MLP に着目したパイプライン化発行キューの動的サイジング

甲 良 祐 也[†,☆] 安 藤 秀 樹[†]

メモリ・インテンシブなプログラムの実行時間を短くするアプローチとして，メモリ・レベル並列性 (MLP: memory-level parallelism) の利用は有効である．MLP 利用の 1 手法として，積極的なアウト・オブ・オーダ実行がある．このためには大きな発行キューが必要であるが，そのような発行キューはクロック速度を低下させるという問題がある．発行キューをパイプライン化すればこの問題を回避することができるが，発行が遅延し計算インテンシブなプログラムでは IPC が著しく低下するという問題が生じる．本論文では，MLP が利用可能なときのみ発行キューを拡大する動的サイジング手法を提案する．本手法は，MLP が利用可能かどうかを最終レベル・キャッシュのミスの生起によって予測あるいは判断し，発行キューをサイジングする．SPEC2000 ベンチマークを用いて評価した結果，提案の動的サイジング手法を用いれば，評価したほとんどのプログラムでサイズを固定した場合での最善の性能とほぼ同等かそれ以上の性能を達成できることを確認した．平均では，サイズを固定した場合での最善の性能より SPECint2000 で 0.7%，SPECfp2000 で 10.2%，両ベンチマーク・スイートで 11.1%高い性能を達成した．

MLP-Aware Dynamic Sizing of Pipelined Issue Queue

Yuya Kora[†,☆] and Hideki Ando[†]

Exploiting memory-level parallelism (MLP) is an effective approach to reduce execution time of memory-intensive programs. One of schemes to exploit MLP is aggressive out-of-order execution. For this, a large issue queue is required, but it degrades the clock rate. Although pipelining the issue queue solves this problem, it delays instruction issue and thus degrades IPC in compute-intensive programs dramatically. This paper proposes a dynamic sizing scheme that enlarges the issue queue only when MLP is exploitable. Our scheme changes the size of the issue queue by predicting or determing whether or not MLP is exploitable, based on occurence of the last-level cache misses. Our evaluation results using the SPEC2000 benchmark programs show that, in most programs, our dynamic sizing scheme achieves as well or better performance, compared with the best performance in a fixed-size issue queue. On an average, a processor with our scheme achieves 0.7%, 10.2%, or 11.1% higher performance, in SPECint2000, SPECfp2000, or both benchmark suites, respectively, over the best performance in a fixed-size issue queue.

1. はじめに

プロセッサとメモリの間の速度差は非常に大きく，一般に，メモリ・ウォールと呼ばれている．メモリ・ウォールは，メモリ・インテンシブなプログラムの性能を著しく制限している．積極的なアウト・オブ・オーダ実行は，この問題を解決する 1 手法である．これは，メモリ・レベル並列性 (MLP: memory-level parallelism) を利用し，メモリ・レイテンシを隠蔽しようとするものである．

アウト・オブ・オーダ実行でメモリ・ウォール問題を解決するには，プロセッサがサポートする in-flight 命令を，現在の商用プロセッサで実現されているその数より大幅に増加させる必要がある．このために重要なハードウェアとしては、リオーダ・バッファ(ROB: reorder buffer)，発行キュー，ロード/ストア・キュー (LSQ: load/store queue)，レジスタ・ファイルがある．しかし、これらのサイズを増加させると，一般には，クロック・サイクル時間を悪化させるという問題がある．本研究では，主として発行キューに焦点を当てる．

動作時間が長くクロック速度に悪影響を与える論理

† 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University
☆ 現在，ローム (株)
Presently with Rohm Co., Ltd.

において，その悪影響を除く一般的手法としてパイプライン化がある．発行キューをパイプライン化することは回路的には可能であるが，命令レベル並列 (ILP: instruction-level parallelism) を有効に利用できなくなるという問題がある．具体的には，発行された命令の結果タグは直後のサイクルに発行キューに放送され，依存する命令を即座に発行できなければ，互いに依存のある命令を連続したサイクルで発行できなくなる．このため，MLP の利用機会が乏しい計算インテンシブなプログラムでは IPC が低下する．

以上をまとめると，次のようになる．メモリ・インテンシブなプログラムでは，主記憶レイテンシが実行時間を大きく支配するので性能上 ILP 利用は重要でなく，MLP を利用することが有効である．このためパイプライン化した大きな発行キューは有効である．逆に，計算インテンシブなプログラムでは，計算が実行時間を支配するので ILP 利用は重要であるが，MLP の利用機会が乏しいので，パイプライン化した大きな発行キューは不利である．

以上を踏まえ本論文では，MLP が利用可能と予測した時に発行キューを拡大し (この時，パイプライン段数を増加させる) 性能向上を図り，MLP が利用不能となった時点で縮小する (この時，パイプライン段数を減少させる) 発行キューの動的サイジング手法を提案する．MLP 利用可能かどうかは，L2 キャッシュ・ミス (本論文では，最終レベル・キャッシュを L2 とする) が 1 度でも起こった場合にそのように予測する。これは，L2 キャッシュ・ミスが時間的にかたまって生じる傾向があることを利用するものであり，1 度起これば，その後も早期に生じると期待できるからである．一方，MLP 利用不能の判断は予測ではなく実際にそのようになった時をもって判断する．すなわち，最後に L2 キャッシュ・ミスが生じてから主記憶レイテンシだけ経過したときに MLP 利用不能と判断する．

本論文ではまず，2 節で関連研究について述べる．次に，3 節で発行キューの構成とパイプライン化の方法について説明する．4 節で発行キューの動的サイジングを提案し，そして 5 節で評価を行う．最後に 6 節でまとめる．

