 Composite Core

Composite core (CC) は InO と OnO の 2 つのバックエンドを持ち（図 1），これらはそれぞれ LP モードと HP モード中に使用される [11, 12]．CC はコールド・スタートを避け実行モードの高速な切り替えを実現するため，両バックエンド間で分岐予測器や L1D/I キャッシュなどを共有する．

CC では以下のようにして実行モードを切り替える：1) まず命令フェッチを停止し，アクティブなバックエンドから全ての命令がリタイアするのを待つ．2) フェッチの停止と並行して，独立したレジスタ・ファイル間で値のマイグレーションを投機的に開始する．3) 前述のパイプラインからの命令のリタイアが終了すると，投機的な転送に失敗したデータを再転送する．4) 値の再転送が全て終了すると，切り替え先のバックエンドで実行を開始する．これらのリタイアと転送に必要なサイクルが CC の切り替えペナルティとなる．

CC の実行モード切替時に生じるペナルティは従来の HMC のそれと比べると極めて短いものの [*1]，CC は細粒

[*1] CC の切り替えペナルティは我々の評価では 40 サイクル程度であるが，HMC では一般に切り替えには 20 μs 程度が必要となる [1].

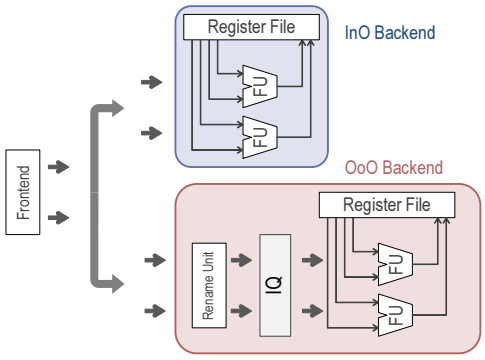

図 1: CC のブロック図.

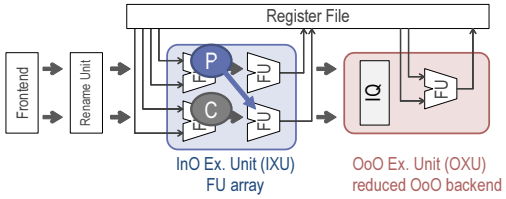

図 2: FXA のブロック図.

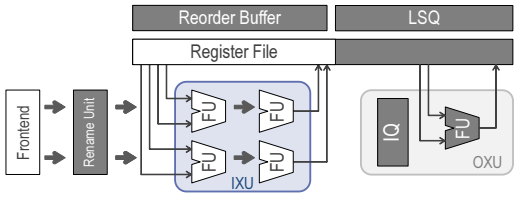

図 3: LP モードにある DM-FXA のブロック図.

度な切り替えを行うため無視できない影響を持つ．このペナルティの影響により，CC では切り替えの粒度は数百命令程度以上に制限されている [12].

2.2 Frontend Execution Architecture

2.2.1 構造と振る舞い

FXA もまた OXU と IXU の 2 つの実行系を持つ（図 2）．OXU は縮小された OoO プロセッサのバックエンドである一方，IXU は演算器アレイからなる．図 2 に示すように，IXU はフロントエンドのリネーム・ステージの後に配置され，命令を InO に実行する．

FXA では命令を以下のように処理する（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) まず，フロントエンドのレジスタ読み出しステージにおいてソース・オペランドの取得を試みる．オペランドは，(1-a) RF からの読み出しか，(1-b) IXU 内の FU からのバイパスによって得る（OXU からのバイパスはしない）．値は異なるステージにある FU 間でもバイパスされ，たとえば 2 段目において 1 段目の結果を受け取ることができる．そしてこの結果，全てのオペランドが得られ，命令がレディであるかを判定する．(2) レディな命令は IXU で実行し，命令パイプラインから取り除かれ IQ にディスパッチしない．(3) 一方，レディでない命令は NOP として IXU をそのまま通過する．通常の InO プロセッサで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が，FXA ではパイプラインはストールされずに命令は流れ続ける．その後，命令は IQ にディスパッチされ実行される．

IXU は通常の InO プロセッサよりも高い能力を持つ [15]．通常の InO プロセッサでは依存関係にある複数の命令が同時にデコードされると，パイプラインをストールさせる．これに対し，IXU は多段に配置された演算器によってパイプライン・ストールなしにそのような命令を実行できる．たとえば図 2 において，プロデューサ命令Ⓟが 1 段目で実行された場合，同じサイクルにデコードされたコンシューマ命令Ⓒは IXU の 1 段目を NOP として通過し 2 段目で実行される．このような動作の結果，IXU は多数（我々の評価ではおよそ 60%）の命令を実行することができる．

IXU では整数演算命令の他に，ロード/ストア命令，分岐命令を処理する．それ以外の複雑な floating-point（FP）命令などについては，IXU に演算器を追加した場合の回路資源増加が大きいため，IXU では実行しない．

2.2.2 利点と問題点

FXA は以下のような利点を持つ：

- 消費エネルギーが削減される．これは，IXU は命令スケジュールのためのハードウェアを持たないため，高効率に命令を実行できるためである．別の理由としては，IXU は OXU へ送られる命令を効果的にフィルタできるため（およそ 60%の命令がフィルタされる），性能を下げることなく OXU を縮小できることがある．
- 性能が向上する．これは FXA では，図 1 と図 2 に示されるように，OXU が縮小される一方で搭載している演算器の総数は増えているためである．同図の場合，CC は最大 2 命令までを実行可能だが，FXA は 5 命令を 1 サイクルあたりに実行できる．

しかし，FXA は CC とくらべて消費エネルギー削減の効果が限定されている．CC では完全に OoO 実行に必要な処理を省略できる．しかし，FXA では IXU によりフィルタできなかった命令は OXU で実行する必要があるため，リネーム・ユニットや load store queue (LSQ) などは常に稼働し続ける必要がある．

