Vol.2012-ARC-198 No.10
2012/1/20

L1データ・キャッシュ・ミスに着目した命令発行キューの動的リサイジング

有松 優[†1] 塩谷亮太[†1] 安藤秀樹[†1]

発行キューを拡大することは，命令レベル並列 (ILP:instruction-level parallelism) の利用において重要である．しかし，大きな発行キューは電力を多く消費するという問題があり，また，拡大に対する性能向上率はしだいに低減する．よって，単純な拡大は電力効率が悪い．本論文では，大きな発行キューが必要となるL1データ・キャッシュ・ミス時に拡大を行うことでペナルティを隠蔽し，高いIPCを達成しつつ、一方で，そのようなイベントがない定常状態では電力効率を重視し，可能なだけ発行キューの縮小を行う動的リサイジング手法を提案する．SPEC2000ベンチマークを用い本手法の評価を行った結果，128エントリの発行キューを持つプロセッサに対し，SPECint2000では，約0.3〜1.5%の性能低下で，約50〜67%の発行キュー・サイズの削減率を，SPECfp2000では，約3〜7%の性能低下で，約50〜65%の発行キュー・サイズの削減を達成した．

1. はじめに

一般に，プログラムから命令レベル並列性 (ILP: instruction-level parallelism) を多く引き出すには，大きな発行キュー (IQ: issue queue) が必要となる．IQの消費電力はそのサイズに比例する一方で，IQ拡大に対するIPCの増加率は低減していく．このため，単純な拡大は，電力効率を低下させるという問題がある．

本論文では，高いIPCを達成しつつも，高い電力効率を得るために，IQを動的にリサイズする手法を提案する．具体的には，IQが大きくあるべき特別な瞬間を狙って拡大する．一方，そのような場合がない定常状態では電力効率を重視し，可能なら縮小し，必要サイズに実際のサイズをフィットさせる．本論文では，大きなIQが必要な特別な瞬間として，L1データ・キャッシュ(以下、L1Dキャッシュと記す) ミスが生じた時に着目する．L1Dキャッシュ・ミスが生じると，そのペナルティにより大きく性能が低下するが，これを軽減するには，ミスに依存しない独立な命令をできるだけ多く発行し，ペナルティを隠蔽することが重要である．独立な命令を見つけるには，大きなIQが必要であるから，ミスが生じた時に拡大する．これにより，高いIPCを実現する．逆に，ミスが生じていない定常状態では，あらかじめ定められたIQの末尾の領域に命令がなくなったら，IQへのサイズ要求は低下したとして，その領域をIQから削除し，縮小する．これにより，定常状態での電力効率を向上させる．

本論文の残りの部分は次のような構成となっている．まず，2節で関連研究について述べる．次に，3節で発行キューの動的リサイジング手法を提案する．そして4節で評価を行い，それを受けて5節で今後の課題を述べる．最後に6節で本論文をまとめる．

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

2. 関連研究

Folegnaniらは，性能への寄与が小さなIQの領域の使用を取りやめることによるIQのリサイジング手法を提案した[1]．この手法では，IQの末尾の領域のエントリから発行された命令のコミット数を数えることによりIPCへの寄与を計算し，それがあらかじめ定めた閾値より低ければ，その領域の使用を取りやめる．一方，拡大の利益があるかどうかの判断のために，定期的にIQを拡大し，性能寄与があるかどうかをチェックする．この手法は，IQの特定の領域の性能への寄与をモニターするという直接的な方法であるが，IQ拡大の明確な指針がないため，キャッシュ・ミスによる性能低下に対応できない．

PonomarevらはIQの占有率に着目した動的サイジングを提案した[4]．この手法では，一定期間のIQ内の平均命令数を調査し，その数が現在のIQサイズより一定数少なければ縮小を行う．また，IQがいっぱいであるためにディスパッチが停止したサイクル数を数え，その値があらかじめ定めた閾値以上となれば拡大を行う．IQ縮小のタイミングは一定期間の平均命令数の調査の後であり，また，拡大のタイミングも，一定期間ディスパッチが停止した後に行われるため，Folegnaniの手法同様，必要な時にタイムリーにIQを拡大することは困難である．