2. 関 連 研 究

2.1 発行キューのサイジング

Folegnani らは，性能への寄与が小さな発行キューの領域の使用を取りやめることによる発行キューのサイジング手法を提案した[1]．この手法では，発行キュー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，定期的に発行キューを拡大し，性能寄与があるかどうかをチェックする．この手法は，発行キュー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，発行キュー拡大の明確な指針がないため，MLP 利用を狙い急激な変動に対しタイムリーに発行キューを拡大することが困難である．

Ponomarev らは発行キューの占有率に着目した動的サイジングを提案した[2]．この手法では，一定期間での発行キュー内の平均命令数を調査し，その数が現在の発行キュー・サイズのより一定数少なければ縮小を行う．また，発行キューが一杯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．発行キュー縮小のタイミングは一定間隔であるのに対して，拡大は，発行キューが一杯になった任意のタイミングで行われるので，MLP 利用に適している．しかし，発行速度よりディスパッチ速度を高く設計するのが一般的であるから，キャッシュ・ミスが生じなくてもいずれは発行キューは一杯になり，MLP を利用できない状況でも発行キューを拡大してしまうという問題がある．

2.2 発行キューの拡大

Stark らは，間接的に依存する命令により発行キューで待ち合わせている命令を投機的にウェイクアップすることにより，発行キューをパイプライン化し，サイズを拡大する手法を提案した[3]．間接的に依存する命令を特定し，ウェイクアップするタイミングを計算するために，フロントエンドの論理が非常に複雑化するという問題がある．このため，パイプラインの深さは 2 段程度で制限される．同様のアイデアは，加藤らによっても試された[4]．間接的に依存する命令を履歴ベースで見つける手法であり，論理の複雑度は下げられ，より深いパイプライン化が可能である．

Brown らは，選択論理を介さず，ウェイクアップ論理単独で発行のループを構成することによる投機発行により，発行論理をパイプライン化する手法を提案した[5]．選択論理を介さないことにより，発行後実際には選択されず実行できない命令が生じ，かつそれに依存する命令も誤って発行される投機ミスが生じうる．このため，複雑な再発行機構が必要となる．

Hrishikesh らは，発行キューをセグメントに分け，パイプライン化する構成を提案した[6]．古いセグメントほどパイプライン化の遅延を受けないよう工夫されているが，タグ RAM からは同時に選択された命令の

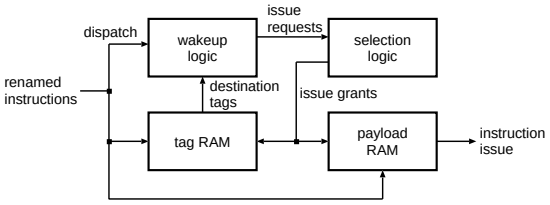


図 1 発行キューの構成

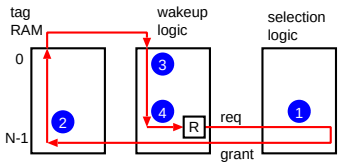


図 2 発行キューのクリティカル・パス

タグを読み出す必要があるため (さもなければ，ファンインが発行キュー・サイズのタグ線の調停器が別途必要でこれは選択回路と複雑度が等しい) 残念ながら古いセグメントもパイプライン化の不利を被る．

Brekelbaum らは，発行遅延を伴う大きな発行キューと小さく 1 サイクルで発行できる発行キューを組み合わせる階層化手法を提案した[7]．レディとなった若い命令は大きな発行キューから発行されるが，レディでない古い命令は小さな発行キューに移され，レディになるのを待つ．小さな発行キューへのレディでない古い命令を移動するという複雑な論理が必要となる．また，大きな発行キューは計算インテンシブなプログラムには無駄であり，電力を浪費する．

2.3 大きな発行キューの代替手法

Lebeck らは，L2 キャッシュ・ミスでストールする命令及びそれに依存する命令を発行キューから抜き出し，WIB(waiting instruction buffer) と呼ぶバッファに退避させることにより，発行キューのエントリを有効活用する手法を提案した．しかし，実際に発行キューのエントリを有効活用するには，退避した命令が使用していたエントリを詰めるための高度な発行キューのコンパクション[8] が必須であり，発行キューが複雑化する．また，WIB と発行キューの間で命令を移動させる必要があり，ディスパッチ，発行バンド幅を圧迫する．

Mutlu らは，L2 キャッシュ・ミスでストールしている間，実質的な実行を停止し，MLP 利用のためだけに後続の命令を実行する runahead 実行と呼ぶ手法を提案した[9]．runahead 実行中にはプロセッサ状態を更新する実質的な命令実行を行えないため，大きな発行キューを持つ場合に性能を損失する。

3. 発行キューのパイプライン化

提案の手法は，発行キューのパイプライン化の方法として，2.2 節で述べたようなアーキテクチャ支援により工夫されたものではなく，単純に回路的に直接パイプライン化することを前提としている．本節では，その回路について述べる．

発行キューは，**図 1** に示すようにウェイクアップ論理，選択論理，タグ RAM，ペイロード RAM からなる．ウェイクアップ論理の各エントリは，対応する命令の 2 つのソース・オペランドのタグとデータ依存の状態 (解決か未解決か) を示すレディ・フラグを保持する．両方のデータ依存が解決した場合，選択論理に発行要求を出す．選択論理では，資源競合を考慮し，発行許可を出す．発行許可を受け，ペイロード RAM から対応する命令の情報が出力され，発行が完了する．発行許可信号は同時に，タグ RAM にも出力され，発行される命令のデスティネーション・タグを得る．これをウェイクアップ論理の全エントリに放送し，レディ・フラグを更新する．回路の詳細については，文献 10) を参照されたい．