3. Dual-Mode FXA

本論文では FXA をベースとした **dual-mode front-end execution architecture (DM-FXA)** を提案する．DM-FXA は CC と同様に LP と HP のモードを持ち，両モードを切り替えながら命令を実行する．LP モードでは **IXU のみで**全ての命令を実行し，HP モードでは従来の FXA と同じ動作にて命令を実行する．

3.1 LP モード

図 3 に，LP モードにある DM-FXA のブロック図を示

す．DM-FXA の物理的なアーキテクチャは FXA をベースとしている．同図では，LP モードにおいて動作を停止するユニットを暗く塗りつぶしている．

LP モードでは，IXU においてソース・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 InO プロセッサの動作と同じである．IXU で実行できない複雑な命令（2.2.1 節）をデコードした場合は，即座に HP モードに切り替える．

LP モードでは以下のようにして OoO 実行に必要な処理を完全に停止する：

(1) IQ，ROB，LSQ を含む OXU の動作を停止する．
(2) レジスタ・リネームを停止し，論理レジスタ番号を用いて RF へアクセスする．

この結果，LP モードは 2.2.2 節で述べた FXA における消費エネルギーの削減が十分でない問題を解決する．なお，復帰には大きな時間がかかるため，既存研究 [11,12] と同様にパワーゲーティングは行わない．

3.2 HP から LP モードへの切り替え

HP モードから LP モードへの切り替え動作は以下の通りである．1) まずフェッチを停止し，パイプライン中の全ての命令のリタイアを待つ（CC と同様）．2) 次に，論理レジスタ番号で RF にアクセスができるよう，RF 内のデータを論理レジスタ番号の順に並べ替える．

上記の並び替えでは，先頭領域内で使用されているレジスタの値（以降では生きている値と呼ぶ）を一旦 RF 上の別の部分に退避した後，先頭領域へ値を移動する．

3.3 LP から HP モードへの切り替え

LP から HP モードへの切り替えは，(1) register alias table (RAT) の初期化と，(2) OoO 実行の再開により行われる．CC とは異なり，DM-FXA では LP モードから HP モードへの切り替えには**ペナルティがない**．これは，RAT の初期化は LP モード中に並列して実行できることと，パイプラインの構造上 IXU での LP モードの命令実行が全て済むまで OXU へは命令がディスパッチされず，双方のモードの命令が混じって OoO 実行されることはないためである．

4. Single-scale アルゴリズム

本節では TCHC における既存のモード切り替えアルゴリズムである **single-scale アルゴリズム (SSA)** [11,12] について説明する．

4.1 アルゴリズムの概要

図 4 に示されるように，SSA は **1) s-loop** と **2) t-loop**

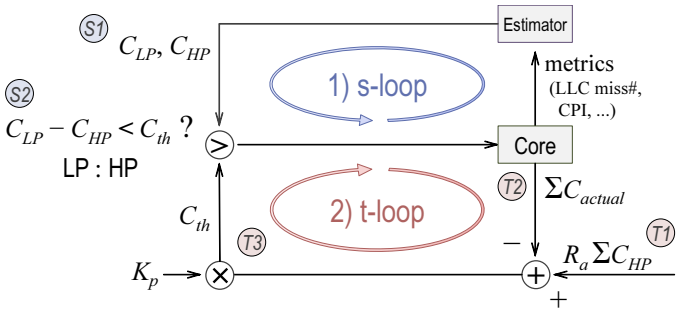

図 4: 2 つのループを持つ SSA.

と呼ぶ 2 つのループからなる．SSA では，一定数の命令（たとえば 500 命令）が実行されるごとにこれらのループが実行される．以下では，このループの実行単位となる一定数の命令を**インターバル**と呼ぶ．

4.1.1 S-Loop

図 4 上部の s-loop は，インターバル $i-1$ から i に移る際，以下のようにして実行モードを選択する．なお，図中の S1 などのラベルは，以下の各ステップと対応している．

S1 実行サイクル数 $C_{LP}(i)$ と $C_{HP}(i)$ を推定する．ここで $C_{LP}(i)$ と $C_{HP}(i)$ はそれぞれ，インターバル i を仮に LP モードと HP モードでそれぞれ実行した場合にかかると推定されるサイクル数である．この推定については 4.2 節で述べる．

S2 上記で推定したサイクル数の差 $C_{LP}(i) - C_{HP}(i)$ を閾値 $C_{th}(i)$ と比較する．この差が閾値よりも小さい場合は，そのインターバルを LP-friendly と見なし LP モードを，逆の場合は HP モードを選択する．

4.1.2 T-Loop

図 4 下部の t-loop では，s-loop で用いた閾値 $C_{th}(i)$ を動的に更新する．実際の t-loop は Proportional Integral (PI) 制御 [5] に基づいているが，本稿では簡単のため PI 制御から積分項を省いた P 制御として説明を行う．これは，積分項は挙動には大きくな影響を与えず [11]，我々が着目する問題点とも関係しないためである．

t-loop は $C_{th}(i)$ を動的に変更することで，実行サイクル数が目標実行サイクル数に追随させる役割を持つ．具体的には，$C_{th}(i)$ は各インターバル i ごとに以下の式によって求められる：

$$C_{th}(i) = K_p(R_a \Sigma C_{HP} - \Sigma C_{actual}), \qquad (1)$$

この式の各項の意味は以下の通りである：

T1 $R_a \Sigma C_{HP}$ は**目標実行サイクル数**をあらわす．ここで，$\Sigma C_{HP}(i)$ は過去の $C_{HP}(i)$ の累積である．なお，この累積の計算では，HP モードが使用されなかったインターバルついては，s-loop で求めた推定値が用いられる．R_a は許容する実行サイクル数の伸び率である．

たとえば R_a が 1.05 の場合，HP モードによる実行時から 5%の実行サイクル数の伸びを許す．

(T2) ΣC_{actual} は**累積実行サイクル数**をあらわす．式 1 は，目標実行サイクル数から累積実行サイクル数をひくことにより，実行サイクル数の余裕を計算する意味を持つ．実行サイクル数に余裕があるほど閾値 $C_{th}(i)$ が大きくなるため，より LP モードで実行されやすくなる．

