

フロントエンド実行方式における 高エネルギー効率なメモリ・レベル並列性の利用

出岡 宏二郎^{†1} 塩谷 亮太^{†1} 安藤 秀樹^{†1}

概要: プロセッサとメモリ間の速度差は非常に大きく、コンピュータの性能を大きく制限している。この問題を解決する方法の一つとしてメモリ・レベル並列性 (MLP: memory-level parallelism) の利用がある。本研究では高いエネルギー効率で MLP を利用する手法としてフロントエンド実行方式を拡張した continual front-end execution を提案する。フロントエンド実行方式は通常のアウト・オブ・オーダー・スーパースカラ・プロセッサのバックエンドに加え、フロントエンドにインオーダーに命令を実行する実行系を持つ。フロントエンドの実行系で命令をインオーダーに実行することにより、バックエンドへの命令ディスパッチをフィルタし、エネルギー効率を上げる。提案手法は、プロセッサのバックエンドが最終レベル・キャッシュ・ミスによってストールしている間、フロントエンドのみで命令実行を継続するものである。フロントエンドで命令実行を継続することにより、プログラム上で後続する LLC ミスを起こす命令を多数実行する。この継続した実行はフロントエンドでインオーダーに行われる。このため、従来の MLP を利用して多数の LLC ミスを起こす命令を実行するアーキテクチャと比べると、少ない消費エネルギーで MLP を利用することができる。

1. はじめに

プロセッサとメモリの速度差は非常に大きく、ラスト・レベル・キャッシュ (LLC: last-level cache) のミスによる性能低下は著しい。この性能低下を緩和する方法としてメモリ・レベル並列性 (MLP: memory-level parallelism) の利用がある。通常は LLC のミスでプロセッサ全体がストールするが、多くの手法ではなんらかの方法で実行を継続して複数のメモリ・アクセスを同時に処理する [4], [5], [6], [7], [9]。これらの既存手法の多くは OoO 実行のための機構を拡張して MLP を利用するため、無駄なエネルギーを消費する。

本論文では **continual front-end execution architecture (C-FXA)** を提案する。C-FXA は front-end execution architecture (FXA) をベースとして、高いエネルギー効率で MLP を利用するアーキテクチャである。FXA はインオーダー実行系 (IXU: in-order execution unit) と OoO 実行系 (OXU: out-of-order execution unit) の二つの実行系からなる。IXU は主に演算器とバイパス・ネットワークのみからなり、プロセッサのフロントエンドでインオーダーに命令を実行する。このため、命令実行の消費エネルギーが小さい。一方、OXU は通常の実行コアと同様の構成を

持ち、アウト・オブ・オーダーに命令を実行できる。IXU は OXU へのフィルタとして働き、IXU で実行された命令はパイプラインから取り除かれる。このため、OXU の消費エネルギーを削減することができる。

C-FXA は FXA を拡張し、発行キュー (IQ: issue queue) の資源不足により OXU がストールしている間に IXU で命令の実行を継続する。LLC ミスが発生した場合、通常のプロセッサでは資源不足によりプロセッサ全体がストールする。FXA もまた、LLC ミス時にはフロントエンドの IXU を含むプロセッサ全体がストールする。これに対し C-FXA は、**hibernation queue (HQ)** と呼ぶバッファを利用する。OXU で資源不足が発生した時には、OXU を使わず IXU だけで命令の実行継続を試みる。このとき IXU で実行できない命令は HQ に入れパイプラインから取り除く。IXU での実行を継続することにより、より後続のメモリ・アクセスを実行する。

C-FXA は、消費エネルギーを抑えつつ LLC ミスを起こす多数のロード命令を実行し性能を改善できる。消費エネルギーを抑えられるのは IQ を用いない IXU で実行を継続するためである。従って C-FXA は、既存手法より低い消費エネルギーで実行の継続が可能である。シミュレーションの結果、SPEC CPU2006 [2] のメモリ・インテンシブなベンチマークにおいて、C-FXA は IQ の大きさを変えるこ

