2008－ARC－179 (7)
2008／8／6

命令発行キューの深いパイプライン化

加藤伸幸† 安藤秀樹†

発行キューはパイプライン化すれば，クロック速度を低下させず拡大が可能になり，メモリ・レベル並列性を利用し性能を向上させることができる．しかし，単純にパイプライン化するだけでは，命令の発行が遅れIPCが低下するという欠点を持つ．本論文ではこの問題に対して，過去のスケジューリングのタイミング履歴を利用した投機発行を行うことで，命令の発行の遅れを減少させる手法を提案する．SPECfp2000ベンチマークを用いて評価を行った結果，256エントリの命令キューを8段にパイプライン化した場合，本手法を使わない場合に比べ，28.7%性能を改善することを確認した．

Deep Pipelining of Instruction Issue Queue

Nobuyuki Kato† and Hideki Ando†

A large issue queue improves processor performance by exploiting memory-level parallelism. The issue queue can be enlarged without adverse effect on the clock cycle time if it is pipelined. Simple pipelining, however, degrades IPC due to the delay of issue. This paper proposes a speculative issue scheme based on the history of past issue timing, which allows decrease of issue delay. Our evaluation results using SPECfp2000 benchmark programs show that our scheme improves the performance of a processor with an 8-stage pipelined 256-entry issue queue by 28.7%, compared to that with the simple pipelining.

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．この問題を解決する1手法として，アウト・オブ・オーダ実行がある．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，発行キュー，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，発行キューに焦点を当て，クロック・サイクル時間を悪化させることなく，サイズを大きくする方式について提案する．

動作時間が長くクロック速度に悪影響を与える論理において，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，一方で，最適な命令スケジューリングが行えず，IPCが低下し，必ずしも得策ではない．IPCが低下する具体的理由は，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，依存する命令を発行させなければ，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いことである．

本研究では，大きな発行キューをパイプライン化し，クロック速度への悪影響を取り除く一方で，IPCを大きく低下させない手法を提案する．本手法の第1のキーは，従来のように，依存解決をトリガとして発行タイミングを定めるのではなく，最適な発行タイミングを事前に予測し，その予測にしたがって依存が解決する前に発行動作のトリガをかけることである．具体的には，発行キューをS段にパイプライン化した場合，ある命令の発行動作の開始を，その命令が間接的に依存する命令であって，予測した最適なタイミングよりSサイクル前に発行される命令によって起動する．この命令を発行命令に対する**トリガ命令**と呼ぶ．トリガ命令による投機的な発行動作開始が成功すればIPCを低下させることはない．最適な発行タイミングの予測は，履歴ベースで行う．プロセッサを$1/S$の遅いクロック周波数で動作させ，発行キューをパイプラ

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

イン化させずに動作させる**学習モード**を設ける．このモードでは通常のプロセッサと同様，依存解決によって発行タイミングが定まり，これを最適発行タイミングとみなす．各命令の発行タイミングを記録し，それを通常のクロック周波数 (これを**通常モード**と呼ぶ) でも再現させるべく，各命令の発行動作を開始させるトリガ命令を定める．

本方式の第 2 のキーは，発行タイミング予測を誤り，早期に発行されてしまった場合の効果的な回復措置である．この状況が最も起りうる場合は，トリガ命令と発行命令の間の命令でデータ・キャッシュ・ミスが生じたときである．MLP を利用しようとするようなメモリ・インテンシブなプログラムではこの状況は頻発する．このような場合，ソース・オペランドが利用可能となる前に発行されるので，実行はできない．そのため，ソース・オペランドが利用可能となるまで待ち合わせる必要があるが，発行キューでそれを行うと，依存解決から S サイクル経過した後でなければ発行できず，パイプライン化の悪影響を被る．そこで，本手法では待ち合わせのために小さなバッファを用意する．このバッファを**再発行キュー**と呼ぶ．早期に発行されてしまった命令は再発行キューに移され，そこでソース・オペランドが利用可能になるのを待ち，発行される．再発行キューは，発行キューと異なり，1 サイクルで発行動作が完了するに十分なだけ小さなものとする．これにより，投機には失敗してもオペランドが利用可能となれば，その直後に発行されうる．したがって，早期発行の投機失敗が生じても大きな IPC 低下を引き起こすことはない．