(T3) K_p は実験的に求められる比例定数である [*2]．

t-loop の詳細な振る舞いについては，後の 5.1 節で詳しく述べる．

4.2 Algorithm Variation

SSA には，以下の 2 つのバリエーションがある．

REACT [11]: REACT アルゴリズムは s-loop の実行サイクル数の推定において，直前のインターバル $i-1$ で使用されていたモードについては，単純に直前のインターバルのサイクル数をそのまま用いる．インターバル $i-1$ で使用されていないモードによる実行サイクル数は直接にはわからないため，線形回帰式により推定する [11,12][*3]．ここでは，直前に実行した区間と同様の特性を持った命令列が次区間も現れるという仮定がおかれている．
先行研究によると，性能推定や PI 制御のための追加のハードウェア・コストは 5 μW の電力と 0.002 mm^2 のチップ面積であり，コア全体から比べると十分に小さいことが示されている [11,12].

PRED [12]: REACT アルゴリズムにおいて，もし細粒度に特性が変化する場合，区間を実行し終わった後に反応して切り替えたのでは手遅れになる場合がある．これに対し，PRED アルゴリズムでは予測器を導入している．この手法では REACT アルゴリズムと同様にしてモードを選択した結果を予測器に学習し，過去の実行パスから次のインターバルにどちらのモードを使用すべきか予測する．なお，閾値の制御は REACT アルゴリズムと同様に行われる．

5. SSA の振る舞いと問題点

我々は SSA の振る舞いを解析し，その問題点を見出した．本節ではそれらについて順に説明する

[*2] SSA における K_p は，我々の評価では 0.001 から 100 の範囲内では 0.95 の場合に最も良い結果を得た．
[*3] 具体的には，キャッシュ・ミス数などのメトリクスを独立変数とし，そこから線形回帰式によって実行サイクル数を求める．この線形回帰式で使用される各係数は，事前にプログラムを実行した際の結果から得られたものを用いる．

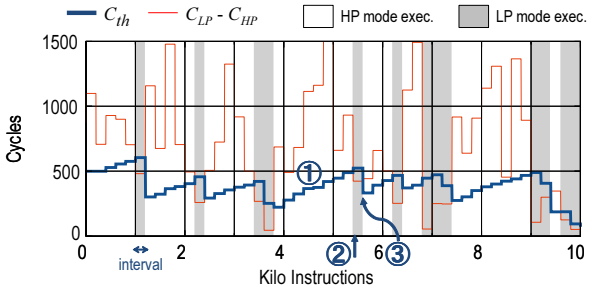

図 5: mcf のスイッチング・チャート．

5.1 振る舞い

5.1.1 スイッチング・チャート

SSA の切り替えの振る舞いを可視化するため，我々は**スイッチング・チャート**と呼ぶ図を導入する．**図 5** は SPEC CPU 2006 [16] に含まれる mcf のスイッチング・チャートである．同図の横軸は命令数であり，縦軸はサイクル数である．ここではインターバルの長さは 500 命令としている．同図の赤い細線は上記の推定実行サイクル数の差 $C_{LP}(i) - C_{HP}(i)$ を表す．この差は，LP モードで実行した場合の，HP モードからの実行時間の延びを意味する．説明の簡単のため，同図ではサイクル数の推定は理想的に行われており，実際の実行サイクル数差を表す．そして，同図の青い太線は閾値 $C_{th}(i)$ を表す．背景が白い区間は HP モードで実行されたことを，灰色の区間は LP モードで実行されたことを示す．s-loop により，赤い細線の $C_{LP}(i) - C_{HP}(i)$ が，青い太線の $C_{th}(i)$ より小さい場合は LP モード（灰色）が選択される．同図では赤い細線が下にあるほど実行サイクル数の差が小さく，そのインターバルが LP-friendly インターバルであることを意味する．

5.1.2 振る舞い

前述したように，t-loop は実行サイクル数が目標サイクル数を追随するよう $C_{th}(i)$ 制御する．この制御を，我々は $C_{th}(i)$ の**貯蓄**と**消費**として以下のように考える：

貯蓄: HP モード中は $C_{th}(i)$ が単調に**貯蓄**される．HP モードでは $C_{actual} = C_{HP}$ となるため，これを式 1 に代入すると以下の式を得る：

$$\begin{aligned} C_{th}(i) &= K_p(R_a \Sigma C_{HP} - \Sigma C_{HP}) \\ &= K_p(R_a - 1)\Sigma C_{HP}. \end{aligned} \tag{2}$$

ここで R_a は許容する実行サイクル数の伸び率であるため 1 より大きく，したがって $C_{th}(i)$ は R_a に応じて少しづつ増加する．この結果，HP モード中は徐々に LP モードに遷移しやすくなる（図 5 の①）．

消費: LP モード中は基本的に $C_{th}(i)$ が**消費**される．これは，LP モードでは HP モードよりも実行に時間がかかるため，式 1 において $\Sigma C_{HP} R_a$ の項よりも $C_{actual}(i)$ の項が相対的に増加するためである（LP モード中であるため，$C_{actual}(i)$ は $C_{LP}(i)$ に等しくなる）．この

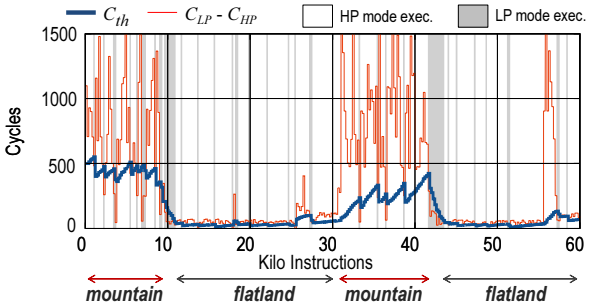

図 6: SSA による切り替えの様子.