3. 命令発行キューの動的リサイジング

3.1 概　　要

本論文では，高いIPCを達成しつつも電力効率を向上させるため，以下の方針をとる:

- IQが大きくあるべき特別なイベントが生じれば拡大する．

Vol.2012-ARC-198 No.10
2012/1/20

- 特別なイベントがない定常状態では，必要に応じてIQを縮小する．

ここで，IQが大きくあるべき特別なイベントとして，L1Dキャッシュ・ミスとL2キャッシュ・ミスが考えられる(本論文では，キャッシュは2階層と仮定する)．一般に，これらのキャッシュ・ミスが生じると，大きなペナルティを被るが，IQが大きければ，そのペナルティを実質的に軽減できる．その理由は以下の2点である:

(1) ペナルティを被っている間，ミスとは独立な命令を実行し，ペナルティを隠蔽することができる．

(2) ミスが未解決の間に他のミスが生じれば，複数のミス処理を並列化することができ，実質的にミス1つ当たりのペナルティを削減することができる．

L1Dキャッシュ・ミスでは，ミス・ペナルティは10サイクル程度であるので，ミスした命令に依存する命令により，IQは数10エントリ程度が占有される．これに加えて，ミスに独立な命令を見つけるためにさらに数10エントリ必要と思われる．よって，IQに要求されるサイズは，数10エントリから100エントリ程度と考えられる．一方，L2キャッシュ・ミスでは，ミス・ペナルティは数100サイクル程度であるので，L1Dキャッシュ・ミスの際の議論と同様に考えれば，IQに要求されるサイズは数100から1000エントリ程度である．

さて，現在のプロセッサのIQサイズは，たとえば，Intel Sandy Bridgeでは，56エントリである[2]．世代が進むとIQは拡大されると考えらえるが，次世代のプロセッサのIQは現在のサイズの倍程度の128エントリ程度と推測される．そのようなサイズのIQでは，L1Dキャッシュ・ミスによる性能低下に対応可能であるが，L2キャッシュ・ミスのそれには対応できない．よって，本論文では，IQが大きくあるべき特別なイベントとして，L1Dキャッシュ・ミスに着目する．

図1に，L1Dキャッシュ・ミスの生起率とIQサイズ拡大による性能向上率の関係を示す．横軸は，L1Dキャッシュ・ミスを起こしたがL2キャッシュにヒットしたロードの割合(1K命令当たりのL1Dキャッシュ・ミスから1K命令当たりのL2キャッシュ・ミスを減じた値)であり，縦軸は，IQサイズを32から128にしたときのプロセッサの性能向上率である．プロセッサの構成は，後に表1で示す．ただし，IQ以外の資源の制約を除くため，リオーダ・バッファ，レジスタ・ファイル，ロード/ストア・キューは全て十分大きく1Kエントリとしている．同図からわかるように，弱いながらも正の相関があり，L1Dキャッシュ・ミス時にIQを拡大することは有効であることがわかる．

一方，縮小については次のように考えられる．特別なイベントが生じていない定常状態では，IQの必要サイズは小さく，小さくしてもIPCはあまり低下しないと考えられる．そこで，使われていないエントリがあるなら縮小し，必要サイズに実際のサイズをフィットさせる．これにより電力効率を向上させる．ただし，過剰に小さくなり大きな性能低下を招かないように，下限を与えておく．