^{†1} 現在、名古屋大学大学院工学研究科
Presently with Graduate School of Engineering, Nagoya University

となく、IQを4倍に拡大したプロセッサと同程度の性能を達成した。また、C-FXAによって早期実行できたLLCミスを起こす命令数もIQを4倍に拡大したプロセッサと同程度の結果となった。

本論文の以降の構成は次の通りである。2章では関連研究について述べる。3章では提案手法のベースとなるフロントエンド実行方式について説明した後、4章で提案手法について述べる。5章では評価結果を示し、6章でまとめる。

2. 関連研究

MLPを利用して性能を向上させる手法はこれまで多く研究されている [4], [5], [6], [7], [9]。しかし、既存手法の多くはOoO実行のための機構を拡張してMLPを利用するため、通常のOoOスーパースカラ・プロセッサ以上の電力を消費する。この節では既存手法について詳しく説明する。

2.1 Runahead 実行

Runahead 実行 [5] は LLC ミスが発生した際に、Runahead モードと呼ぶ状態に遷移し、命令の仮実行によりプリフェッチを行う。Runahead モード中は、キャッシュ・ミスが発生した場合でもデータが得られたものとしてヒット時のように命令の実行、リタイアを継続する。これにより、LLC ミス処理時間中にできるだけ多くの命令を実行する。このようにして実行された命令の中に LLC ミスを起こす命令があればそれは Runahead モードから通常モードに復帰した時のプリフェッチとして働く。

前述したように Runahead モードは仮実行であり、その実行結果は全て捨てられてしまう。すなわち、Runahead モード中はコミットを行わず、通常モードへの復帰時にはプロセッサ状態が回復される。このような冗長な実行により、余分なエネルギーが消費されてしまう。

2.2 Decoupled KILO-Instruction Processor

Pericas らは、**decoupled kilo-instruction processor (D-KIP)** という OoO プロセッサとインオーダー・プロセッサが直列に結合された構造を持つアーキテクチャを提案している [6]。D-KIP では IQ と物理レジスタ・ファイル (PRF: physical register file) を長時間占有する LLC ミスに依存した命令をインオーダー・プロセッサに送ることで、OoO プロセッサでの実行を継続し、MLP を利用している。D-KIP は前方に OoO プロセッサ、後方にインオーダー・プロセッサを持ち、各プロセッサの間はバッファによって命令と値の受け渡しが行われている。OoO プロセッサは命令をフェッチし一定時間が経過すると、その命令が LLC ミスに依存するかどうかを調べる。LLC ミスに依存した命令はインオーダー・プロセッサにバッファを介して送られる。

D-KIP は OoO プロセッサとインオーダー・プロセッサで

独立したレジスタ・ファイルを持つ。そのため、これらの間で相互に値をコピーする必要があり、無駄なエネルギーや遅延が発生する。

2.3 Long Term Parking

C-FXA と同様にバッファに命令を退避させることで実行を継続する手法は多く提案されている [4], [7], [9]。最新の研究では Andreas らが、長期間実行できない命令をフロントエンドで見つけ、それらの命令を **long term parking (LTP)** と呼ぶバッファに退避させることでリネームとディスパッチを遅らせる手法を提案した [7]。これにより通常であれば IQ, PRF の資源を長期間占有する命令を命令パイプラインから一時的に取り除く。LTP からの復帰は、先頭の命令がレディになったとき、先頭から順に命令をディスパッチする。LTP で待機する命令はレジスタ割り当ても遅延されるため、必要となる物理レジスタ数も削減できる。さらに、実行のクリティカル・パス上にない (not-urgent) 命令を予測し、それらも同時に LTP に挿入する。これにより、IQ と PRF が長期間占有されることを防いでいる。

LTP は Runahead 実行や D-KIP と異なり無駄な実行や値の転送を行うことはないが、結局すべての命令を OoO に実行するため、エネルギー削減は行えない。また、LTP は not-urgent な命令の予測や (IQ より単純ではあるが) ウェイクアップ機構などが追加が必要となる。我々の提案手法ではこれに対し、単純な FIFO の追加のみでインオーダーに実行を継続する。