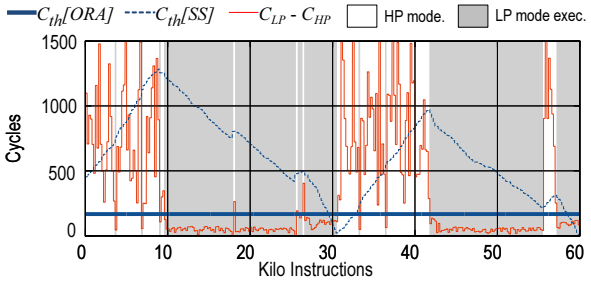

図 7: ORA による切り替えの様子.

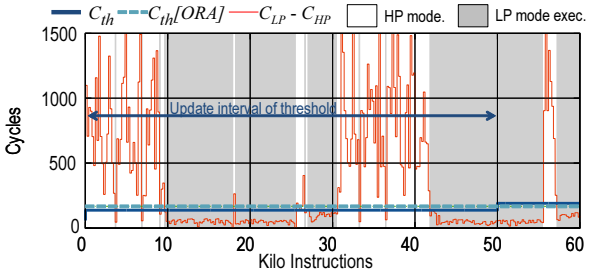

図 8: MSA による切り替えの様子.

ため，結果として LP モード中は $C_{th}(i)$ が小さくなり HP モードに遷移しやすくなる（図 5 の③）.

5.2 問題点

5.2.1 局所最適

TCHC におけるモード切り替えアルゴリズムの目的は，できる限り多くの命令を LP モードで実行することである．これを実現するためには，LP-friendliness のより高いインターバルから順に LP モードを選べばよい．

しかし SSA は，サイクル数差の傾向がマクロスケールで変化した場合には，この目的を達成できない．なぜなら，SSA は C_{th} が $C_{LP}-C_{HP}$ に可能な限り素早く追随するよう**貪欲（greedy）**に LP モードを選択しているためだ．

前節で述べたように SSA は，スイッチング・チャートにおいて青い太線の C_{th} が赤い細線の $C_{LP}-C_{HP}$ を横断した際に実行モードを即座に切り替える．この横切り後の即座の切り替えは，C_{th} を目標サイクル数にできる限り早期に追随させるよう働く．

図 5 に示すように，この横切りは赤い細線が下がっている部分で起こりやすい．このため，先行するインターバルより**相対的**に LP-friendly なインターバルが LP モードに選ばれやすいと言える．しかし，それが**絶対的**に LP-friendly である保証はない．

このため，SSA は局所最適に陥ることを我々は発見した．**図 6** にそのような局所最適が起きている典型的な例を示す．同図は 5 節を横軸方向に 1/6 に縮小したものである．すなわち，図 6 の最初の 10K 命令は図 5 と全く同じ実行区間を表しており，残りの区間はそれより未来の区間を表している．

同図では，細い赤線の実行サイクル数差が大きい ***mountain* フェーズ**と，小さい ***flatland* フェーズ**がある．特に，*flatland* フェーズでは実行サイクル数差が 0 に近く，細い赤線は横軸至近にあり LP-friendliness が絶対的に高い．したがって，*flatland* フェーズの大部分では LP モードが選択されるべきである．

しかし，以下の 2 つの現象により，SSA はこれを実現できない．

- *mountain* フェーズにおいて，C_{th} はノコギリ型に遷移する．LP モードにおける C_{th} の消費量は $C_{LP}-C_{HP}$，すなわち細い赤線の高さに比例する．したがって，C_{th} は細い赤線が高い位置にある *mountain* フェーズにおいてよりシャープに落ちる．このノコギリ型のパターンに示されるように，C_{th} は繰り返し消費され，将来にある *flatland* フェーズのための十分な貯蓄ができない．
- *mountain* フェーズから *flatland* フェーズに遷移する際の ***downhill* フェーズ**において，SSA では C_{th} を 0 近くまで消費してしまう．前述したように，太い青線の赤い細線への横断は，細い赤線が落ちるときにより起きやすい．したがって，*downhill* フェーズでは LP モードが連続して選ばれる．もし，*downhill* フェーズが図のように連続的に変化する場合，C_{th} は 0 近くまで消費されてしまう．

これらの結果として，SSA は LP-friendliness が**絶対的**に高い *flatland* フェーズを LP モードで実行することに失敗している．

この問題は，ある程度の長さの HP-friendly なフェーズに対して，LP-friendly なフェーズが続くと顕在化する．7.3.6 節の評価で示すように，そのような状況はプログラムにおいて一般に存在する．

5.2.2 オラクル・アルゴリズムとの比較

SSA と対比するため，ここで **ORA（ORA）**を導入する．このアルゴリズムは，プログラム実行における全てのインターバルの $C_{LP}-C_{HP}$ を前もって知っており，最も LP-friendly なインターバルを選ぶことができる [*4]

[*4] なお，ORA は理想的なアルゴリズムではない．なぜなら，切り替え時のペナルティが考慮されていないためである．

ORA は，プログラム全体で最適な固定の閾値 C_{th} を与える t-loop を持った SSA である考えることができる．一方，ORA の s-loop は C_{th} に従い SSA と全く同じ方法で実行モードを選択する．

この固定された C_{th} は以下のようにして決定される．まず，C_{th} を 0 からはじめて徐々に増やしていく．C_{th} が大きくなるにつれ，s-loop はより多くのインターバルを LP モードに選択する．このように C_{th} を大きくしていった場合，LP モードへの選択は **LP-friendliness の高いインターバルから順に**行われる．そしてこの結果，徐々に実行サイクル数が増加する．この C_{th} の増加は，実行サイクル数が目標実行サイクル数に達するまで続く．この時の最終的な C_{th} の値は，プログラム全体で LP-friendliness の高い順にインターバルを LP モードに選択することと，目標実行サイクル数を満たすことの双方を達成している．