発行キューのクリティカル・パスは，ウェイクアップ論理→選択論理→タグ RAM を経てウェイクアップ論理に戻るパスである．**図 2** に，1 サイクルで動作する発行キューのレイアウトとその上でのクリティカル・パスを示す．クリティカル・パスは，発行キューの末尾のエントリのレディ・ビットを保持する FF(R と記された四角) から出発し，発行要求信号が送られ，選択論理 (マーク (1)) を経由して発行許可信号をタグ RAM に送り，タグがビット線 (マーク (2)) を経由して出力され，ウェイクアップ論理の先頭から末尾に向かってタグがドライブされ (マーク (3))，末尾のエントリでタグ比較が行われ (マーク (4))，レディ・ビットを保持する FF に入るまでのパスである．発行キュー・サイズを拡大すると，このクリティカル・パスの遅延 IQ_delay がクロック・サイクル時間 $cycle\_time$ より長くなる．クロック・サイクル時間を伸ばさないためには，$S = \lceil IQ\_delay / cycle\_time \rceil$ 段にパイプライン化しなければならない．

図 3 に，$S = 3$ 段にパイプライン化する場合のパイプライン・レジスタの挿入例を示す (発行幅 4)．小さな四角が FF である．クリティカル・パス (赤の線) には，レディ・ビットを保持する必ず必要な FF(R でマークされている) 以外に 2 つの FF(黒とピンクの四角) を挿入している．挿入位置は，パイプライン・レジスタ間の信号遅延が，クロック・サイクル時間を超えないように決定する．クリティカル・パス以外のパ

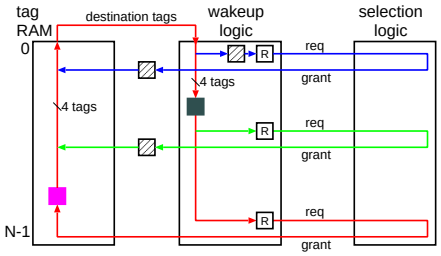
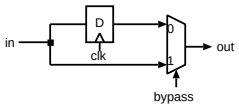
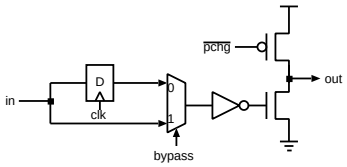


図 3 3 段にパイプライン化した発行キュー



(a) 静的配線の場合



(b) 動的配線の場合

図 4 バイパス回路付き FF

スには，その信号タイミングがクリティカル・パスの信号タイミングと合い，発行キューが論理的に正しい動作をするよう，FF(ハッチングされた四角) を挿入する．

異なる発行キュー・サイズで，パイプライン段数が 3 段以外の場合も同様にしてパイプライン・レジスタを挿入する位置を決定する．そして，ある特定のサイズ/パイプライン段数 S で発行キューを動作させるときには，その段数のときに挿入を決定したパイプライン・レジスタのみ有効とし，その他のパイプライン・レジスタはバイパスする．図 3 の例では，R のマークがない四角で表された FF は，$S = 3$ 以外の段数の時はバイパスされる．**図 4**(a) に，静的配線に挿入する FF (黒およびハッチングされた四角) を，(b) に動的配線に挿入する FF(ピンクの FF) の回路を示す．この他，発行キューの使用しない領域に伸びるウェイクアップ論理のタグ線のゲーティング (AND 回路による) とタグ RAM のビット線のゲーティング (トランスミッション・ゲートによる) が，遅延を不要に伸ばさないために必要である．

パイプライン化を行えば，どのような大きさの発行キューも実現できそうであるがそうではない．なぜなら，選択論理についてはパイプライン化することが困難であるからである．ウェイクアップ論理から発行要求が選択論理に出力され，それが許可されたかどうかを 1 サイクル以内にウェイクアップ論理が受けとらなければ，次のサイクルに再び発行要求を出すべきかどうかわからない．よって，選択論理はパイプライン化はできず，選択論理の遅延によって発行キューの最大サイズが決まる．

4. パイプライン化発行キューの動的サイジング

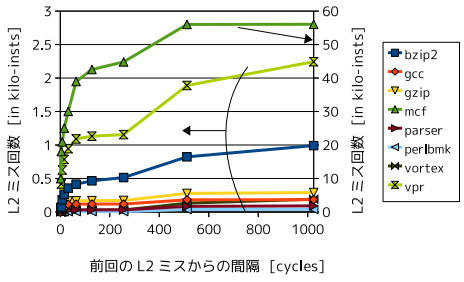
1 節で述べたように，利用できる MLP が存在する場合は大きな発行キューが有効であるが，そうでないときはパイプライン化による損失を受ける．本節では，MLP に着目した発行キューの動的サイジング手法を提案する．

4.1 発行キューの拡大

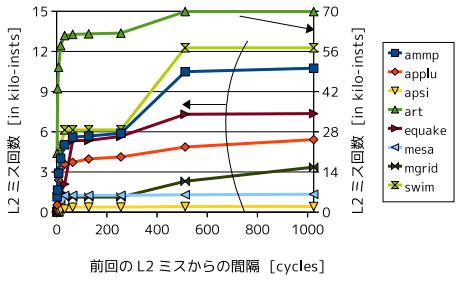
L2 キャッシュ・ミス (以下，単にキャッシュ・ミスと呼ぶ) により主記憶よりデータを読み出している間に，他のキャッシュ・ミスが生じれば，読み出しをオーバラップでき実効的にミス・レイテンシを削減できる．できるだけミスをオーバラップさせるには，早期に多くのロードを発行する必要があり，発行キューの拡大が有効である．