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，投機発行方式を提案する．4 節で評価を行い，5 節で本論文をまとめる．

2. 関 連 研 究

非常に多くの in-flight 命令をサポートしたアウト・オブ・オーダ実行が MLP を利用して IPC を向上させることができることは，広く知られた事実であり，たとえば，文献 1) で述べられている．

Stark らは，2 つ前の依存命令のタグを発行キューに保持することにより発行キューを 2 段にパイプライン化する手法を提案した[2]．この手法を，$S(>2)$ 段パイプラインに拡張するには，トリガ命令を見つけるために，リネーム・ステージで依存関係を追跡し，そのパス上の命令のレイテンシを加える論理が必要で，非常に複雑化する．また，トリガとなる命令と発行命令の間の命令でキャッシュ・ミスが生じると，発行命令は非投機で発行，すなわち，S サイクルほど待ち合わせてからでなければ発行できず，メモリ・インテンシブなプログラムでは大きな不利を被る．

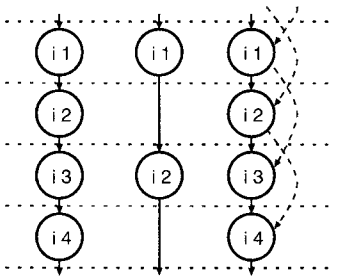

図 1 発行のタイミング図 1

Michaud らは，小さな発行キューとその前段にデータフローに基づきプレスケジューリングされた命令を保持する FIFO を置く構成を提案した[3]．プレスケジューリングにより，発行キューに命令が滞在するする時間を短くし，小さな発行キューを有効に利用できる．しかし，キャッシュ・ミスの発生によりプレスケジューリング通りに命令が発行されなければ大きく性能が低下するという欠点があり，MLP の利用には適さない．

Raasch らは，発行キューを小さなセグメントに分割し，パイプライン化する方式を提案した[4]．命令のセグメント移動は，発行までの遅延時間が，あらかじめ定められた遅延時間を下回ったときに行われ，最終段のセグメントに至ると命令は発行される．発行キュー内の命令は依存関係を識別され，「鎖」を構成する．キャッシュ・ミスが生じると，それにつながった鎖上にある命令はセグメント内でストールさせる．これにより，最終段のセグメントが発行できない命令で埋まることを防止することができる．しかし，セグメント間移動の制御に，予測待ち時間とセグメント移動を許可する閾値の大小比較などが必要であり，それがクリティカル・パスを長くしている．

3. 投 機 発 行

本章では，発行キューのパイプライン化に伴う IPC 低下を緩和するための投機発行の手法について提案する．

3.1 投機発行の概要

図 1 を用いて，投機発行の基本的なアイデアを説明する．命令 `i1`〜`i4` は順に依存関係にあり，それを実線の矢印で表している．実行レイテンシは全て 1 サイ

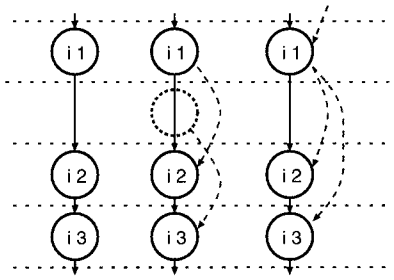

図 2 発行のタイミング図 2

クルとする．下方向が時間の増加方向であり，1 サイクルごとに点線で仕切られており，丸の位置は命令が発行されるタイミングを示す．

通常モードでの発行キューのパイプライン段数 S が 2 の場合を考える．学習モード (1/2 のクロック周波数) では，発行キューのパイプライン段数が 1 なので，命令 `i1`〜`i4` は (a) のように毎サイクル発行される．クロック周波数を上げ，発行キューを単に 2 段にパイプライン化すると，(b) のように毎サイクル発行できなくなり，IPC は低下する．そこで本手法では，学習モード時に (a) のように発行された場合，点線で表しているように，命令 `i1` が発行されるとその結果タグを命令 `i3` が受け取り，発行動作を投機的に開始する．同様に `i2` が出力する結果タグを `i4` が受け取り，発行動作を投機的に開始する．このように依存解決により発行動作を開始させるのではなく，S サイクル前に発行動作を開始させることにより，パイプライン化による命令発行の遅れをなくす．命令 `i1`，`i2` を，それぞれ，命令 `i3`，`i4` に対するトリガ命令と呼ぶ．このような投機発行のために，各ソース・オペランドには通常のタグの他，トリガ命令の結果タグである**トリガ・タグ**を，発行キューに保持する．