図 7 に，ORA による切り替えの様子を示す．同図は 6 節と同じ実行区間の様子である．ORA によって決定された C_{th} は，$C_{th}[ORA]$ として図上で水平な直線となる．また，仮に SSA の t-loop が裏で働いていたとして，そこで算出した C_{th} を $C_{th}[SS]$ として示す．この $C_{th}[SS]$ は実行モードの選択には使用されないが，ORA による切り替えを行った場合に，SSA の C_{th} がどのように振る舞うかを見ることができる．

同図では，*mountain* フェーズでは HP モードを選択して $C_{th}[SS]$ を貯蓄し，それを *flatland* フェーズで LP モードを選択することにより消費してしている．図 6 と図 7 を比較すると，SSA は C_{th} を $C_{LP} - C_{HP}$ に最速で一致させるよう貪欲に実行モードを選択しているため，最適な切り替えを実現できていないといえる．

6. Multi-scale アルゴリズム

本節では我々が提案する **multi-scale アルゴリズム (MSA)** について述べる．MSA は**表 2** に示す複数のスケールを意識したアルゴリズムである．SSA は micro スケールのみで制御を行うため，結果として meso スケールにおいて局所最適が生じる．これに対し，MSA は **(1)macro スケールの閾値更新**を導入することで，micro スケールで起きる局所最適の問題を解決する．そして同時に **(2)instruction スケールの即時切り替え**を導入することで，macro スケールの閾値更新による反応性の悪化を補う．

表 2: スケールの一覧

scale	# insns.	phenomenon	SSA	MSA
insn.	1	LLC miss		immediate
micro	100 – 1K		s/t-loop	s-loop
meso	10K – 20K	mountain/ flatland		
macro	50K –			t-loop

6.1 Macro-scale の閾値更新

6.1.1 異なるインターバル長

驚くべきことに，s-loop のインターバル長を保ったまま t-loop のインターバル長のみを長くすることにより，ORA に近い結果を得ることができる．以下では t-loop と s-loop のインターバルをそれぞれ **t-interval** と **s-interval** と呼ぶ．

MSA では，ループ間のインターバル長の違いにより，その制御は以下の 2 つの部分問題に分けられる：

Inter-t-interval: SSA と同様にして，t-interval ごとに良い C_{th} を決定する．

Intra-t-interval: 上記で決定された C_{th} を使い，ORA と同様にして，t-interval 内で最も LP-friendliness が高い s-interval を LP モードとして選択する．

これらのうち，Inter-t-interval の部分問題については SSA と同じであるため，以下では Intra-t-interval の部分問題について詳しく述べる

6.1.2 Intra-t-interval 部分問題

この部分問題，すなわち macro スケールにおいて最も LP-friendliness が高い s-interval を選択すること，は SSA には存在しないため非常に重要である．SSA では，インターバル長は可能な限り**短く**していた．対照的に，MSA は**長い** t-interval 内で固定された C_{th} を用い，ORA と同様にして，その t-interval 内で LP-friendliness の高い順に LP モードを選択する．

この動作を**図 8** を用いて説明する．同図は図 6 と同じ実行区間の様子である．同図では，t-interval の長さは 50K 命令（macro スケール）に設定されている．この長さは，1 つ以上の *mountain* フェーズと *flatland* フェーズを含むように決定した．また，各 t-interval は 100 個の s-interval を含む．各 t-interval において C_{th} は固定され水平な線分となる．また比較のため，ORA によって決定した $C_{th}[ORA]$ を単一の水平な線として示す．

t-interval 内では，MSA は固定された C_{th} を用いて ORA と同じように振る舞う．すなわち，MSA は，LP-friendliness の最も高い s-interval を t-interval 内の 100 個の s-interval の中から選ぶことができる．結果として，MSA は ORA（図 7）と同様に，*flatland* フェーズ内の**絶対的**な LP-friendly インターバルを LP モードで実行している．

なお，SSA では t-interval を長くしたとしても同じ効果は得られない．なぜなら，s-interval もまた同時に長くなってしまうためである．MSA は最も LP-friendliness の高いインターバルを選択することができるが，これは t-interval 内に候補となる十分な数（たとえば 100 個）の s-interval があるためである．対照的に，SSA ではそもそも t-interval ごとに 1 つの s-interval しか含まず，候補の概念そのものがない．

6.1.3 t-interval の長さ

t-interval の長さについては以下の 2 つの相反する要請がある：

(1) t-interval は，intra-t-interval 部分問題においては**長い方がよい**．なぜなら，LP-friendliness の順の選択は，t-interval が十分な数の *mountain* フェーズと *flatland* フェーズを含んでいる場合にうまく働くためである．

(2) t-interval は，inter-t-interval 部分問題においては**短い方がよい**．なぜなら，t-interval を長くすると C_{th} の反応性が悪化するためだ．この反応性の悪化は，次節で述べる instruction スケールの即時切り替えによって有効に緩和することができる

7.3.6 節の評価では，上記を満たすためには t-interval は 50K 程度の長さ（macro スケール）であれば良いことが示されている．

6.2 即時切り替え

t-interval を長くすることによって C_{th} の反応性が悪化した場合，特に LLC ミス発生時に C_{th} が大きく消費されてしまうことがある．以下ではこの C_{th} の大きな消費と，それを回避するための**即時切り替え**について説明する．

6.2.1 OoO 実行と LLC ミス

C_{th} の深刻な消費の背景として，OoO 実行と LLC ミスについて説明する．一般に LLC ミスは InO と OoO プロセッサにおいて性能差を生じさせないと思われがちである．なぜなら，双方共に LLC ミス時にはパイプラインがストールするためである．

しかし，実際には InO と OoO プロセッサの間では大きな性能差が生じることがある [17]．これは HP モードでは OoO 実行により LLC ミスの影響が隠蔽されるためである．一方，LP モードでは InO 実行を行うため，このような隠蔽はできない．この結果，数十サイクル程度はモード間で実行サイクル数に差がつくことが多い．