一般に，キャッシュ・ミスは時間的にかたまって生じる傾向がある．これは，プログラム実行のフェーズ変化に応じてメモリ・アクセスの局所性が低い瞬間があるためと考えられる．**図 5** に，キャッシュ・ミス間隔に対するミス回数の累積分布を示す (プロセッサ構成は，後に示す表 2 のとおりである)．*mcf* と *art* は，他のプログラムと比較してキャッシュ・ミス回数が著しく多く左の縦軸ではスケールに入らないので，縦軸を別途右に設けている．横軸の値が x のとき縦軸の値が y とは，前回のキャッシュ・ミスからの間隔が x サイクル以下であったミスの回数は，1000 命令当たり y 回であったことを示す．同図より，グラフは急峻に立ち上がっており，多くのミスが 32〜64 サイクル以内の間隔で起こっていることがわかる．この事実に基づき，本手法では，キャッシュ・ミスが 1 度生じたら，今後もしばらくキャッシュ・ミスが生じ続けると予測し，発行キューを拡大する．

拡大の方法は次のとおりである．今，クロック速度を悪化させず，パイプライン段数 S で動作する発行キューの最大サイズを $IQS(S)$ とする．現在，パイプライン段数 S_{curr} で，発行キュー・サイズ $IQS(S_{curr})$ で動作しているとする．拡大と判断した場合は，パイプライン段数を $S_{curr} + 1$ とし，発行キュー・サイズを $IQS(S_{curr} + 1)$ に拡大する (現在がすでにサイジ



(a) SPECint2000



(b) SPECfp2000

図 5 L2 キャッシュ・ミス間隔に対するミス回数の累積分布

ング可能な最大サイズなら，そのサイズにとどめる)．

4.2 発行キューの縮小

発行キューが大きい状態で MLP が利用できなくなると，発行キューのパイプライン化による ILP 利用上の損失が性能低下をもたらす．最後にキャッシュ・ミスが生じてから主記憶レイテンシだけ経過したときに MLP は利用不能となるから，この時点で発行キューを縮小する．縮小方法は，拡大方法と単純に逆であり，パイプライン段数を $S_{curr}-1$ とし，発行キュー・サイズを $IQS(S_{curr})$ から $IQS(S_{curr}-1)$ に縮小する (現在がすでにサイジング可能な最小サイズなら，そのサイズにとどめる)．縮小においては，縮小により削除される領域に命令が存在しなくなるまで命令の発行キューへの挿入をストールし，実際に縮小を行う．

4.3 電力上の利点

これまで性能上の利点を述べてきたが，動的サイジングは電力上の利点もある．文献 1) によれば，発行キューが消費する電力はプロセッサが消費する電力の 25%を占め，高い電力効率 (性能当たりの電力が低いこと) が要求される．発行キューは主に以下の回路で電力を消費し，動的サイジングを用いることにより電力を抑制することができる．

- タグ比較器: 結果タグと発行キューで待ち合わせている命令のソース・タグを比較するウェイクアップ論理の中の回路である．一般に，高速化のためにダイナミック回路で構成される[11]．タグが不一致の時に電力が消費されるが，ほとんどのタグ比較の結果は不一致なので，非常に多くの電力が消費されることになる．文献 1) によれば，発行キューの消費電力の 63%がこれにより費やされる．これに対し，動的サイジングでは，使用されなくなったエントリで，比較器のプリチャージを抑制するようにすれば，電力は消費されない．
- 選択論理: ウェイクアップ論理からの発行要求から，発行幅以下の要求を許可する回路である．調停回路[11] やプレフィクス・サム回路[12] で実装する方法が発表されている．前者は，4 つの要求から 1 つを選ぶ小さな調停器をツリー状につないで構成される．後者は，加算器をツリー状につないで構成される．これらの回路を静的回路で構成する場合，使用されないエントリからの発行要求は常に 0 となるので，回路を構成するトランジスタのスイッチング回数が減少し，電力は抑制される．動的回路で構成する場合，使用されないエントリからの発行要求を受ける回路のプリチャージを抑制すれば，電力は消費されない．

以上，動的電力について述べたが，リーク電力については，使用されないエントリの回路を Vdd ゲーティングによって電源を遮断すれば，抑制できる．

4.4 発行キュー以外の重要な資源のサイズ

in-flight 命令数を決める資源として，発行キュー以外に LSQ と ROB がある．レジスタ・リネーミングの方法によっては，レジスタ・ファイルも決定要素となる．発行キューを拡大した場合，これらの資源もバランスをとって拡大する必要がある．

LSQ の実装には種々のものが提案されているが (例えば，文献 13)，14))，発行キューと同じく基本的に FIFO なので，パイプライン化しサイジング可能である．ただし，パイプライン段数分発行が遅延するという悪影響が生じる．

ROB も，発行キューと同じく基本的に FIFO なので，サイジング可能である．パイプライン化による悪影響についてはレジスタ・リネーミングの方式に依存する．Intel Pentium 4 のように，物理レジスタが ROB とは別途存在するタイプでは，ROB のパイプライン化については悪影響は生じないと考えられる．なぜなら，フロントエンドで開始される割り当てられたエントリの初期化は実行終了までに終了すればよいので，その遅延は隠蔽される．また，書き込みやコミットの遅延は命令のレイテンシに影響を与えない．一方，Intel P6 アーキテクチャのように，ROB のエントリ

表 1 ベンチマーク・プログラム

(a) SPECint2000

program	misses/K-insts		main memory	is memory-	branch
	L1 data	L2	access rate	intensive?	mispreds/K-insts
bzip2	12.0	1.1	0.5%	no	10.89
gcc	77.9	0.2	0.1%	no	1.89
gzip	8.9	0.3	0.1%	no	13.52
mcf	127.9	56.1	16.2%	yes	24.80
parser	17.2	0.1	0.0%	no	7.32
perlbmk	3.4	0.0	0.0%	no	4.56
vortex	2.4	0.3	0.1%	no	0.38
vpr	11.2	2.4	0.8%	yes	8.86

