計算機アーキテクチャ 133－4
（1999．5．21）

データ投機実行のための命令再発行機構と命令スケジューリング機構の分割

佐藤寿倫†

本稿で我々は，命令スケジューリングのための機構と命令再発行のための機構を分けることを提案する．従来の命令スケジューリング機構に命令再発行の機能を持たせると，プロセッサの性能に悪影響を与える可能性がある．投機実行されている命令とデータ依存の関係にある命令を命令ウインドウ中に保持し続けなければならないために，命令ウインドウの実効容量が減少し，その結果命令スケジューリングの自由度が低下してしまう．命令スケジューリング機構と命令再発行機構を分割すれば，この問題は解決できる．小さな命令ウインドウが命令スケジューリングを担当し，大きな命令バッファが命令再発行を担当する．シミュレーションの結果，この分割の有効性を確認できている．

Decoupling Instruction Reissue snd Scheduling Mechanisms

TOSHINORI SATO†

In this paper, we propose to decouple the recovery mechanism for data speculation from dynamic instruction scheduling structure. Instruction reissue mechanism for data speculation has a serious impact on processor performance. The effective capacity of instruction window is reduced since instructions dependent upon a speculated instruction must remain in instruction window until it is committed. The decoupling of the recovery and scheduling mechanisms solves the problem. A small instruction window schedules instructions and its entry is released immediately when an instruction is dispatched. A large instruction buffer is active only when a misspeculation occurs and is used to reissue instructions dependent upon the misspeculated instruction. Using a cycle-by-cycle simulator, we evaluated the proposal and found that the decoupling is useful.

1. まえがき

近年，より大きな命令レベル並列度 (instruction level parallelism:ILP) を引き出すために，データ依存の投機実行に関わる研究が盛んである．データ値予測器を用いて命令の実行結果を予測し，その命令とデータ依存にある命令を投機的に実行する．その結果，より大きなILPを引き出すことが可能になる．もし投機に失敗すると，プロセッサの状態を投機開始前の状態に回復させなければならない．最も容易に実現可能な回復方法は，現在のプロセッサに備えられている分岐予測に用いられる回復機構を流用することである．しかし，このような命令破棄による回復機構はデータ投機実行には適していない．なぜなら，データ依存にない命令の演算結果も捨てられてしまい，有効な演算が無駄になってしまうからである．また，破棄された命令を再びフェッチしなければならないことも，大きなペナルティの要因となっている．この問題を解決できる機構の一つが命令再発行に基づく機構であり，これまで様々な検討がされてきた[7),8),10)]．しかし，命令再発行機構にはプロセッサの性能に悪影響を与える可能性がある．投機実行されている命令とデータ依存の関係にある命令を命令ウインドウ中に保持し続けなければならないために，命令ウインドウの実効容量が減少し，その結果命令スケジューリングの自由度が低下してしまう．本稿で我々は，命令スケジューリングのための機構と命令再発行のための機構を分けることを提案する．命令スケジューリング機構と命令再発行機構を分割すれば，上記の問題は解決できる．小さな命令ウインドウが命令スケジューリングを担当し，大きな命令バッファが命令再発行を担当する．

本稿は以下の構成から成る．2節で関連研究をまとめる．3節では命令スケジューリングと命令再発行の機能を分割したデカップル命令ウインドウを提案する．さらに4節で提案手法の評価環境について述べ，5節でシミュレーション結果を紹介する．最後に6節でまとめとする．

2. 関連する研究

データ値予測に基づくデータ投機実行[4),5)]は，予測されたデータ値を用いて投機的に命令実行を開始することで，より大きなILPを引き出そうとする技術である．しかし，データ投機実行を実現するためには，データ値予測器だけでなく，予測に失敗した時のためのプロセッサ状態の回復機構も必要となる．

この回復方法には以下の二つの機構が提案されている．一つは命令破棄に基づく機構であり，残りの一つは命令再発

† 株式会社東芝 セミコンダクター社 マイクロエレクトロニクス技術研究所
Toshiba Microelectronics Engineering Laboratory
toshinori.sato@toshiba.co.jp