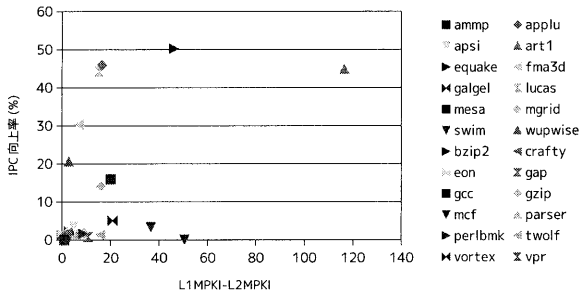


図1 L1Dキャッシュ・ミスの生起率とIQ拡大による性能向上の関係

3.2 過去の研究との違い

過去に発表された研究は，いずれも，IQサイズに対する必要度の「時間平均」に基づいてIQをリサイズしていた[1),4)]．しかし,「時間平均」に基づくため，急激に必要度が上昇する瞬間には適応できず，遅れて反応するという欠点がある．これに対して，我々の手法では，必要度が上昇する「瞬間」に拡大し要求に応え，そうでない間はサイズを抑え無駄な電力消費を回避するという点が特徴である．

図2を用いて，従来手法と我々の手法との違いを説明する．同図は，IQの必要サイズ，従来手法によるIQサイズ，本論文の手法によるIQサイズそれぞれの時間変化を描いている．L1Dキャッシュ・ミスが生じた瞬間からしばらくの間，IQの必要サイズは定常値より上昇する．必要度の時間平均に基づく従来手法では，その値があらかじめ定めた閾値を超える時間が経過した後でIQは拡大される．図からわかるように，L1Dキャッシュ・ミスによる急激な必要度上昇には対応できない．また，すでに必要度が定常状態に戻った時刻に拡大が行われれ，無駄に電力を消費する．これに対して，我々の手法では，L1Dキャッシュ・ミスが生じ，必要度が上昇したした瞬間にIQサイズを拡大し，タイムリーに対応する．また，必要度が低い定常状態ではそれに対応したサイズに戻り，無駄な電力を消費しない．

3.3 アルゴリズム

前節の考えに基づいたIQの動的リサイジングのアルゴリズムを，**図3**に疑似コードで示

Vol.2012-ARC-198 No.10
2012/1/20

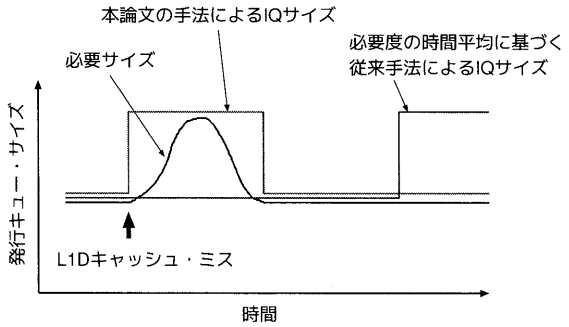


図 2 IQ サイズ必要度に対する従来手法と我々の手法の動作の違い

す．変数や定数の定義は以下の通りである:

- `cycle`: 現在のクロック・サイクル
- `tail`: IQ の使用されている領域の末尾のエントリ番号
- `miss`: L1D キャッシュ・ミス
- `checkpoint`: IQ 縮小のタイミング
- `interval`: IQ 縮小タイミングの間隔
- `delta`: リサイジング幅 (IQ 縮小/拡大の幅)
- `minsize`: 許容する IQ の最小サイズ
- `maxsize`: 許容する IQ の最大サイズ (IQ の物理的なサイズ)

実行中に，L1D キャッシュ・ミスが発生したサイクルでは，リサイジング幅 `delta` だけ IQ サイズを増加させる (5 行目)．ただし，許容する最大サイズ `maxsize` はこえないようにする．そして，次回の IQ 縮小のタイミングとして，現在のサイクルより `interval` だけ先のサイクルを `checkpoint` として記憶する (6 行目)．L1D キャッシュ・ミスが生じていないサイクルで，IQ 縮小タイミング `checkpoint` のサイクルとなった場合，縮小可能かをチェックする (7 行目)．ここで，縮小可能とは，IQ の現在の末尾から `delta` ほどの領域に命令がなく空があることである．縮小可能なら，IQ サイズを `delta` だけ縮小する (8 行目)．ただし，許容する最小サイズ `minsize` より小さくはしない．そして，現在のサイクルより `interval` だけ先のサイクルを `checkpoint` として記憶する (9 行目)．縮小できなかった場合，縮小できるようになるか，L1D キャッシュ・ミスが起こるまで，毎サイクル縮小可能かをチェックする (7 行目)．

```
1:  cycle = checkpoint = 0;
2:  tail = maxsize;
3:  foreach cycle {
4:    if (miss) {
5:      tail = min(tail + delta, maxsize);
6:      checkpoint = cycle + interval;
7:    } else if (cycle >= checkpoint && IQ[tail - delta:tail] is empty) {
8:      tail = max(tail - delta, minsize);
9:      checkpoint = cycle + interval;
10:   }
11: }
```