投機成功のためには，トリガ命令と投機発行命令との間の相対サイクル数差が学習モードと通常モードで変化しないことが望ましい．このために，トリガ命令は投機発行命令から発するクリティカル・パス上にある命令とする．つまり，これまで述べなかったが，図 1 の依存関係を表す矢印は，2 つのソース・オペランドの内，後にレディとなった関係を表す．

ある命令のトリガ命令が必ずしも，S サイクル前に存在するわけではない．**図 2** にそのような場合を示す．同図では，図 1 の場合と異なり命令 `i1` のみレイテンシが 2 サイクルとする．このとき学習モードでは (a) のように発行される．この場合，(b) に示すように，命令 `i3` の 2 サイクル前に発行された命令はなく，トリガ命令を設定できない．そこで (c) に示すように，それより以前で最も近い命令，この場合，命令 `i1` をトリガ命令とする．この場合，正しいタイミングで命令 `i3` が発行されるためには，`i1` の結果タグを受け取ってから 1 サイクル待たなければならない．このために，タグが一致した後レディ・ビットをセットするまでの待ち時間を記憶するシフト・レジスタを，発行キューの各エントリに用意する．n サイクルの待ち時間を要する場合，右から第 n ビット目をセットする．トリガ・タグが結果タグと一致したら，それ以後，シフト・レジスタを毎サイクル右にシフトさせ，シフト・レジスタから 1 が出力されたらレディ・ビットをセットする．このシフト・レジスタを特に，**遅延シフト・レジスタ**と呼ぶ．

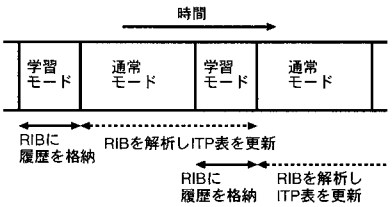

図 3 発行タイミング学習の過程

3.2 発行タイミングの学習

発行タイミングの学習過程は，2 段階からなる (**図 3** 参照)．第 1 段階は，発行タイミングの履歴を採取するもので，学習モードで行う．このとき **RIB** (retired instruction buffer) と呼ぶバッファを使う．RIB の第 i エントリは，学習モードにおける第 i サイクルに対応している．各エントリは，命令発行幅と同数のフィールドを持ち，各フィールドは，当該エントリに対応するサイクルに発行された命令の「年齢」，デスティネーション・レジスタ番号，後でレディになった方の論理ソース・レジスタ番号を保持する．ここで「年齢」とは，命令フェッチ順に付けられた順序番号である．命令がコミットされたら，その命令が発行されたサイクルに対応する RIB のエントリに上記情報が記録される．

第 2 段階は，RIB を探索し発行タイミング情報を後述する **ITP 表** (issue timing prediction table) と呼ぶ表に書き込むもので，第 1 段階の学習モードに続く通常モードと次の学習モードの間に行われる．

ITP 表について説明する．ITP 表は，命令の PC をインデクスとするセット連想表で，各エントリは複数のフィールドを持つ．各フィールドは，以下の 3 つの情報を保持する．

- dist: トリガ命令と投機発行命令との年齢差

- dly: トリガ命令と投機発行命令の発行サイクル差 $-S-1$
- cnf: 当該情報の信頼性

cnf はカウンタであり，同じ情報 (dist, dly) を学習すれば 1 増加させ，異なる情報を学習すれば 1 減少させる．エントリやフィールドは LRU で入れ換える．

RIB の解析は格納された 1 命令ごとに行う．ある命令 I が書き込まれている RIB のエントリから，順にさかのぼり，依存している命令をたどる．$S+1$ サイクル以上前に発行された命令を発見することができれば，その命令をトリガ命令とし，命令 I が対応している ITP 表のエントリを更新する．発見できなければ，投機はしないとして ITP 表は更新しない．ここで，トリガ命令を 3.1 節で説明したように，命令 I が発行された S サイクル以上前の命令ではなく，$S+1$ サイクル以上前の命令とする理由は次節で説明する．