行に基づく機構である．命令破棄を採用すると，データ値予測に失敗した命令に後続する命令は全て破棄されてしまう．この命令破棄の機構は，分岐予測のための機構としてすでに現在のプロセッサに備わっているため，実装は非常に容易である．しかしこれまでの研究から，命令破棄に基づく機構を用いると，データ投機実行の効果が得られないばかりでなく，プロセッサの性能に悪影響を与える場合があることがわかっている[8),10)]．一方，命令再発行に基づく機構は，投機に失敗した命令とデータ依存にある命令だけを選択して再実行する．これらの命令は命令ウインドウ中で選択的に検出され再発行される．この命令再発行を実現するためには，投機された命令と依存関係にある命令が，ディスパッチ後もスケジューリングウインドウ内に留まっておく必要がある．値を予測した命令が実際の演算を終了すると，予測値と実際の値とは比較され予測の成功が判定される．もし二つの値が一致すれば予測は成功であり，依存関係にあった命令はスケジューリングウインドウを解放できる．もし一致しなければ，誤った値に基づく演算結果は無効化され，その命令は再発行される．この命令再発行のコンセプトはLipastiら[5)]によって提案されたが，彼らは実装方法には触れていなかった．

Tysonら[10)]は彼らのロードデータ値予測機構の研究において，Lipastiらの命令再発行を評価している．彼らの報告によると，命令破棄に基づく機構を用いるとプロセッサ性能に悪影響が現れるプログラムに対しても，命令再発行機構はデータ投機実行を有効に活用できる．しかし，彼らもまた命令再発行の実装方法を検討してはいない．Rotenbergら[7)]は，彼らの提案しているトレースプロセッサにおける命令再発行機構の効果を検討し，データ投機実行に有効であることを報告している．しかし彼らは，現在主流のスーパースカラプロセッサでの効果を評価してはいない．

我々はすでに，命令再発行機構の現実的な実装方法を検討している[8)]．レジスタ更新ユニット (register update unit:RUU)[9)]を拡張して実現している．RUUはコミットされるまで全ての命令を保持し続けるので，命令再発行に非常に適している．我々の提案している命令再発行機構では，再発行されるべき命令を徐々に発見していくため，一度に全てを発見する場合に必要になる比較器の数を大幅に減少できている．

命令再発行機構はデータ値予測失敗のペナルティを大幅に軽減できるが，プロセッサの性能に悪影響を与える可能性がある[3)]．すでに述べたように，命令再発行を実現するためには，ディスパッチ後も全ての命令が命令ウインドウ中に留まっていなければならない．このためスケジューリングウインドウの実効容量が低下し，命令スケジューリングの自由度も低下する恐れがある．これを補うためには命令ウインドウの容量を増やせば良いが，命令ウインドウの容量はプロセッサの実行速度と密接に関連している[6)]ために，容易には増やすことができない．

ごく最近，Akkaryら[1)]が命令ウインドウの階層化を提案している．2階層の命令ウインドウを用意する．一つは容量が小さく，命令はディスパッチされると直ちに割り当てられていたエントリを解放する．データ投機に失敗した場合には，もう一つの大きい方の命令ウインドウがバックアップとして動作する．このバックアップの操作は，ちょうどキャッシュのリフィル操作と同じである．再発行されるべき命令が，大きなウインドウから小さなウインドウに投入される．投機失敗の頻度が小さければ，このリフィルによるプロセッサ性能への影響は無視できる．彼らは，彼らの提案するマルチスレッドプロセッサでおいてのみ，この2階層命令ウインドウを評価している．

3. デカップル命令ウインドウ

本節でデカップル命令ウインドウを提案する．すでに述べている通り，不容易に命令再発行機構を実装すると，以下の問題をひき起こす可能性がある．全ての命令はコミットされるまで命令ウインドウ中に保持されなければならないため，実効的な容量が低下してしまい命令スケジューリングの自由度も低下してしまう．プロセッサの性能を維持するにはウインドウの容量を増やせば良いが，その容量はプロセッサのサイクルタイムと密接に関連している[6)]ために，容易には増やすことができない．