3. Frontend Execution Architecture

本研究は **front-end execution architecture (FXA)** [8] をベースとしている。以下では FXA の構成について説明した後、基本的な動作について述べる。

3.1 構成

図 1 と図 2 にそれぞれ通常の OoO スーパースカラ・プロセッサ (以下、通常のスーパースカラ・プロセッサ) と FXA のブロック図を示す。FXA は図 1 のような通常のスーパースカラ・プロセッサをベースとして、フロントエンドに演算器とバイパス・ネットワークを追加した構造を持つ。このフロントエンドに追加した演算器とバイパス・ネットワークを IXU, バックエンドを OXU と呼ぶ。以下でそれぞれの実行系について説明する。

3.1.1 OoO 実行系

OoO 実行系 (OXU: out-of-order execution unit) は通常のスーパースカラ・プロセッサの実行コアと同様のものである。図 2 に示すように、OXU は IQ 以降のバックエンド部分となる。OXU は主に演算器やバイパス・ネットワーク、動的命令スケジューリングを行う IQ からなる。

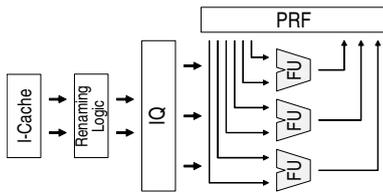


図 1 通常のスーパースカラ・プロセッサ

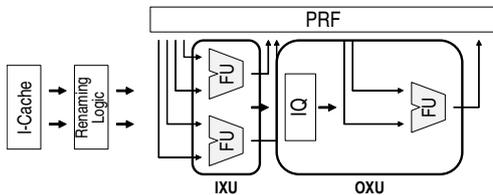


図 2 FXA

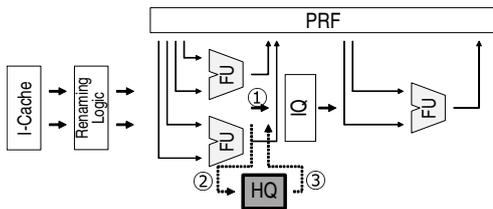


図 3 C-FXA の構成

3.1.2 インオーダー実行系

インオーダー実行系 (IXU: in-order execution unit) は図 2 のように、リネーム論理と IQ の間に配置される。IXU は主に演算器とバイパス・ネットワークからなる。FXA は PRF 読み出しステージが、OXU とは別にリネーム論理の後にもある。読み出されたソース・オペランドは IXU に入力され、そこで命令をインオーダーに実行する。

3.2 基本的な動作

FXA では IXU が 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。なお、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であるとする。

- (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。ソース・オペランドは以下の 2 通りの経路によって得られるため、これらによって命令がレディであるかどうかを判断する。
 - (a) PRF からの読み出し。
 - (b) IXU 内の実行結果のバイパス。
- (3) 命令がレディかどうかにより、次のように処理する：
 - (a) レディな命令は IXU で実行し、IQ にディスパッチしない。
 - (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。

上記の内、IXU が通常のインオーダー・スーパースカラ・プロセッサと異なるのはレディでない命令の処理である。通常のインオーダー・スーパースカラ・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。

3.3 消費エネルギーの削減

FXA は IXU で多くの命令を実行することにより、消費エネルギーを削減する効果がある。IXU で実行された命令はパイプラインから取り除かれるため、IQ にディスパッチされる命令数が削減される。このため、プロセッサの性能を落とすことなく IQ の容量と同時発行数を削減できる。IXU の追加による消費エネルギーの増加は小さく、IQ の縮小によって消費エネルギーを大幅に削減するため、FXA は消費エネルギーを削減できる。