3.3 投機発行の方法

投機発行を行うためには，まず，トリガ命令のタグを得る必要がある．このために，**STQ** (sequential tag queue) と呼ぶキューを用意する．このキューは，命令に割り当てられたデスティネーション・タグをイン・オーダで保持するものである．命令は PC をインデクスとして ITP 表を参照する．エントリが見つかったら，それが保持している (dist, dly) の組の中で最も cnf が大きい組を選択し，得る．当該命令の年齢から dist だけ前の年齢の命令がトリガ命令であるから，STQ の 末尾から dist 先のエントリのデスティネーション・タグを得，トリガ・タグとする．

ディスパッチ時には，発行キューに従来の情報の他，トリガ・タグを記憶する他，ITP 表から読み出した dly で遅延シフト・レジスタを初期化する．

命令は発行されると，従来と同じく，結果タグをウェイクアップ論理に放送する．発行キューで待ち合わせている命令の中で，トリガ・タグとそれが一致する命令があれば，その命令は dly サイクル後にレディとなり，セレクト論理に発行要求信号を出す．選択されれば発行される．

投機発行された命令は，物理ソース・レジスタに対応するビジー・ビットをアクセスすると共に，バイパス論理のレジスタ番号比較器を動作させる．これにより，ソース・レジスタが利用可能と分かれば，投機成功であり，実行可能である．この場合，結果タグを発行キューに放送し，パイプラインを前へ進む．利用可能でなければ，投機失敗であり，実行できない．この場合の動作は 3.4 節で説明する．ここで，ソース・レジスタが利用可能かどうかをチェックするために，1

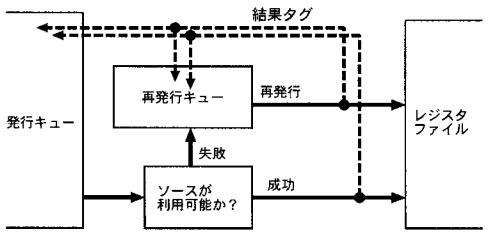

図 4 投機失敗時の再発行の仕組み

表 1 ロード命令のメモリ・アクセス率 (%)

ammp	applu	apsi	art
12.7	8.1	0.6	42.6
equake	mesa	mgrid	swim
5.1	0.1	3.8	28.0

サイクルを要し，その後に結果タグが発行キューに放送されることに注意されたい．この 1 サイクルが必要であるために，RIB の探索では，投機発行命令の S ではなく $S+1$ サイクル以上前の命令をトリガ命令とするのである．

3.4 投機発行失敗からの回復

1 節で述べたように，投機発行に失敗した命令は再発行キューに移される．**図 4** に構成を示す．再発行キューは十分に小さく，依存が解決すれば 1 サイクルで発行可能である．再発行キューに命令を移動するために 1 サイクルのペナルティは存在するが，発行キューからの再発行する場合に要する S サイクルの遅延は被らない．なお，再発行キューに空きがなければ，発行キューからの投機的な発行のみ中止し，再発行キューからのみ発行し，空きができるのを待つ．これは IPC を低下させるが，一般に，再発行キューに格納される命令は，滞在時間が短く短時間の内に発行されるため，この影響は小さいと思われる．

4. 評価結果

4.1 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・プログラムとしてメモリ・インテンシブなプログラムを多く含む SPECfp2000 から 8 本のプログラムを使用した．**表 1** にロード命令のメモリ・アクセス率を示す．*apsi*, *mesa* 以外はメモリ・インテンシブなことがわかる．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．シミュレーション時間が過大にならないように，命令のスキップと実行命令数の制限を行った．