この問題を解決するために，我々は命令スケジューリングのための機構と命令再発行のための機構を分割することを提案する．我々はこの分割されたウインドウをデカップル命令ウインドウと呼んでいる．図1に，デカップル命令ウインドウを用いるプロセッサの例を示す．デカップル命令ウインドウは，命令スケジューリングのための小さな命令ウインドウと，命令再発行のための大きな命令バッファとから構成されている．命令がフェッチされデコードされると，命令はこれら両方のウインドウに発行される．命令がウインドウから機能ユニットにディスパッチされると，命令は小さなスケジューリングウインドウ中のエントリを直ちに解放する．しかし，大きな命令バッファ中のエントリには待機し続ける．命令がコミットされると，ようやく大きな命令バッファ中のエントリが解放される．命令ウインドウあるいは命令バッファが一杯になった時には，ウインドウへの命令発行は停止する．

小さな命令ウインドウが命令スケジューリングを担当する．つまり，ほとんどの場合はスケジューリングウインドウから命令がディスパッチされる．ディスパッチされると命令はスケジューリングウインドウのエントリを解放するので，上で説明した命令ウインドウの実効的な容量低下の問題は解決されている．さらに，容量が小さいためにプロセッサのサイクルタイムに悪影響を与える心配もない．しかし，データ投機実行に失敗した場合，このスケジューリングウインドウ内部では命令を再発行できない．そのためのバックアップとして，大きな命令バッファが用意されている．命令再発行は命令バッファが担当する．全ての命令はコミットされるまで命令バッファに待機している．データ

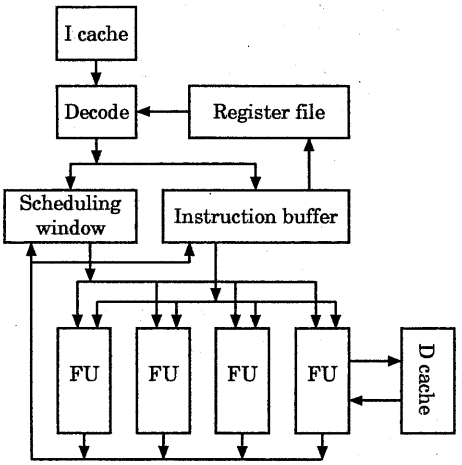

図1 プロセッサ図
Fig. 1 Processor diagram

投機実行に失敗すると，再発行すべき命令は命令バッファから獲得されディスパッチされる．大きなデータ投機実行の効果を得るために，命令バッファを非常に大きくする必要がある．そのような大きな命令バッファはプロセッサのサイクルタイムに影響するため，この悪影響を回避するために，命令バッファをパイプライン化する．命令バッファは投機失敗時だけに必要なので，このパイプライン化はプロセッサ性能に甚大な影響を与えないと予想している．

二つの命令ウインドウから得られる二つの同じ命令から一方を選択する方法は容易である．もしある命令が投機に失敗した時には，再発行すべき命令は命令バッファからしか得られない．したがって選択する必要はない．データ依存が存在してもまだディスパッチされていない命令は両方のウインドウから供給される．しかし，命令バッファはパイプライン化されているため，スケジューリングウインドウよりも同じ命令の供給が遅れる．したがって，機能ユニットにディスパッチされるのはスケジューリングウインドウから供給される命令であり，命令バッファから得られた同じ命令のディスパッチは抑制できる．

上記のように，デカップル命令ウインドウは命令再発行機構の実装により生ずる可能性のある問題を，プロセッサ性能を維持したまま解決できると思われる．

4. 評価方法

本節で，デカップル命令ウインドウを評価するための，シミュレーション環境とベンチマークプログラムを説明する．

4.1 評価モデル

シミュレータは実行ベースであり，投機失敗時に実行される命令を正確にシミュレーションできている．我々はSimpleScalar ツールセット[2] を用いて，このシミュレータを構築した．SimpleScalar/PISA の命令セット (instruction set architecture:ISA) は MIPS ISA に準拠している．