4. 提案手法

本章では、continual front-end execution architecture (C-FXA) を提案する。C-FXA では FXA をベースとして、その実行方式を拡張する。C-FXA は hibernation queue (HQ) と呼ぶキューを利用し LLC ミス時に実行を継続する。FXA ではバックエンドにある IQ のエントリが不足するとディスパッチを行えないため、IXU を含むフロントエンドまでストールするが、C-FXA は IQ のエントリ不足時には IXU で実行できない命令を HQ に避け、フロントエンドの IXU で実行を継続する。これにより、通常の FXA であればストールしてしまう場合でも、C-FXA は後続するロード命令を IXU で実行し LLC ミスを起こす命令を実行し MLP を利用できる。

4.1 ハードウェア構成

図 3 に C-FXA のブロック図を示す。FXA から C-FXA への主な変更点は単純な FIFO である HQ の追加である。HQ は OXU の資源不足時に IXU で実行できない命令を保持する。HQ の各エントリは IQ のペイロード RAM と同様、op コード、ソース・レジスタの物理レジスタ番号、デスティネーション・レジスタの物理レジスタ番号を持つ。

4.2 動作

本節では C-FXA の基本動作とその例を説明する。

4.2.1 基本的な動作

C-FXA では IQ の資源不足が発生しない限り、通常の FXA と同様に動作する (図 3 の①)。IQ の資源不足が発生するときは通常の FXA と異なり、HQ に命令が挿入される (図 3 の②)。通常の FXA であれば IQ に空きがない場合、IXU で実行できない命令をディスパッチすること

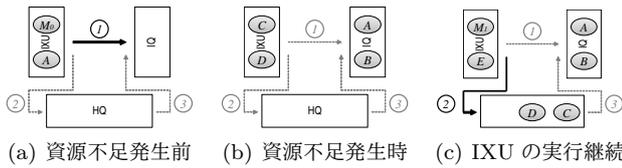


図 4 LLC ミス時の C-FXA の動作

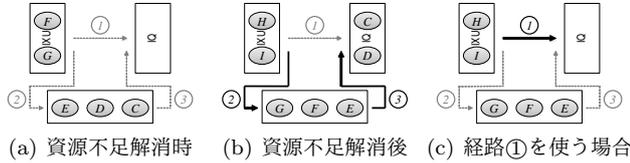


図 5 IQ 資源不足解消時の C-FXA の動作

ができず、プロセッサ全体がストールする。C-FXA では HQ に命令を退避させることで、フロントエンド部分のみで実行を継続することができる。IQ に空きが生じた場合は HQ から命令を取り出す (図 3 の③)。HQ から命令を取り出している最中、IXU で実行されない命令は並行して HQ に挿入される。この理由は 4.2.2 節でより詳しく述べるが、HQ に命令が入っているときに、図 3 の①に示す経路でディスパッチを行うとデッドロックが生じる可能性があるためである。

4.2.2 動作例

本節では図 4, 5 を用いて C-FXA の動作について説明する。これらの図において、グレーの楕円は命令を表しており、 M_0, M_1 は IXU で実行可能な LLC ミスを起こす命令、 A は M_0 に依存する命令、 $B \sim I$ はそれぞれ、直前の命令に依存する命令である。例えば、 B は A に依存し、 C は B に依存している。

まず IQ の資源不足発生時の C-FXA の動作について、図 4 を用いて順に説明する。

図 4(a) : LLC ミスを起こす命令 M_0 が実行され、パイプラインから取り除かれる。一方、 A は M_0 に依存しているため LLC ミスが解決されるまで実行できず、IQ にディスパッチされる。

図 4(b) : B は A に依存するため、 A の依存元である M_0 による LLC ミスが解決されるまで実行できず、IQ に留まる。 A, B が LLC ミスの解消まで発行不能なため、IQ は資源不足を起こす。IQ の資源不足により、 C, D はディスパッチできず、通常の FXA であればこのときプロセッサ全体がストールする。

図 4(c) : IXU で実行されない命令 C, D は HQ に挿入され、IXU で後続の命令 M_1, E の実行が継続される。LLC ミスを起こす M_1 を IXU で実行したことにより、MLP を利用できる。