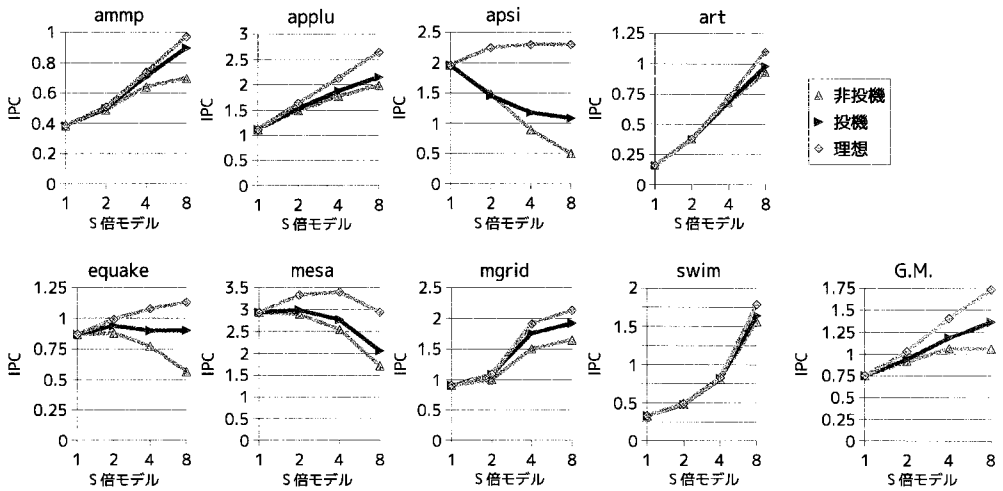

図 5 IPC

表 2 ベース・プロセッサの構成

Pipeline width	8-instruction wide for each of fetch, decode, issue, and commit
ROB	64 entries
LSQ	32 entries
Issue queue	32 entries
Function unit	8 iALU, 4 iMULT/DIV, 4 Ld/St, 6 fpALU, 4 fpMULT/DIV/SQRT
L1 I-cache	64KB, 2-way, 32B line
L1 D-cache	64KB, 2-way, 32B line, 4 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	15-bit history gshare, 256K-entry PHT, 10-cycle misprediction penalty
Physical register	total 128 (64 for each of int and fp)

表 3 提案手法に関するパラメータ

学習モード	1K サイクル
通常モード	1M サイクル
RIB	1K エントリ
ITP	64K エントリ, 8 ウェイ
	1 エントリあたり 8 フィールド
再発行キュー	32 エントリ

表 2 にベース・プロセッサの構成を示す．また，提案手法に関するパラメータを**表 3** に示す．発行キューがベースの設定に対して，S 倍 $(S = 1, 2, 4, 8)$ で，S 段にパイプライン化したものを評価した．これらを S **倍モデル**と呼ぶ．各 S 倍モデルでは，発行キューのサイズとバランスをとるため同時に ROB，LSQ，レジスタ・ファイルのサイズを S 倍に拡大している．これらも発行キューと同様，全て S 段にパイプライン化されていると仮定する．このため，分岐予測ミス・ペナルティが増加している．分岐処理のパスには，ROB，発行キュー，レジスタ・ファイルの 3 つの論理が含まれるので，S 倍モデルにおける分岐予測ミス・ペナルティは $3 \times (S-1) + 10$ である．

4.2 IPC

図 5 に IPC の測定結果を示す．3 本の折れ線グラフは，本手法を使う場合（投機），使わない場合（非投機），理想の場合（理想）の IPC である．理想は，サイズが拡大しても発行キューが 1 サイクルで動作すると仮定した場合である．ただし，分岐予測ミス・ペナルティはパイプライン化した場合と同じく増加させている．

グラフから次のことがわかる．まず，極度にメモリ・インテンシブな *art*，*swim* においては，非投機でも非常に大きな性能向上を達成しており，また，理想に近い IPC を達成できている．これらのベンチマークでは，第 1 に，メモリ・アクセスがデータフローの多くのクリティカル・パスを形成しており，大きな発行キューの恩恵を大きく受けられる．また，非投機ではスケジューリングが悪化するが，メモリ・アクセス時間は非常に長く，クリティカル・パス長の増加率が小さい．これらのプログラムでは，非投機でも理想に近いので，投機の必要性は小さい．