シミュレータは現実的なアウトオブオーダー実行を行なう 8 並列のスーパースカラプロセッサをモデル化している．動的スケジューリングは RUU[9] に基づいて行なわれる．RUU のエントリは 128 である．我々は二種類の RUU を評価している．一つはレイテンシが 1 サイクルである通常の RUU で，もう一つはパイプライン化されレイテンシが 2 サイクルである RUU である．各機能ユニットはあらゆる命令を 1 サイクルで実行できる．ただし，乗算と除算のレイテンシはそれぞれ 4 サイクルと 12 サイクルである．データの供給には 4 ポート，ノンブロッキング，容量 128KB，ラインサイズ 32B，2 ウエイセットアソシアティブの L1 データキャッシュを用意する．L1 データキャッシュのアクセスには，データアドレス生成後 1 サイクルを要する．キャッシュミス時のレイテンシは 6 サイクルである．L1 データキャッシュは，容量 8MB，ラインサイズ 64B，ダイレクトマップの L2 キャッシュがバックアップしている．L2 キャッシュのキャッシュミス時のレイテンシは，最初のワードが得られるまでに 18 サイクル，それに続くワードにはそれぞれ 2 サイクルである．全てのメモリ参照命令は，先行するストア命令が完了しなければ実行できない．命令供給には，容量 128KB，ラインサイズ 32B，2 ウエイセットアソシアティブの L1 命令キャシュを用意する．L1 命令キャッシュは L1 データキャッシュと L2 キャッシュを共有している．

分岐予測には，エントリ数 1K，4 ウエイセットアソシアティブの分岐先バッファ (branch target buffer:BTB)，エントリ数 4K，gshare タイプの 2 レベル適応型分岐予測器，エントリ数 8 のリターンアドレススタックを用いた．分岐予測器の更新は，命令がコミットされた時に行なうこととした．

本研究で用いるデータ値予測器は，エントリ数 4096，ダイレクトマップのストライド型予測器[4] である．図 2 に予測器を示す．命令アドレスでインデックスされ，各エントリには，タグアドレスフィールド (tag)，前回のデータ値フィールド (prev_value)，ストライドフィールド (stride)，そして信頼性フィールド (conf) がある．タグフィールドは登録されている命令を他の命令から区別するために用いられる．前回のデータ値フィールドには，前回同じ命令が実行された時に得られたデータが保持されている．ストライドは同じ命令が過去 2 回に生成した値の差である．予測データ値は，前回のデータ値とストライドとの和で生成される．信頼性フィールドは 2 ビット飽和型カウンタで構成され，予測された値を用いて投機実行すべきかどうかを決定する．予測が成功した時にはカウンタを +1 増加し，失敗した時には −1 減少させる．もしカウンタの値が 2 よりも小さい時には，投機実行は行なわない．今後本稿では，このデータ値予測器を用いるプロセッサモデルを予測モデルと呼ぶことにする．

デカップル命令ウインドウは，エントリ数 64 の集中型リ

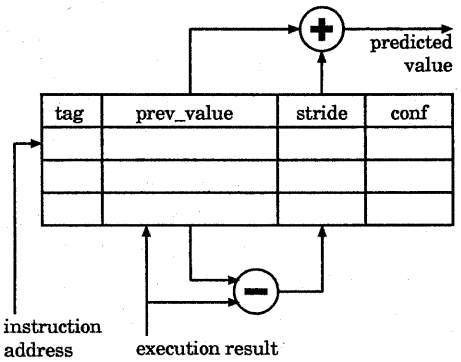

図2 ストライド型データ値予測器
Fig. 2 Stride value predictor

ザベーションステーションと命令再発行機能を持つエントリ数128のRUUから構成される．このRUUはパイプライン化されており，レイテンシは2サイクルである．図3にこのRUUを示す．各フィールドの詳細は8)を参照されたい．図4にはスケジューリングウインドウを示す．