次に、図 5 を用いて IQ の資源不足が解消されたときの C-FXA の動作を示す。

図 5(a) : M_0 の LLC ミスが解消され、 A, B が IQ から発行

されたとする。通常の FXA であれば、IXU で実行されない命令 F, G は IQ にディスパッチされる。

図 5(b) : HQ から IQ に命令 C, D をディスパッチする。同時に、IXU で実行されなかった命令 F, G は HQ に挿入される。

ここで、図 5(c) のようなときに、①の経路で命令をディスパッチするとデッドロックになることを説明する。同図は、図 5(b) で、 C, D が発行された後の様子を表している。IXU で実行されない命令 H, I が①の経路でディスパッチされると、 H, I は自身より古い命令 E, F, G を追い越して IQ に入る。IQ は H, I によって資源不足となる。 I は直前の命令 H に依存し、 H の実行完了を待つ。同様に、 H は直前の命令 G に依存し、 G の実行完了を待つ。しかし、 G は HQ 内で IQ の資源不足が解消されること、すなわち H, I が発行されることを待っている。こうして、 H, I と G は互いの実行完了を待つ状態 (デッドロック) となる。

4.3 消費エネルギー

C-FXA は実行継続にあたり、消費エネルギーの増加が非常に少ない。これは実行の継続を消費エネルギーの小さいインオーダー実行によって行なっていることと、HQ が単純な FIFO で消費エネルギーが小さいためである。従って C-FXA は非常に高いエネルギー効率で実行を継続し、MLP を利用することができる。一方、LTP は C-FXA と同様に FIFO に命令を挿入することで実行を継続している [7] が、C-FXA と比べるとその消費エネルギーは大きい。これは次の 3 つの理由がある。まず、LTP は実行の継続を OoO 実行によって行ない、命令スケジュールに大きなエネルギーを必要とする。次に、LTP へ挿入する命令の選択は、追加の機構を必要とし、それを使う消費エネルギーが増加する。最後に、LTP は IQ とは別に専用のウェイクアップ論理があり、これも消費エネルギーが増加する。以上より、C-FXA は LTP より小さな消費エネルギーで実行を継続することができる。

5. 評価

シミュレーションにより C-FXA とその他のアーキテクチャを評価した。以下では評価環境について述べた後、性能と MLP 利用効果についての評価結果を説明する。

5.1 評価環境

性能の評価は、鬼斬式シミュレータ [10] をベースに提案手法を実装したシミュレータを用いた。評価では SPEC-CPU2006 [2] に含まれるベンチマークの内、ベース・モデルでの平均メモリ・レイテンシが 10 サイクル以上の 15 本のベンチマーク・プログラムをメモリ・インテンシブなベンチマークとして選択した。ベンチマークは全て gcc 4.5.3 を用いてコンパイルし、コンパイル・オプションには -O3

表 1 評価モデルの構成

	BASE	C-FXA
F/I/C Wid.	3/4/3	3/2/3
IQ	64 entries	←
PRF(INT/FP)	128/128 entries	512/512 entries
Functional Unit	ALU:2, FPU:2, MUL:1, MEM:2	←
LQ/SQ	32/32 entries	128/128 entries
ROB	128 entries	512 entries
Branch Pred.	gshare, 4K entries PHT, 512 entries BTB	←
L1 I-Cache	48KB, 12-way, 3 cycles, 64 bytes/line	←
L1 D-Cache	32KB, 8-way, 4 cycles, 64 bytes/line, 2 ports	←
L2 Cache	512KB, 8-way, 10 cycles, 64 bytes/line	←
Main Memory	300 cycles, 64 bytes/cycle	←
IXU	none	wid. 3 dep. 4