6.2.2 大きな消費

LLC ミスにより実行サイクル数差が大きくなった場合，C_{th} は大きく消費される．なぜなら，C_{th} の消費量は**実行サイクル数差に比例**するためである（5.1.2 節）．

この問題は SSA と MSA の双方にあるが，LLC ミスが連続して発生するような場合は MSA においてより深刻となる．なぜなら，MSA の t-interval は，SSA のそれよりも 100 倍長いからである．SSA では最初の LLC ミスによって C_{th} が消費されると，次のインターバル（500 命令後）では HP モードが選択される．一方 MSA では，C_{th} は連続した LLC ミスにより，次の t-interval（50K 命令後）まで繰り返し消費され続けてしまう．

6.2.3 即時切り替え

そこで，我々は LLC ミスが発生した場合に LP モードから HP モードへ**即時切り変える**ことを提案する．また，HP モード中は将来の LLC ミスを見込んだ単純な予測を行う．すなわち，直前に実行された s-interval で LLC ミスが 1 回以上起きていた場合は，次の s-interval も LLC が起きると予測して HP モードで実行する．これにより，macro スケールの閾値更新によって C_{th} が大量に消費されてしまうことを未然に防ぐことができる．

7. 評価

7.1 評価環境

本節では評価環境について述べる．性能の評価には鬼斬 2 [14] を用いた．このシミュレータは gem5 [3] のような実行ドリブンなシミュレータであるが，gem5 と比べると動的命令スケジューリングをより正確にモデル化しており，たとえばキャッシュミス時のリプレイの挙動をより正確にシミュレーションできる．また，消費エネルギーの評価には McPAT [10] を用い，**表 4** に示すデバイスのパラメータを使用した．

評価では，SPECCPU 2006 INT に含まれるベンチマーク・プログラムを用いた．ベンチマーク・プログラムは GCC ver.4.5.3 でコンパイルし，コンパイル・オプションには “-O3” 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．これらの使用ベンチマークやプログラム実行区間は CC の先行研究 [11, 12] と基本的に同じである．

7.2 評価モデル

評価したモデルについては，以下の通りである：

(1) **BASE**: ベースラインとなる通常の OoO スーパスカラ・プロセッサを実装したモデル．

(2) **CC+SS5**: CC を実装したモデル．切り替えアルゴリズムは PRED アルゴリズム [12] を使用した．先行研究による評価と同様に，トレース・ベースの予測器には 9 ビットのトレース ID と，それに対応した PHT を用いた．最短切り替え間隔（スーパー・トレース長）[12] は 500 命令とした．また，このモデルでは先行研究 [11, 12] と同様に $R_a = 1.05$（BASE から 5%の性能低下を許容）とした．

(3) **CC+SS10**: CC+SS5 において，$R_a = 1.1$（10%の性能低下を許容）としたモデル．

(4) **FXA**: FXA を実装したモデル．このモデルでは IQ のサイズと発行幅は BASE の半分となっている．これは 2.2.2 節で述べたように，IXU が IQ への命令を性能低下なくフィルタできるためである．

(5) **DMFXA+MS5**: 提案する DM-FXA と MSA によるモデル．s-loop の更新は 500 命令毎に行い，t-loop の更新は 50K 命令毎に行う．このモデルでは $R_a = 1.05$

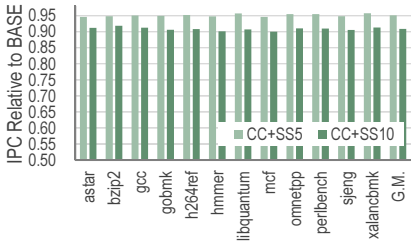

図 9: CC+SS の IPC.

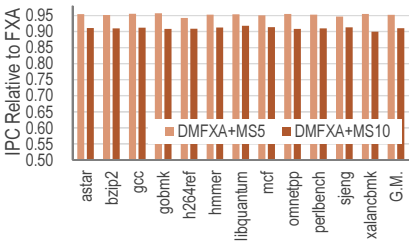

図 10: DMFXA+MS の IPC.

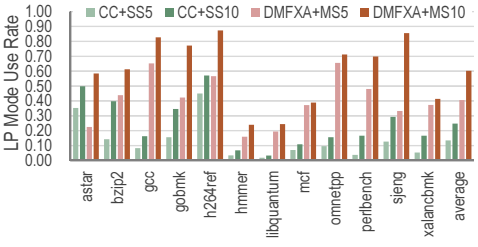

図 11: LP モードの使用率

（BASE ではなく FXA から 5%の性能低下を許容）とした．2.2.2 節で述べたように，FXA は BASE よりも平均 6.8% 性能が高いため，結果としてこのモデルは BASE とほぼ同じ性能をもつ．なお，このモデルでは FXA と同様に縮小された IQ を持つ．

(6) **DMFXA+MS10**: DMFXA+MS5 において $R_a = 1.1$（10% の性能低下を許容）としたモデル．DMFXA+MS5 と同じ理由により，このモデルは CC+SS5 とほぼ同じ性能を持つ．

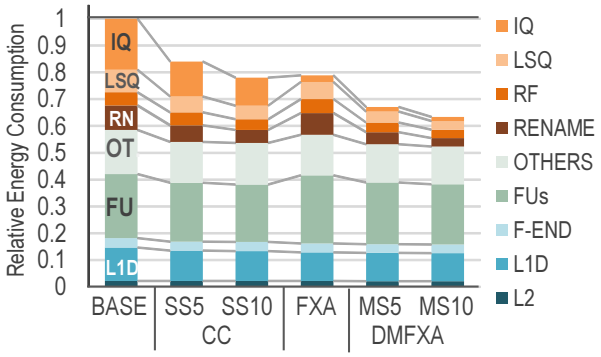

図 12: BASE に対する消費エネルギー削減量

表 3 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/A53 [4,8] にあわせている．また，CC+SS10 の InO バックエンドは 3-issue の InO スーパスカラである．これらのプロセッサ構成は，FXA の先行研究の評価で用いられた構成と基本的に同じである [15].