図 3 発行キュー動的リサイジングのアルゴリズム

4. 評　　価

4.1 評価環境

評価には，SimpleScalar Tool Set Version 3.0a[5] をベースに提案手法を実装したシミュレータを用いた．命令セットには DEC Alpha ISA を用いた．ベンチマーク・プログラムとして，SPEC2000 から整数系，浮動小数点系プログラムをそれぞれ 12 本ずつ使用した．バイナリは，Compaq C 及び Fortran コンパイラを用いて-fast -O4 のオプションでコンパイルした．入力には ref 入力を用い，シミュレーション時間が過大とならないよう，SimPoint[3] によって選択した 100M 命令を実行した．

評価の基準となるベース・プロセッサの構成を**表 1** に示す．また，**表 2** に，各ベンチマーク・プログラムにおけるロードの L1 データ・キャッシュ・ミス率 (MPKI: misses per kilo-instructions)，L2 キャッシュ・ミス率 (MPKI)，それらの差 (L1D キャッシュ・ミスしたが，L2 キャッシュ・ヒットした割合) を示す．

4.2 リサイジング・パラメータ

動的リサイジング・アルゴリズムにおけるパラメータは，3.3 節に示した通り，IQ 縮小タイミングの間隔 (interval)，リサイジング幅 (delta)，許容する IQ の最小サイズ (minsize)，IQ の許容する最大サイズ (maxsize) である．このうち，今回の実験では，maxsize はベース・プロセッサの IQ サイズ (128) とし，interval は L1D キャッシュ・ミス・レイテンシ (=

表 1 ベース・プロセッサの構成

Pipeline width	4-instruction wide for each of fetch, decode, issue, and commit
ROB	256 entries
LSQ	128 entries
Issue queue	128 entries
Physical register	256 for each of int and fp
Function unit	4 iALU, 2 iMULT/DIV, 4 fpALU, 2 fpMULT/DIV/SQRT
L1 I-cache	64KB, 2-way, 32B line
L1 D-cache	64KB, 2-way, 32B line, 2 ports, 2-cycle hit latency, non-blocking
L2 cache	2MB, 4-way, 64B line, 12-cycle hit latency
Main memory	300-cycle min. latency, 4B/cycle bandwidth
Branch prediction	16-bit history gshare 64K-entry PHT 10-cycle misprediction penalty

L2 ヒット・レイテンシ) である 12 サイクルとした．L1D キャッシュ・ミスからこの時間が経過すれば，L2 キャッシュにヒットするなら定常状態に入るからである．minsize，delta は変化させて測定した．**表 3** に，実験時に設定したこれらのパラメータの値を示す．

4.3 性能損失対 IQ サイズ削減率

4.3.1 ベンチマーク平均

図 4 に，IQ サイズ削減率と性能 (IPC) 低下率のベンチマーク平均の関係を示す．性能の基準はベース・プロセッサの性能である．5 本の折れ線グラフは，4 種の minsize の場合の動的リサイジング手法の場合と，IQ サイズを固定とした場合 (図の凡例では fix と表示) である．動的リサイジング手法の折れ線グラフの 4 つ点は右から，delta がそれぞれ，8，16，32，64 の場合である．固定サイズの折れ線グラフの 5 つの点は，右からエントリ数が 8，16，32，64，128 の場合である．図の右下の方ほど，電力効率が良い．