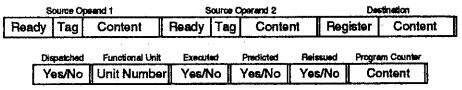

図3 拡張レジスタ更新ユニット
Fig. 3 Extended register update unit

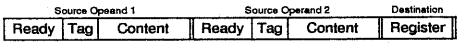

図4 スケジューリングウインドウ
Fig. 4 Instruction scheduling window

4.2 ベンチマークプログラム

ベンチマークプログラムにはSPECint95ベンチマークを用いた．SPEC協会が提供しているtestファイルを入力ファイルとして用いている．表1にベンチマークプログラムと入力ファイルを示す．各プログラムはGNU GCC (version 2.6.3)を用いて最適化オプション-O3でコンパイルした．各プログラムが終了するまで，あるいは最初の1億命令をシミュレーションした．命令はコミットされたものだけを数えた．

5. シミュレーション結果

本節でシミュレーション結果を紹介する．まず，データ値予測器の性能を評価する．つづいて，命令ウインドウがパイプライン化された時のプロセッサ性能への影響を調べる．最後に，デカップル命令ウインドウの効果を紹介する．

5.1 データ値予測の効果

図5に本研究で用いたデータ値予測器の予測可能性を示す．各グラフは3つの部分に分かれている．下の部分(黒)はデータ値が正しく予測された命令の割合を示している．中央の部分(白)はデータ値の予測に失敗した割合である．最後に上の部分(灰)は予測をしなかった割合である．図より平均して36.8%の命令が正しくデータ値を予測できていることがわかる．

表1 ベンチマークプログラム
Table 1 Benchmark program and input file

program	input file
099.go	null.in
124.m88ksim	ctl.in
126.gcc	cccp.i
129.compress	test.in
130.li	test.lsp
132.ijpeg	specmun.ppm
134.perl	primes.in
147.vortex	vortex.in

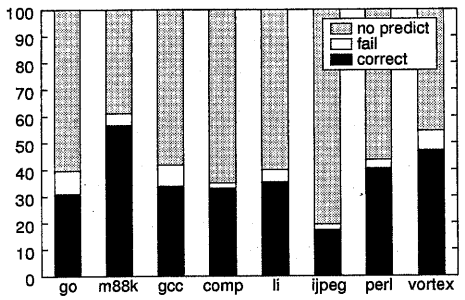

図5 (%) データ値予測可能性
Fig. 5 (%)Characteristics of value prediction

図6は，このデータ値予測器を用いた時のプロセッサ性能の向上率である．性能にはコミットされたサイクルあたりの命令数(committed instruction per cycle:IPC)を指標とした．各プログラムの予測モデルのIPCを基本モデルのIPCで正規化している．プロセッサ性能は平均して5.38%向上している．これはやや小さな性能向上であるが，データ予測器の性能向上が本稿の目的ではないので，このまま進めることにする．

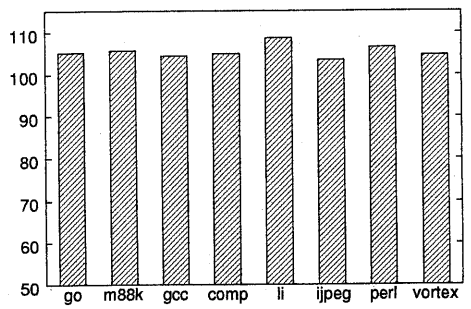

図6 (%) プロセッサ性能向上率
Fig. 6 (%)Performance improvement using value prediction

表2 命令ウインドウ使用率
Table 2 Instruction window utilization

program	w/o data prediction				with data prediction			
	total		waiting		total		waiting	
	avg	max	avg	max	avg	max	avg	max
099.go	21.6	128	11.0	126	21.5	128	9.6	126
124.m88ksim	18.3	128	8.3	121	16.2	128	5.6	127
126.gcc	21.7	128	11.9	127	21.6	128	11.1	127
129.compress	87.6	128	68.4	121	88.5	128	72.0	121
130.li	17.9	128	8.2	121	17.7	128	6.8	121
132.ijpeg	71.8	128	37.5	121	71.7	128	36.5	121
134.perl	23.8	128	10.9	121	22.0	128	9.1	121
147.vortex	27.8	128	15.0	121	24.8	128	12.8	121