を用いた。入力セットは ref を使用し、プログラムの先頭 16G 命令をスキップした後の 100M 命令について測定した。

5.2 評価モデル

本節では評価モデルについて説明する。PRF と LSQ については、今回の評価では単純に拡大している。以下のモデルの評価を行なった。

BASE : 通常のスーパースカラ・プロセッサ・モデル

LARGE+IQx1 : PRF, LSQ, ROB を BASE の 4 倍にしたモデル

LARGE+IQx4 : LARGE+IQx1 の IQ を 4 倍にしたモデル

C-FXA : 提案手法のモデル

表 1 に BASE と C-FXA の基本的な構成を示す。BASE のパラメータは、ARM Cortex-A57 と同等としている [1]。LARGE+IQx4 モデルを比較対象とした理由は、過去の研究で IQ, ROB, RPF, LSQ を 4 倍にしたモデルと Runahead 実行モデルが同程度の性能を示すことに基づいている [3]。

5.3 性能評価

図 6 に各モデルの IPC を示す。この図は縦軸に BASE で正規化した IPC, 横軸にベンチマークをとっている。G.M.mem. はメモリ・インテンシブなベンチマーク, G.M.comp. は計算インテンシブなベンチマークにおける IPC の幾何平均である。

メモリ・インテンシブなほとんどのベンチマークにおいて LARGE+IQx1 は BASE と同程度の性能である。これは IQ の資源数がボトルネックになっているためである。LARGE+IQx1 に十分な IQ を加えた LARGE+IQx4 では BASE や LARGE+IQx1 より非常に高い性能を達成している。

milc を除き C-FXA は LARGE+IQx4 と同程度の性能を達成している。C-FXA が LARGE+IQx4 と同程度の性能であ

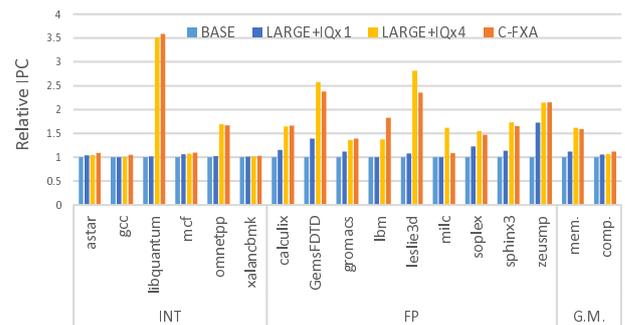


図 6 各モデルの IPC 評価

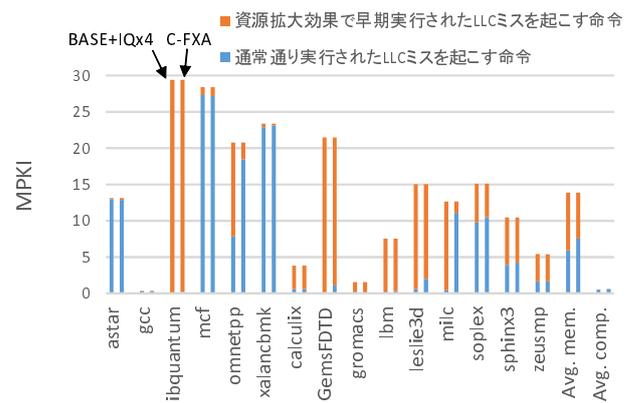


図 7 各モデルの MPKI

るといことは、HQ によって IQ が小さくとも十分に実行の継続と、LLC ミスをするロード命令の早期実行ができることを示している。実際に LARGE+IQx4 と C-FXA において IQ の拡大または HQ の追加によって LLC ミスを起こす命令を早期実行できた数を図 7 に示す。同図は全 MPKI と、その内 IQ 拡大または HQ の追加によって早期実行できた LLC ミスをする命令による MPKI を表している。同図は縦軸に各ベンチマークの MPKI を表しており、赤色の領域は IQ の拡大または HQ の追加によって早期実行できた LLC ミスをする命令による MPKI を表している。各ベンチマークの左側の棒グラフが LARGE+IQx4, 右側の棒グラフが C-FXA の MPKI を表している。Avg. mem. と Avg. comp. はそれぞれメモリ・インテンシブなベンチマークと計算インテンシブなベンチマークの MPKI の平均である。同図より omnetpp と milc を除いたベンチマークで、IQ の拡大または HQ の追加によって早期実行できた LLC ミスを起こす命令数は同程度であることが確認できる。