まず，SPECint2000 では，どのパラメータでも概して，動的リサイジングにより，小さな性能損失で大きな IQ サイズ削減がなされていることがわかる．minsize によって，IQ サイズ削減率と性能損失の間にトレードオフが見られるが，性能損失は小さく，顕著なものではない．minsize = 8 の時が最も性能低下が大きいが，それでも 4%程度であり，70%程度 IQ サイズを削減できる．このような大きな削減率が得られるのは，SPECint2000 は，L1D キャッシュ・ミスの少ないプログラムが多く，小さな IQ でも高い IPC が得られるためである．delta に対する性能感度はほとんどなく，IQ サイズ削減率への感度は非常に小さいことも特徴としてあげられる．固定サイズのグラフを見ると，性能低下が数%程度に抑えられる最小サイズは，32 エントリ程度であり，動的リサイジング手法は，minsize = 16，delta = 8 でそれをほぼ達成できる．

一方，SPECfp2000 では，minsize によって，IQ サイズ削減率と性能損失の間に顕著なトレードオフが見られる結果となった．minsize が小さいほど，性能損失が大きいが，IQ サ

表 2 メモリ・アクセスに関する統計

(a) SPECint2000

program	MPKI		L1 data MPKI
	L1 data	L2	− L2 MPKI
bzip2	9.4	0.7	8.8
crafty	2.1	0.1	2.0
eon	0.3	0.0	0.3
gap	3.3	1.8	1.5
gcc	3.1	1.7	1.4
gzip	9.1	0.1	9.0
mcf	112.9	62.5	50.4
parser	6.5	0.5	6.0
perlbmk	0.0	0.0	0.0
twolf	16.0	0.1	15.8
vortex	3.4	0.5	2.9
vpr	13.4	2.7	10.7

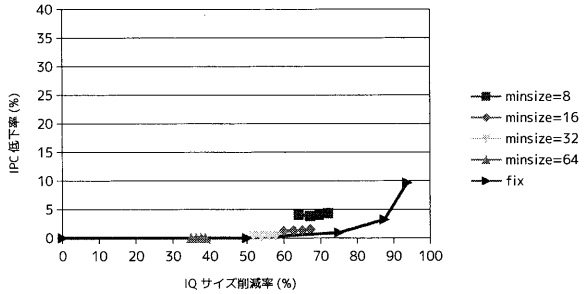
(b) SPECfp2000

program	MPKI		L1 data MPKI
	L1 data	L2	− L2 MPKI
ammp	20.7	0.7	20.1
applu	33.4	16.9	16.5
apsi	5.8	1.0	4.8
art	128.1	11.7	116.4
equake	72.1	26.1	46.0
fma3d	19.1	12.1	7.0
galgel	22.0	1.1	20.9
lucas	33.4	18.1	15.4
mesa	1.2	0.3	0.9
mgrid	22.7	6.7	16.1
swim	56.6	20.1	36.6
wupwise	5.2	2.4	2.8

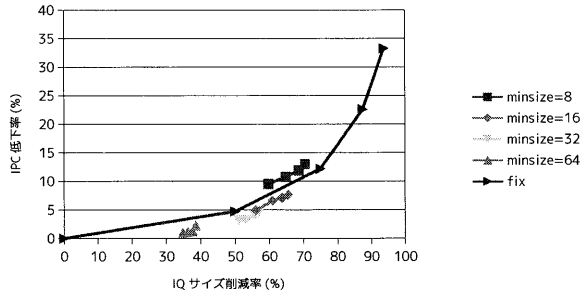
Vol.2012-ARC-198 No.10
2012/1/20

表 3 動的リサイジングにおけるパラメータの設定

minsize	8, 16, 32, 64
delta	8, 16, 32, 64



(a) SPECint2000



(b) SPECfp2000

図 4 IQ サイズ削減率と性能低下の関係 (平均)