(b) SPECfp2000

program	misses/K-insts		main memory	is memory-	branch
	L1 data	L2	access rate	intensive?	mispreds/K-insts
ammp	33.7	10.8	3.8%	yes	1.25
applu	11.0	5.4	2.9%	yes	0.28
apsi	1.3	0.4	0.2%	no	0.34
art	159.4	69.9	27.6%	yes	1.10
equake	19.4	7.5	2.6%	yes	1.79
mesa	3.5	1.4	0.5%	no	2.12
mgrid	11.2	3.4	0.9%	yes	0.07
swim	34.3	12.3	5.4%	yes	0.01

にリネーム・レジスタを保持するタイプでは，レジスタ数は可変にできるが，レジスタの読み出しに ROB のパイプライン段数分の遅延を被り，フロントエンドのパイプラインが長くなり分岐予測ミス・ペナルティが増加するという悪影響を被る．

Pentium 4 型のレジスタ・リネーミング手法では，レジスタ・ファイルは発行キューと共に拡大する必要があるが，そのサイジングは困難である．なぜなら，生存を開始するエントリも終了するエントリも空間的にランダムに生じるからである．したがって，アクティブな発行キューのサイズに関わらず，発行キューの最大サイズにバランスしたサイズで維持せざるをえない．大きなレジスタ・ファイルの実装方法には種々のものが提案されているが (階層化[15),16)]，バンク化[17)])，本論文では，Intel Pentium 4 型のレジスタ・リネーミング手法を仮定し，かつ単純なパイプライン化を仮定する．これによりクロック速度に悪影響を与えないが，深いパイプラインにより分岐予測ミス・ペナルティが増加し IPC に悪影響を与える．

5. 評　　価

5.1 評価環境

SimpleScalar Tool Set Version 3.0a をベースにシミュレータを作成し，評価した．命令セットは，MIPS R10000 を拡張した SimpleScalar/PISA である．ベンチマーク・プログラムとして SPECint2000 と SPECfp2000 からそれぞれ 8 本ずつ，計 16 本のプログラムを使用した．バイナリは，gcc ver.2.7.2.3 を用い，-O6 -funroll-loops のオプションでコンパイルし作成した．**表 1** に，後述するベース・プロセッサでの実行におけるロードのメモリ・アクセスに関する統計及び分岐予測ミス率を示す．SPECint2000 では，*mcf, vpr* のみメモリ・インテンシブであるのに対し，SPECfp2000 では *apsi, mesa* を除いた全てのベンチマークがメモリ・インテンシブである．

評価の基準となるベース・プロセッサの構成を**表 2** に示す．命令発行は 1 サイクルで行える．

5.2 評価モデル

ベース・プロセッサの構成を修正した大きな発行キューを持つ次のモデルを評価した．

- **固定サイズ・モデル**: 発行キューは S 段にパイプライン化されており，命令発行に S クロックを要する．発行キューは後に**表 3** に示すサイズに固定されている．
- **動的サイジング・モデル**: 固定サイズ・モデルと同様に発行キューはパイプライン化されている．発行キューは，物理的には表 3 に示す最大サイズのものを持っているが，提案手法によりサイズを動的に変更する．
- **理想モデル**: 発行キューのサイズは固定サイズ・モデルのそれと同じであるがパイプライン化されておらず，1 クロックで動作する．それによるクロック速度への影響はないとする．

表 2 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	128 entries
Issue queue	64 entries
LSQ	64 entries
Physical register	128 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, 8B/cycle bandwidth
Branch prediction	hybrid of 16-bit history 64K-entry PHT gshare + 2K-entry bimodal with 4K-entry choice predictor, 10-cycle misprediction penalty

表 3 S 段パイプライン化発行キューに許されるエントリ数

S	1	2	3	4
エントリ数	64	256	480	544

5.3 発行キュー・サイズ

パイプライン化した 4 命令発行の発行キューで，クロック・サイクル時間を悪化させない最大エントリ数を，HSPICE を用いた回路シミュレーションにより求めた[10]．32nm LSI 技術を仮定している．配線遅延が全体の遅延に大きな影響を及ぼすので，長い配線には積極的にリピータを挿入している．ベース・プロセッサの発行キューの遅延によってプロセッサのクロック・サイクル時間が決定されると仮定する．

3 節で述べたように，発行キューの最大サイズは選択論理の遅延で決まるので，ベース・プロセッサの発行キューの遅延時間で動作可能な最大サイズの選択論理を求め，そのサイズを発行キューの最大サイズとした．このとき，ウェイクアップ論理とタグ RAM は合わせて 3 段にパイプライン化されなければならないことが回路シミュレーションによりわかった．よって，最大サイズの発行キューのパイプライン段数は 4 である．

4 段未満のパイプライン構成での発行キュー・サイズは，3 節で述べたようにして求まる．パイプライン段数と対応する発行キューのサイズを表 3 に示す．

5.4 ROB，LSQ，レジスタ・ファイル

固定サイズ・モデルで，発行キューのサイズをベース・プロセッサのそれの N 倍にしたときは，それとバランスをとるために，同時に ROB，LSQ，レジスタ・ファイルを N 倍に拡大する．動的サイジング・モデルにおいては，これらは物理的には最大サイズである $IQS(S=4)/IQS(S=1)$ 倍に拡大するが，4.4 節で述べたように，ROB と LSQ については発行キューと共に動的にサイジングする．レジスタ・ファイルについてはサイジングしない．同節で述べたように，これらの資源を拡大したときは，パイプライン化によりクロック速度への悪影響を避けることができるとする．以上のように，これらの資源の拡大倍率は発行キューのそれと同率と単純に仮定したが，サイズとパイプライン段数の正確な関係は，構成を仮定し，回路シミュレーションにより求める必要がある．今後の課題としたい．