次に，適度にメモリ・インテンシブな *ammp*，*applu*，*equake*，*mgrid* では，S が増加するにつれ，理想では

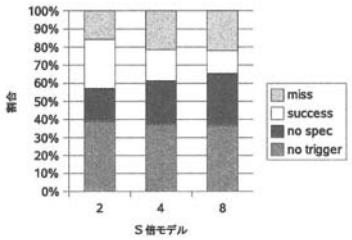

図 6 命令発行の分類

大きく IPC が向上しているが，非投機では，IPC がそれより大きく劣化していることがわかる．この劣化は，投機により大きく回復できている．このことより，本方式はスケジューリングを大きく悪化させることなく，MLP を利用することができていることがわかる．

最後に，メモリ・インテンシブでない *apsi*，*mesa* では，MLP がほとんど存在しないため，S を増加させても，理想においてさえ，あまり IPC は増加しないか，分岐予測ミス・ペナルティ増加の影響で低下してしまうことがわかる．非投機では，スケジューリングの悪化により，大きく IPC を悪化させてしまう．投機によりある程度回復するが，十分ではない．

平均では，本手法による投機により，非投機の場合に比べて，$S = 2, 4, 8$ の場合，それぞれ，2.8%，11.3%，28.7%IPC を改善することができた．この結果，ベースに対して，それぞれ，25.1%，57.4%，81.3%IPC を向上できることを確認した．

4.3 考　　察

発行キューより発行された命令を以下の項目で分類した結果を**図 6** に示す．

- *no trigger*: ITP 表にトリガ命令が見つからなかった場合．非投機で発行される．
- *no spec*: トリガ命令は見つかったが，投機発行はされなかった場合．トリガ命令が実行されなかったか，実行されたがそれより早く依存命令が実行された場合．非投機で発行される．
- *success*: 投機発行され，非投機の場合に比べて 1 サイクル以上早く発行された場合．
- *miss*: 投機発行されたが，ソースが利用可能でなく，投機に失敗した場合．

図からわかるように，S が増加するほど投機された割合 ($success + miss$) は減っている．投機は，失敗 ($miss$) であっても再発行キューから発行され利益を得るので，失敗を含めてその割合が大きいことが性能上望ましい．**図 7** に，8 倍モデルにおける投機率と IPC 回復率の相関を示す．投機率とは，*success* と *miss* の割合の合計である．IPC 回復率は，理想を基準に非投機により劣化した IPC を投機によりどれほど回復できたかの割合である．下記の式で表される：

$$IPC\ 回復率 = \frac{投機の\ IPC - 非投機の\ IPC}{理想の\ IPC - 非投機の\ IPC}$$

図から投機率と IPC 回復率に正の相関が確認できる．非常に深いパイプラインにおいても，高い投機率を達成することが今後の課題である．

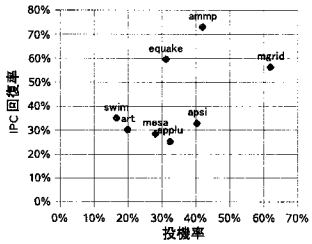

図 7 投機率と IPC 回復率の関係

5. ま と め

本論文では，パイプライン化した命令キューにおいて，発行の遅れを解消する投機発行手法を提案した．SPECfp2000 ベンチマークを用いて評価を行った結果，256 エントリの命令キューを 8 段にパイプライン化した場合，本手法を使わない場合に比べ，28.7%性能を改善することを確認した．

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 19500041）による補助のもとで行われた．

参 考 文 献

[1] A. Cristal, O. J. Santana, F. Cazorla, M. Galluzzi, T. Ramírez, M. Pericás, and M. Valero, "Kilo-Instruction Processors: Overcoming the Memory Wall," *IEEE MICRO*, vol.25, no.3, pp.48–57, May/June 2005.

[2] J. Stark, M. D. Brown, and Y. N. Patt, "On Pipelining Dynamic Instruction Scheduling Logic," *MICRO-33*, pp.57–66, December 2000.

[3] P. Michaud and A. Seznec, "Data-Flow Prescheduling for Large Instruction Windows in Out-of-Order Processors," *HPCA-7*, pp.27–36, 2001.

[4] S. E. Raasch, N. L. Binkert, and S. K. Reinhardt, "A Scalable Instruction Queue Design using Dependence Chains," *ISCA-29*, pp.318–329, May 2002.