イズ削減率も大きい．minsize が大きいほど，その逆の結果が生じる．SPECfp2000 は，プログラムに潜在する ILP が大きく，定常状態であまり小さな IQ にすることは好ましくないことがわかる．delta に対しても，同じようなトレードオフが見られ，delta が大きいほど，性能損失は小さいが，IQ サイズ削減率も小さい．これは，delta が大きいほど，IQ の削減予定の領域が大きくなり，そこから命令がなくなる確率が下がり，縮小機会が少なくなるためである．固定サイズのグラフを見ると，性能低下が数%程度に抑えられる最小サイズは，64 から 128 エントリの間にあり，動的リサイジング手法で，この小さな性能低下とサイズ削減率をほぼ満たすには，minsize = 32，delta ≥ 16 あるいは，minsize = 64 のパラメータが好ましい．

4.3.2 ベンチマーク毎の考察

前節では，ベンチマーク平均での議論を行ったが，動的リサイジングの評価としては，プログラム毎に最適サイズを選ぶことができるかどうかが問われる．本節の議論では，L1D キャッシュ・ミスかつ L2 キャッシュ・ヒット (以下，単に L1D キャッシュ・ミスと呼ぶ) の率の違いで 8 つのプログラムを選んで議論する．動的リサイジングのパラメータとしては，図 4 から良好な結果が得られた minsize = 16, 32 を選んだ．**図 5** に評価結果を示す．グラフの軸や折れ線グラフは，図 4 で説明した通りである．

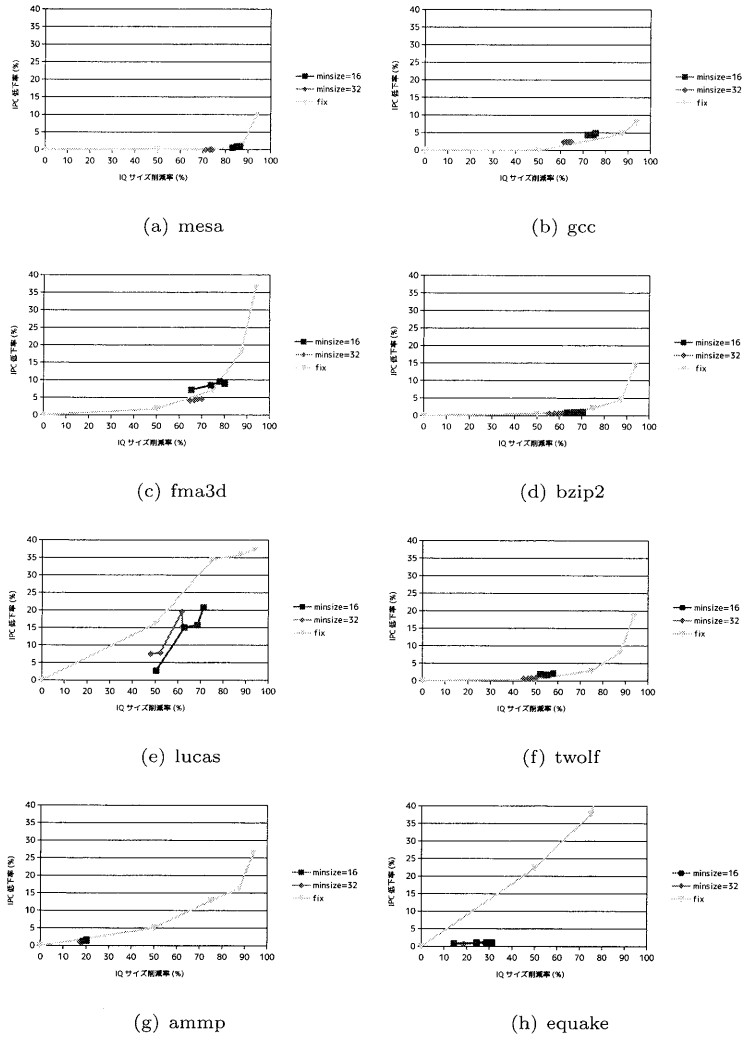
masa, gcc は，L1D キャッシュ・ミスが非常に少ないプログラムである．また，fix のグラフからわかるように，IQ サイズ要求は小さい．動的リサイジングでは，IQ 縮小の圧力が大きく，性能損失を非常に小さくして，サイズを大きく縮小できている．