先行研究と同様に，切り替え制御に用いられるハードウェアの消費エネルギーは評価結果に含まれていない．これは 6 節で述べたように，提案手法と先行研究では制御のためのハードウェアはほとんど同じであり，そしてそれはプロセッサ全体と比べて非常に小さいためである [11,12].

7.3 提案手法全体の効果

以下では，まず提案するアーキテクチャとアルゴリズム全体の効果について評価した後，その後の節でアーキテクチャやアルゴリズムを個別に評価する．

7.3.1 性能の制御の精度

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 9** に BASE に対する CC+SS の，**図 10** に FXA に対する DMFXA+MS の相対性能を示す．CC+SS5, CC+SS10, DMFXA+MS5, DMFXA+MS5 における目標実行サイクル数からの平均誤差率はそれぞれ 0.38%，0.40%，0.30%，0.33%であり，高い精度で目標へ制御できている．これらによる制御の結果，CC+SS5 は BASE とくらべて幾何平均で 95.2% の性能となっている．また，DMFXA+MS10 は BASE とくらべて幾何平均で 96.7% の性能となっている．これは，DMFXA+MS10 の HP モードは BASE よりも 6.8% 性能が高いためである．

7.3.2 LP 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 LP モードで実行された命令数の割合を**図 11** に示す．CC+SS5 の LP モード使用率が平均 14% であるのに対し，同じく 5%の性能低下率を設定した DMFXA+MS5 では平均使用率 41% を

表 3: プロセッサの構成

	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	←
I/D TLB	64/64 entries	←
u-op cache	4KB, one cycle	←
L1 I-cache	48KB, two cycles	←
L1 D-cache	32KB, two cycles	←
L2 cache	512KB, 12 cycles	←
L2 prefetcher	stream prefetcher	←
Main Memory	200 cycles	←
IXU	N/A	width 3, depth 3
ISA	Alpha	←

表 4: デバイスのパラメータ

Technology	22 nm, Fin-FET
Temperature, VDD	320 K, 0.8 V
Device type (core)	High performance (I_off: 127 nA/um)
Device type (L2)	Low standby power (I_off: 0.0968 nA/um)

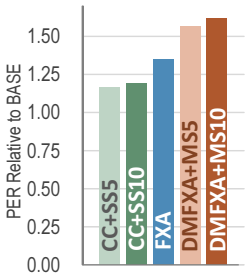

図 13: BASE に対する相対 PER.

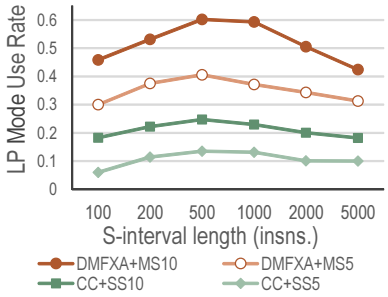

図 14: s-interval 長ごとの LP モード使用率.

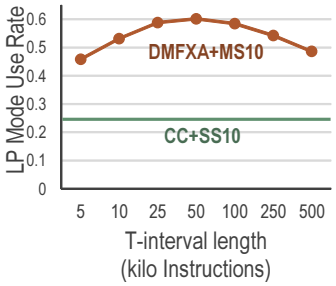

図 15: t-interval 長ごとの LP モード使用率.

達成している．さらに CC+SS5 と同等の性能である DMFXA+MS10 における平均使用率は 60%である．これは 3 節や 6 節で述べた DM-FXA や MSA の効果による．

7.3.3 消費エネルギー

図 12 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．これらは，動的と静的の消費エネルギーを含む．

DMFXA+MS10 は BASE と比較して平均で 38% 消費エネルギーを削減している．これは，OoO 実行のための IQ や LSQ, RF，リネーム・ロジックなどの消費エネルギーを大きく削減しているためである．DMFXA+MS10 は，ほぼ同じ性能をもつ CC+SS5 と比較して平均で 25% 消費エネルギーを削減した．これは前述したように LP モードの使用率が大幅に増えているためである．また DMFXA+MS10 は FXA と比較して平均 20% 消費エネルギーを削減している．これは，DM-FXA の LP モードにおいてリネーム・ロジックなどの OoO 実行に必要な処理が省略されているためである（2.2.2 節）．

7.3.4 性能エネルギー比

本節では，各モデルの消費エネルギー削減量が性能低下に見合うものかを評価するため，性能エネルギー比（PER: performance/energy ratio）[*5] を示す．**図 13** に BASE に対する相対平均 PER を示す．DMFXA+MS10 は，BASE とくらべて 62% だけ PER を改善している．また，CC+SS10 や FXA とくらべて 35% と 20% だけ PER を改善している．これは，DMFXA+MS10 では性能低下に比べて，大きく消費エネルギーを削減しているためである．なお，FXA では消費エネルギー削減量と比べて PER の改善量が大きい．これは FXA では他のモデルとは異なり，性能が向上しているためである．

7.3.5 s-interval の長さ

図 14 に各モデルの s-interval 長を変えた場合の LP モード使用率を示す．全てのモデルにおいて s-interval 長を 500 命令とした時が最もよい使用率を示す．また，インターバル長を 500 命令より短くすると使用率が落ちているのは，これは切り替えペナルティの影響が大きくなるためである．

7.3.6 t-interval の長さ

6.1.2 節で述べたように，t-interval の長さは相反する 2 つの要請がある．LP-friendliness の高さの順に選択を行うためには *mountain* フェーズと *flatland* フェーズが多く含まれるように長い方が良い．一方，C_{th} の反応性を上げるためには短い方が良い．

図 15 に t-interval の長さを変えた場合の DMFXA+MS10 の LP モード使用率を示す．同図より，t-interval 長を 50K 命令とした時に最も高い LP モード使用率が得られている事がわかる．