表2は命令ウインドウの使用率である．コラム2～5は基本モデルの結果であり，コラム6～9が予測モデルの結果である．各4コラムのうちで，最初の2コラムは命令ウインドウに滞在している命令数を表している．これにはすでにディスパッチされた命令が含まれている．本研究で用いている命令ウインドウはRUUなのでディスパッチされた命令もRUUに留まっていることに注意されたい．残りの2コラムはディスパッチされるのを待っている命令数を表してる．これらが動的スケジューリングの対象となる命令である．さらにそれら2コラムのうちで，左は平均の命令数，右は最大の命令数である．データ値予測器を用いる場合，ディスパッチ待ちの命令数は`129.compress`の場合を除いて低下していることがわかる．しかし，RUU内に滞在している全命令数には大きな変化は見られない．このことから，ディスパッチ後の命令を保持し続けるRUUのようなタイプの命令ウインドウでさえ，データ投機実行を行なうと実効的な容量が低下していることがわかる．したがって，命令スケジューリングの自由度を保持するために，命令ウインドウの容量を増加させることが望ましい．また，以下のこともわかる．RUUの容量と比較して，平均の使用率は非常に低い．ほとんどのプログラムで，平均使用率は30%以下である．

5.2 パイプライン命令ウインドウの影響

図7は，命令ウインドウのパイプライン化がプロセッサ性能に与える影響を示している．命令の発火と選択のためのロジックがパイプライン化されている．各2本のグラフのうち，左は予測モデルの性能向上率であり，右がRUUがパイプライン化された時の性能向上率(低下率)である．このモデルもデータ値予測を行なっていることに注意されたい．容易にわかるように，RUUがパイプライン化されると，プロセッサ性能の低下が甚だしい．データ投機実行によって獲得された性能向上は失われ，さらに酷いことに，プロセッサ性能は基本モデルよりも低下している．例えば`132.ijpeg`の場合では，基本モデルと比べると，RUUがパイプライン化されると性能が約25%低下している．この性能低下を埋めるためには，プロセッサの動作速度を25%向上する必要があるが，その実現は非常に困難である．一方，命令ウインドウの容量を増加させることも解決策の一つと考えられるが，この方法はプロセッサのサイクルタイムを悪化させる可能性がある．

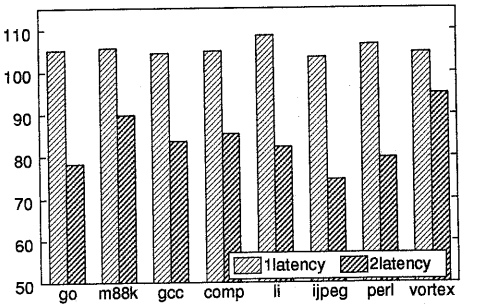

図7 (%) パイプライン化RUUの影響
Fig. 7 (%)Performance degradation by pipelined wakeup and compare logic

5.3 デカップル命令ウインドウの効果

本節で命令ウインドウ分割の効果を評価する．すでに5.1節で見たようにRUUの平均使用率は30%以下であるので，デカップルウインドウのスケジュールウインドウの容量を命令バッファの半分(64エントリ)と決める．

図8はデカップル命令ウインドウを採用した時のプロセッサ性能向上率である．各3本のグラフのうち，左はレイテンシが1サイクルのRUUを用いたモデルの性能向上率，中央はパイプライン化RUUを用いたモデルの性能向上率，右がデカップル命令ウインドウを用いたモデルの性能向上率である．スケジューリング機構と再発行機構を分割することで，命令ウインドウをパイプライン化したことによる性能低下が補われていることがわかる．全てのプログラムで，プロセッサの性能は基本モデルよりも向上している．1サイクルレイテンシRUUを用いた予測モデル(図の左のグラフ)と比較すると性能向上率はやや小さいが，デカップル命令ウインドウを用いるプロセッサは，1サイクルレイテンシRUUを用いるプロセッサよりも動作速度を向上できる可能性がある．したがって，性能向上率が勝る可能性があると言える．

表3は，デカップル命令ウインドウの使用率である．コラム2～3がスケジュールウインドウの使用率であり，コラム4～5が命令バッファの使用率である．2コラムからなるグループのレイアウトと内容は，2と同じである．デ