fma3d, bzip2 は，L1D キャッシュ・ミスが比較的少ないプログラムである．*fma3d* では，IQ サイズ要求が小さくはないが，*bzip2* では小さい．動的リサイジングでは，IQ 縮小の圧力が大きく，IQ は小さくなる．その結果，*bzip2* では性能損失が少ないものの，*fma3d* では，パラメータを適切に選ばなければ，性能損失が大きくなる．

lucas, twolf は，L1D キャッシュ・ミスが比較的多いプログラムである．しかし，fix のグラフからわかることは，IQ サイズ要求は *lucas* では大きいが，*twolf* では小さい．動的リサイジングは，*twolf* では性能損失は少なく，IQ サイズを縮小できている．しかし，*lucas* では，パラメータ依存性が高く，minsize = 16, delta = 64 のときしかうまく適応できていない．

ammp, equake は，L1D キャッシュ・ミスが多いプログラムである．しかし，fix のグラフからわかることは，IQ サイズ要求は，*ammp* ではあまり大きくないが，*equake* では大きい．このため，動的リサイジングは，*equake* では，性能損失は少なく，IQ サイズをある程度縮小できている．しかし，*ammp* では性能損失は少ないものの，十分に IQ サイズを縮小できていない．

Vol.2012-ARC-198 No.10
2012/1/20



(a) mesa (b) gcc

(c) fma3d (d) bzip2

(e) lucas (f) twolf

(g) ammp (h) equake

図 5 IQ サイズ削減率と性能低下の関係 (ベンチマーク・プログラム毎)

5. 今後の課題

図 5 の fix のグラフからわかるように，プロセッサの現実的な構成の下では，図 1 での評価と異なり，L1D キャッシュ・ミスと IQ サイズ要求度の間の相関は弱くなる．これは，IQ より先に他の資源の不足によりストールすることがあるためと考えられる．よって，IQ 拡大に際には，L1D キャッシュ・ミス以外に他の資源不足についても考慮する必要があると思われる．

また，現在のアルゴリズムでは，定常状態の IQ サイズ要求は小さいものと仮定していており，単純に縮小圧力をかけている．しかし，定常状態の IQ サイズ要求は内在する ILP のよって変化する．よって，単純に縮小圧力をかけるのではなく，定常時に達成されると思われる IPC を考慮に入れる必要があると考えられる．

6. ま と め

本論文では，高い IPC を達成しつつも，高い電力効率を実現する IQ の動的リサイジング手法を提案した．本手法では，特別に大きな IQ が要求される時のみ拡大し，定常的には縮小の圧力をかける．上記特別な瞬間として，L1D キャッシュ・ミスの生起に着目した．SPEC2000 ベンチマークを用い本手法の評価を行った結果，128 エントリの IQ を持つプロセッサに対し，SPECint2000 では，約 0.3〜1.5%の性能低下で，約 50〜67%の IQ サイズ削減率を，SPECfp2000 では，約 3〜7%の性能低下で，約 50〜65%の IQ サイズ削減率を達成した．以上のように電力効率を向上させることができたが，実行させるプログラムに応じて最適にサイズを必要最低限のサイズにフィットさせることは，まだ十分に達成しておらず，今後の研究が必要である．

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 22500045）による補助のもとで行われた．

参 考 文 献

1) Folegnani, D. and González, A.: Energy-effective issue logic, *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pp.230–239 (2001).
2) Gwennap, L.: Sandy Bridge Spans Generations, *Microprocessor Report* (9/27/10-01, 2010).
3) Hamerly, G., Perelman, E., Lau, J. and Calder, B.: SimPoint 3.0: Faster and More

Vol.2012-ARC-198 No.10
2012/1/20

Flexible Program Phase Analysis, *The Journal of Instruction-Level Parallelism*, Vol.7, pp.1–28 (2005).

4) Ponomarev, D., Kucuk, G. and Ghose, K.: Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources, *Proceedings of the 34th Annual International Symposium on Microarchitecture*, pp. 90–101 (2001).

5) `http://www.simplescalar.com/`.