7.4 アルゴリズムの効果

これまではアーキテクチャとアルゴリズムを併せて評価を行ってきたが，本節ではアルゴリズム単体による効果を評価する．ここでは DM-FXA において，以下のモデルを評価した：(1) SS: SSA, (2) MT: MSA における macro スケールの閾値更新 (6.1 節), (3) IM: 即時切り替え (6.2 節), (4) MS(MT+IM): MSA, (4) ORA: ORA．また，本評価では R_a は 1.1 に設定した．

図 16 に，これらのモデルにおける LP モードの使用率を示す．平均で，SS では 31%，MT では 46%，MS(MT+IM) では 39%，ORA では 60%，LP モードが使用されている [*6]．6.2 節で述べたように，omnetpp などのメモリインテンシブなベンチマークでは MT は LP モード使用率を大きく落としているが，一方 IM は有効に働いており，それらが組み合わさることで MS(MT+IM) は LP モード使用率を大きく改善している．アルゴリズム単体でみた場合，MS(MT+IM) は SS と比べて 29%使用率を改善している．また，MS(MT+IM) は ORA に近いところまで LP モードを使用しており，その差は 11%である．

7.5 アーキテクチャの効果

前節とは逆に，本節ではアーキテクチャによる効果の評

[*5] PER は ED 積の逆数となる

[*6] 一部のモデルでは ORA の値をわずかに超えているが，これは ORA では切り替えペナルティが考慮されておらず，真に理想的な切り替えは行われていないためである．

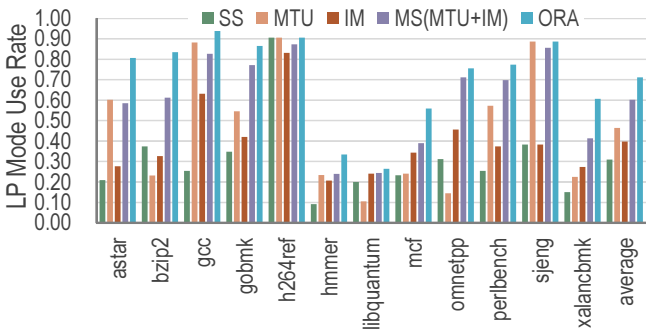

図 16: 切り替えアルゴリズムごとの LP モード使用率

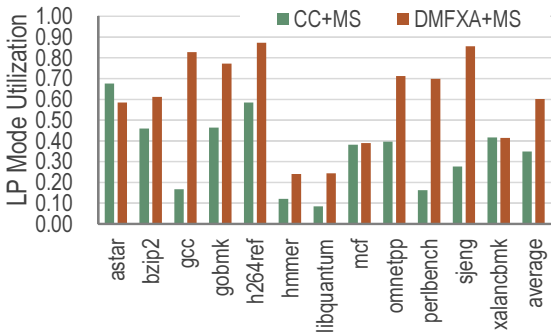

図 17: アーキテクチャごとの LP モード使用率

価を行う．ここでは MSA を CC と DM-FXA に実装して評価した．**図 17** に，それぞれのモデルの LP モード使用率を示す．本評価では R_a は 1.1 に設定した．CC+MS10 に対して，DMFXA+MS10 では同じアルゴリズムでも使用率を平均 25% 向上させている．この理由の 1 つは，DM-FXA の切り替えペナルティは CC よりも小さいためである（3.3 節）．さらに別の理由として，DM-FXA では IXU の性能は通常の InO プロセッサよりも高い点がある（2.2.1 節）．

8. おわりに

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 TCHC が提案されてきた．しかし，既存の TCHC 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 MSA 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% の消費エネルギーが削減できることを確かめた．

参考文献

[1] ARM: ARM Unveils its Most Energy Efficient Application Processor Ever; Redefines Traditional Power And Performance Relationship With big.LITTLE Processing (2011).
[2] Becchi, M. and Crowley, P.: Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures, *Proceedings of the 3rd Conference on Computing Frontiers*, pp. 29–40 (2006).
[3] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D. and Wood, D. A.: The Gem5 Simulator, *SIGARCH Comput. Archit. News*, Vol. 39, No. 2, pp. 1–7 (2011).
[4] Bolaria, J.: Cortex-A57 Extends ARM's Reach (2012). Microprocessor Report 11/5/12-1.
[5] Dorf, R. C. and Bishop, R. H.: *Modern Control Systems*, Pearson (Addison-Wesley) (1998).
[6] Greenhalgh, P.: Big.LITTLE Processing with ARM Cortex-A15 and Cortex-A7 (2011). Whitepaper.
[7] Joao, J. A., Suleman, M. A., Mutlu, O. and Patt, Y. N.: Bottleneck Identification and Scheduling in Multithreaded Applications, *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 223–234 (2012).
[8] Krewell, K.: Cortex-A53 Is ARM's Next Little Thing (2012). Microprocessor Report 11/5/12-2.
[9] Kumar, R., Farkas, K. I., Jouppi, N. P., Ranganathan, P. and Tullsen, D. M.: Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction, *Proceedings of the 36th Annual International Symposium on Microarchitecture (MICRO)*, pp. 81–92 (2003).
[10] 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).
[11] 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).
[12] 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).
[13] Rangan, K. K., Wei, G.-Y. and Brooks, D.: Thread Motion: Fine-grained Power Management for Multi-core Systems, *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pp. 302–313 (2009).
[14] Shioya, R., Goshima, M. and Sakai, S.: Design and Implementation of a Processor Simulator Onikiri2, *Proceedings of Symposium on Advanced Computing Systems and Infrastructures (SACSIS)*, pp. 120–121 (2009).
[15] Shioya, R., Goshima, M. and Ando, H.: A Front-end Execution Architecture for High Energy Efficiency, *Proceedings of the 47th Annual International Symposium on Microarchitecture*, pp. 419–431 (2014).
[16] The Standard Performance Evaluation Corporation: *SPEC CPU 2006 Suite*.
[17] Van Craeynest, K., Jaleel, A., Eeckhout, L., Narvaez, P. and Emer, J.: Scheduling Heterogeneous Multi-cores Through Performance Impact Estimation (PIE), *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 213–224 (2012).