提案手法の C-FXA では milc においてほとんど性能向上していない。これは milc では IXU で十分に命令を実行できないためである。図 8 に C-FXA における IXU の命令実行率を示す。同図より、milc では IXU の命令実行率が極端に少ないことが分かる。

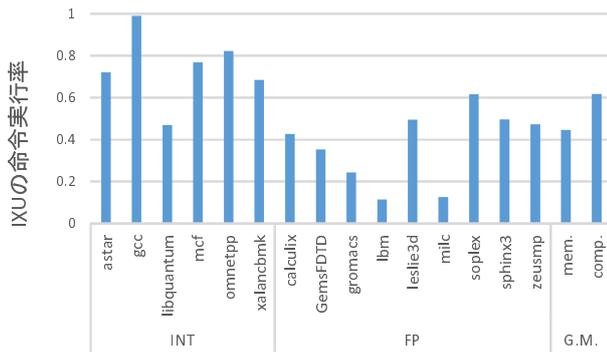


図 8 IXU の命令実行率

6. まとめ

プロセッサとメモリの性能差が大きくなり、メモリ・アクセスによりプロセッサの性能は制限されている。この問題に対して、既存の手法はアウト・オブ・オーダー実行をより強化することで MLP を利用し、性能を改善している。このような手法は、性能を得る代償に大きなエネルギーを消費してしまう。これに対し、本論文ではインオーダー実行で MLP を利用する C-FXA を提案した。C-FXA では IQ の資源不足時に IXU で実行できない命令を HQ と呼ぶバッファに退避することで IXU のみで実行を継続する。評価の結果、C-FXA によって IQ を拡大することなく、IQ を 4 倍に拡大したモデルと同程度の性能を達成した。IQ と同様にスケラビリティの低い PRF, LSQ の資源拡大については今後の課題としている。

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), および日本学術振興会 科学研究費補助金 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] Henning, J. L.: SPEC CPU2006 Benchmark Descriptions, *SIGARCH Comput. Archit. News*, Vol. 34, No. 4, pp. 1–17 (online), available from <http://doi.acm.org/10.1145/1186736.1186737> (2006).
- [3] Kora, Y., Yamaguchi, K. and Ando, H.: MLP-aware dynamic instruction window resizing for adaptively exploiting both ILP and MLP, *Proceedings of the 46th Annual International Symposium on Microarchitecture*, pp. 37–48 (2013).
- [4] Lebeck, A. R., Koppanalil, J., Li, T., Patwardhan, J. and Rotenberg, E.: A large, fast instruction window for tolerating cache misses, *Proceedings of the 35th Annual International Symposium on Microarchitecture*, pp. 59–70 (2002).
- [5] Mutlu, O., Stark, J., Wilkerson, C. and Patt, Y.: Runahead execution: an alternative to very large instruction windows for out-of-order processors, *Proceedings of the 9th Annual International Symposium on*

- High-Performance Computer Architecture*, pp. 129–140 (2003).
- [6] Pericas, M., Cristal, A., González, R., Jiménez, D. A. and Valero, M.: A decoupled KILO-instruction processor, *Proceedings of the 12th Annual International Symposium on High-Performance Computer Architecture*, pp. 53–64 (2006).
- [7] Sembrant, A., Carlson, T., Hagersten, E., Black-Shaffer, D., Perais, A., Seznec, A. and Michaud, P.: Long Term Parking (LTP): Criticality-aware Resource Allocation in OOO Processors, *Proceedings of the 48th Annual International Symposium on Microarchitecture*, pp. 334–346 (2015).
- [8] 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).
- [9] Srinivasan, S. T., Rajwar, R., Akkary, H., Gandhi, A. and Upton, M.: Continual flow pipelines, *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 107–119 (2004).
- [10] 塩谷 亮太, 五島 正裕, 坂井修一: プロセッサ・シミュレータ「鬼斬式」の設計と実装, 先進的計算基盤システムシンポジウム SACSIS 2009, pp. 120–121 (2009).