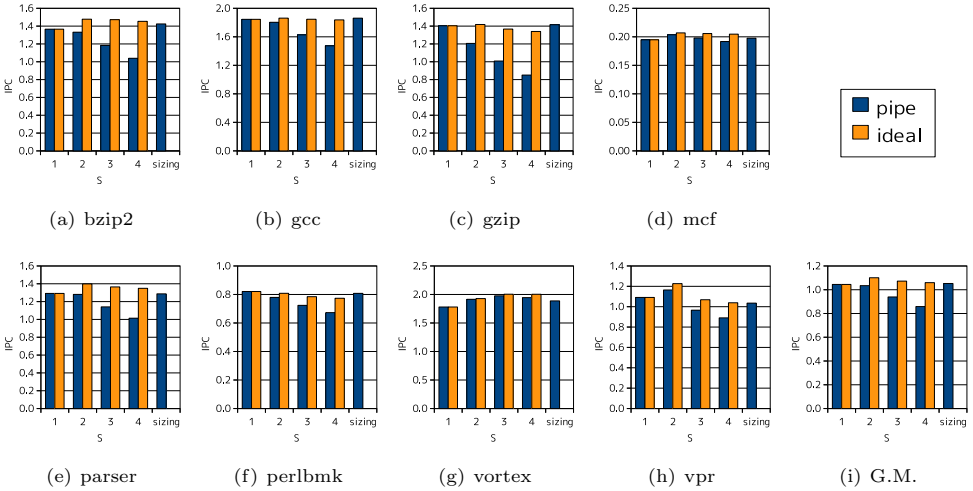
4.4 節で述べたように，LSQ においては，パイプライン化によりその段数分発行が遅延する．一方，ROB のパイプライン化は実行に何ら影響を与えないとする．

レジスタ・ファイルのパイプライン化は，分岐予測ミス・ペナルティを増加させる．ここで，クロック・サイクル時間をベース・プロセッサの発行キューの遅延 T_{base} とすると，S 段パイプライン化発行キューを持つ固定サイズ・モデルの分岐予測ミス・ペナルティは，$BMP_{base} + (S-1) + (\lceil T_{RF}(S)/T_{base} \rceil - \lceil T_{RF}(1)/T_{base} \rceil)$ である．ここで，BMP_{base} はベース・プロセッサの分岐予測ミス・ペナルティ，$T_{RF}(S)$ は S 段にパイプライン化した発行キューに対応するレジスタ・ファイルのアクセス時間である．T_{base} は，回路シミュレーションにより得，$T_{RF}(S)$ は CACTI 6.5[18] を用いて得た．この結果，$S = 2, 3, 4$ のとき分岐予測ミス・ペナルティは，それぞれ 11, 13, 14 サイクルとなった．

動的サイジング・モデルでの分岐予測ミス・ペナルティは，レジスタ・ファイルが最大サイズで固定されているので，前述の分岐予測ミス・ペナルティの式において，第 3 項が $S = 4$ のときの値で固定される．これにより分岐予測ミス・ペナルティは，$S = 1, 2, 3, 4$ のときそれぞれ，11, 12, 13, 14 サイクルとなる．

5.5 IPC

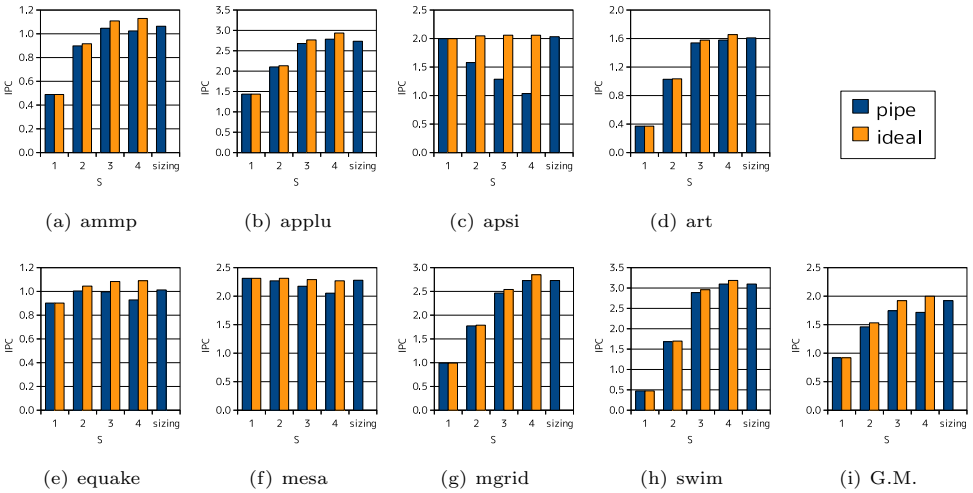
図 6, **図 7** に，それぞれ SPECint2000, SPECfp2000 における IPC の測定結果を示す．青色の棒グラフは，固定サイズ・モデル (横軸が S =1〜4 の棒) または動的サイジング・モデル (横軸が「sizing」の棒) の IPC を示しており，オレンジ色の棒グラフは理想モデルの IPC を示している．なお，$S = 1$ での 2 本の棒グラフについては，固定サイズ・モデルと理想モデルで構成上の差違はなく，単純にベース・プロセッサの IPC である．



(a) bzip2 (b) gcc (c) gzip (d) mcf

(e) parser (f) perlbmk (g) vortex (h) vpr (i) G.M.

図 6 SPECint2000 の IPC



(a) ammp (b) applu (c) apsi (d) art

(e) equake (f) mesa (g) mgrid (h) swim (i) G.M.