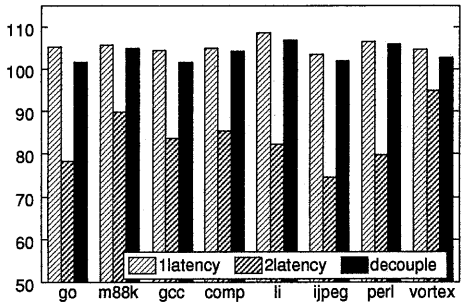

図8 (%) デカップル命令ウインドウの効果

Fig. 8 (%)Performance contribution of decoupled instruction window

カップル命令ウインドウモデルでのスケジューリングウインドウの容量は基本モデルなどのRUUの容量と比較して小さいので，その使用率が小さいのは当然の結果である．しかし，099.goと130.1iにおいて使用率が向上していることは興味深い．

表3 デカップル命令ウインドウの使用率

Table 3 Instruction buffer/window utilization

program	buffer		window	
	avg	max	avg	max
099.go	23.9	128	10.0	64
124.m88ksim	17.4	128	5.6	64
126.gcc	21.3	128	9.4	64
129.compress	57.8	128	40.7	64
130.li	18.7	128	7.0	64
132.ijpeg	68.1	128	32.4	64
134.perl	22.8	128	9.1	64
147.vortex	24.3	128	10.8	64

6. むすび

本稿で我々は，命令スケジューリング機構と命令再発行機構を分割することを提案した．その分割された命令ウインドウを，我々はデカップル命令ウインドウと呼んでいる．デカップル命令ウインドウは，動的命令スケジューリングを行なう小さなスケジューリングウインドウと，データ投機失敗時に命令再発行を行なう大きな命令バッファとから構成される．大容量の命令バッファは投機失敗時に限り使用されるので，プロセッサのサイクルタイムを維持するためにパイプライン化されたとしても，プロセッサの性能に与える悪影響は小さい．サイクルベースのシミュレータを用いてデカップル命令ウインドウを評価した．命令ウインドウをパイプライン化するとプロセッサ性能は著しく低下するが，デカップル命令ウインドウを用いることで性能低下を補うことが可能である．さらにデカップル命令ウインドウは，同じ容量の1サイクルレイテンシの命令ウインドウよりも高速に動作できるので，プロセッサの動作速度を向上でき，プロセッサの性能を向上できる可能性がある．

現在，命令ウインドウの容量を様々に振り，命令ウインドウをパイプライン化することによるプロセッサ性能への影響と，性能低下を補償するために命令ウインドウをデカップリングする効果とを，より詳細に調査中である．

参考文献

1) Akkary,H., Driscoll,M.A.: A dynamic multithreading processor, *31st International Symposium on Microarchitecture* (1998).
2) Burger,D., Austin,T.M.: The SimpleScalar tool set, version 2.0, *ACM SIGARCH Computer Architecture News*, 25(3) (1997).
3) Chrysos,G.Z., Emer,J.S.: Memory dependence prediction using store sets, *25th International Symposium on Computer Architecture* (1998).
4) Gabbay,F.: Speculative execution based on value prediction, *Technical Report #1080*, Department of Electrical Engineering, Technion (1996).
5) Lipasti,M.H., Wilkerson,C.B., Shen,J.P.: Value locality and load value prediction, *International Conference on Architectural Support for Programming Languages and Operation Systems VII* (1996).
6) Palacharla,S., Jouppi,N.P., Smith,J.E.: Complexity-effective superscalar processors, *24th International Symposium on Computer Architecture* (1997).
7) Rotenberg,E., Jacobson,Q., Sazeidas,Y., Smith,J.: Trace processors, *30th International Symposium on Microarchitecture* (1997).
8) 佐藤寿倫: 命令再発行機構によるデータアドレス予測に基づく投機実行の効果改善, 情報処理学会論文誌, 第40巻, 第5号 (1999).
9) Sohi,G.S.: Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers, *IEEE Transactions on Computers*, 39(3) (1990).
10) Tyson,G., Austin,T.M.: Improving the accuracy and performance of memory communication through renaming, *30th International Symposium on Microarchitecture* (1997).