図 7 SPECfp2000 の IPC

グラフよりまず言えることは，動的サイジング・モデルは，*vpr* を除くどのプログラムにおいても，固定サイズ・モデルの S =1〜4 での性能の中で最も高い性能とほぼ同等かそれを上回る性能を達成しているということである (*vpr* については，5.6 節で議論する)．幾何平均では，固定サイズ・モデルで最善の S での性能を SPECint2000 で 0.7%，SPECfp2000 で 10.2%上回っている．また，**図 8** に示すように，両ベンチマーク・スイートの幾何平均では，固定サイズ・モデルの最善の性能を 11.1%上回っており，この結果，ベース・プロセッサに対し 45.1%の性能向上を達成することができた．両ベンチマーク・スイートにおいて，SPECint2000，SPECfp2000 個別の評価における固定サイズの最善からの改善率より大きな改善率が得られたのは，両ベンチマーク・スイートを合わせることにより，計算インテンシブなプログラムとメモリ・インテンシブなプログラムの混在度が高くなったためである．すなわち，その混在度が高いほど，動的サイジング・モデルにおける適応能力の有効性が表面に現れるからである．

また，動的サイジング・モデルは，パイプライン化発行キューのデメリットがほとんどない (分岐予測ミス・ペナルティが増加するというデメリットはある) 理想モデルと比べても，*vpr* を除くどのプログラムでも，S =1〜4 での性能の中で最も高い性能とほぼ同等の性能を達成している．幾何平均では，理想モデルでプログラム毎に最善の S をとったときの性能を SPECint2000 で 5.0%，SPECfp2000 で 4.1%下回る

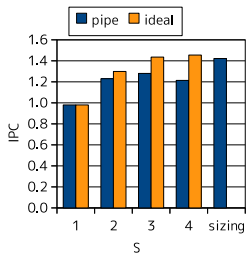
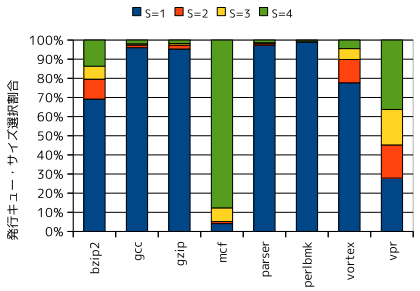


図 8 SPECint2000 と SPECfp2000 を合わせた平均 IPC

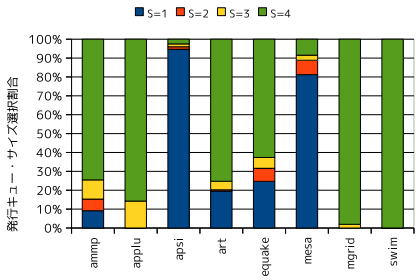
に過ぎない．このことは，動的サイジングがパイプライン化された発行キューのデメリットを避けつつタイムリーに発行キューを拡大し MLP を利用していることを示している．以上のように，提案の動的サイジング手法は，理想モデルと比べてもほとんどのプログラムで遜色ない性能を達成していることから，本論文で提案した方式の枠組みにおいて，より複雑なパラメータ・チューニング (例えば，発行キュー拡大と判断するまでに必要とする L2 キャッシュ・ミス回数のチューニング) は不要といえる．

メモリ・インテンシブなプログラム (表 1 参照) に注目すると，固定サイズ・モデルと理想モデルでは，多くの場合 S を増加させると著しく性能は向上し，$S = 4$ の時に最大性能を示している．また，2 つのモデルの間の性能差は小さい．これは，プログラムの実行時間が主記憶アクセス時間に支配されており，MLP を最も利用できる最大サイズの発行キューが有利であることを示している．しかし，*mcf, vpr, equake* にはその傾向がない．*mcf, vpr* に関しては，分岐予測ミス率が高いことが原因であると思われる (表 1 参照)．分岐予測ミス率が高いと，大きな発行キューを多くの命令で埋めることができなくなり，MLP の利用度が低下する．これに対して，*equake* が大きく性能向上しない点については，今のところ原因は不明である．動的サイジング・モデルは，前述したように *vpr* を除いて，固定サイズ・モデルおよび理想モデルにおける最高性能とほぼ等しい性能を達成している．*vpr* については，分岐予測ミスによる MLP 利用度の低下に加え，分岐予測ミス・ペナルティの影響を強く受け (理想モデルで，S を増加させると性能が著しく低下していることからわかる)，固定サイズ・モデルおよび理想モデルにおける最高性能に達していない．

計算インテンシブなプログラムに注目すると，理想モデルでは S が変化しても性能は変化が少い．このことは，発行キュー・サイズ 64 エントリ (ベース・プロセッサの発行キュー・サイズ) で ILP 利用がほぼ限界にあることを示している．一方，固定サイズ・モデルでは，S を増加させると多くの場合，著しく性能が低下する．これは，MLP 利用の機会が少なく，大きなパイプライン化発行キューが ILP 利用において不利であることを示している．これに対して動的サイジング・モデルは，固定サイズ・モデルと理想モデルにおける最高性能とほぼ等しい性能を達成している．



(a) SPECint2000



(b) SPECfp2000

図 9 選択された発行キュー・サイズの割合

5.6 動的サイジング・モデルにおいて選択された発行キュー・サイズの割合

動的サイジング・モデルにおいて，実行時に選択された発行キューのサイズ (S で表している) の分布を**図 9** に示す．当然ながら，メモリ・インテンシブなプログラムでは S が大きい状態にある割合が高く，逆に計算インテンシブなプログラムでは S が小さい状態にある割合が高くなっている．多くのプログラムで，$S = 1$ または $S = 4$ という極端な状態が多い．しかし興味深いプログラムとして，*bzip2* でのサイズ選択割合に注目されたい．図 6 で示したように，*bzip2* では，固定サイズ・モデルで $S = 1$ のときが最善である．しかし，図 9 に示すように，S =2〜4 の期間が全体の 30%以上を占めている．すなわち，*bzip2* は計算インテンシブではあるが，少数ながらキャッシュ・ミスを起こし，それに合わせてタイムリーに発行キューを拡大し MLP を利用していると言える．この結果，$S = 1$ の固定サイズ・モデルの性能を 4.3%上回ることができた．

同様に興味深い例として *art* がある．このプログラムは表 1 に示すように極めてメモリ・インテンシブでありながら，$S = 1$ の期間が 20%もある．つまり，全体としては非常にメモリ・インテンシブであるが，計算インテンシブな期間が存在するということである．この期間は発行キューを小さくし，ILP 利用に最適化することが得策である．このように振舞った結果，固定サイズ・モデルで最善の $S = 4$ の時より，2.0%とわずかではあるが性能が向上している．

最も例外的なプログラムとして *vpr* がある．このプログラムでは，かなり均等に S =1〜4 が選択されている．表 1 に示す通り，このプログラムはメモリ・インテンシブなものに分類できるが，L2 キャッシュ・ミス率はその中では最も低い．また，図 5 に示す通り，全 L2 キャッシュ・ミス数に対して間隔 256 サイクル以下で生じる L2 キャッシュ・ミスの割合は，メモリ・インテンシブなプログラムの平均で 63%であるのに対して，47%と低い．以上のことから，最適な発行キュー・サイズは時間とともに激しく変化し，しかも MLP 利用の機会があまり多くないという特徴を持っていることがわかる．このため，かなり均等に S =1〜4 が選択されていると思われる．このようなプログラムは発行キュー・サイズの最適な選択は難しく，結果として，図 5 に示す通り，固定サイズ・モデルの最善性能より唯一大きく (11.1%) 性能が低下している．

ただ 1 つのプログラムで問題があるもののその他のプログラムでは，メモリ・インテンシブ，計算インテンシブに関わらず，提案の動的サイジング手法により MLP と ILP を最適に利用しているといえる．

6. ま と め

MLP を利用する有効な手法として，大きな発行キューによる積極的なアウト・オブ・オーダ実行がある．しかし，大きな発行キューのクロック速度への悪影響を除くパイプライン化は ILP の利用を妨げてしまう．本論文では MLP を利用できる時は発行キューを拡大し MLP を利用し，そうでない時は発行キューを縮小し ILP の利用を行う動的な発行キューのサイジング手法を提案した．我々の手法は極めて単純でありながら，非常に有効に機能する．SPEC2000 ベンチマークを用いて評価した結果，提案の動的サイジング手法を用いれば，評価したほとんどのプログラムでサイズを固定した場合での最善の性能とほぼ同等かそれ以上の性能を達成できることを確認した．平均では，サイズを固定した場合での最善の性能より SPECint2000 で 0.7%，SPECfp2000 で 10.2%，両ベンチマーク・スイートで 11.1%高い性能を達成した．

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 22500045）による補助のもとで行われた．本研究は東京大学大規模集積システム設計教育研究センターを通しシノプシス株式会社の協力で行われたものである．

参 考 文 献

[1] D. Folegnani, et al., Energy-effective issue logic, In *ISCA-28*, pp. 230–239, Jun. 2001.
[2] D. Ponomarev, et al., Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources, In *MICRO-34*, pp. 90–101, Dec. 2001.
[3] J. Stark, et al., On pipelining dynamic instruction scheduling logic, In *MICRO-33*, pp. 57–66, Dec. 2000.
[4] 加藤伸幸ほか，命令発行キューの深いパイプライン化のための投機発行，In *SACSIS 2009*, pp. 319–326, 2009 年 5 月.
[5] M. D. Brown, et al., Select-free instruction scheduling logic, In *MICRO-34*, pp. 204–213, Dec. 2001.
[6] M. S. Hrishikesh, et al., The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays, In *ISCA-29*, pp. 14–24, May 2002.
[7] E. Brekelbaum, et al., Hierarchical scheduling windows, In *MICRO-35*, pp. 27–36, Nov. 2002.
[8] J. A. Farrell, et al., Issue logic for a 600-MHz out-of-order excution microprocessor, *JSSC*, Vol. 33, No. 5, pp. 707–712, May 1998.
[9] O. Mutlu, et al., Runahead execution: An effective alternative to large instruction windows, In *HPCA-9*, pp. 129–140, Feb. 2003.
[10] 甲良祐也ほか，命令発行キューの遅延時間評価，In *SACSIS 2010*, pp. 45–52, 2010 年 5 月.
[11] S. Palacharla, et al., Quantifying the complexity of superscalar processors, Technical Report CS-TR-1996-1328, University Wisconsin, Nov. 1996.
[12] 五島正裕, Out-of-order ILP プロセッサにおける命令スケジューリングの高速化の研究, 京都大学，博士論文, 2004 年 3 月.
[13] E. F. Torres, et al., Store buffer design in first-level multibaked data caches, In *ISCA-32*, pp. 469–480, Jun. 2005.
[14] H. W. Cain, et al., Memory ordering: A value-based approach, In *ISCA-31*, pp. 90–101, Jun. 2004.
[15] J. L. Cruz, et al., Multiple-banked register file architectures, In *ISCA-27*, pp. 316–325, May 2000.
[16] R. Shioya, et al., Register cache system not for latency reduction purpose, In *MICRO-43*, pp. 301–312, Dec. 2010.
[17] R. Balasubramonian, et al., Reducing the complexity of the register file in dynamic superscalar processors, In *MICRO-34*, pp. 237–248, Dec. 2001.
[18] N. Muralimanohar, et al., CACTI 6.0: A tool to model large caches, HPL-2009-85, HP Laboratories, Apr. 2009.