2007－ARC－174 (29)
2007／8／2

物理レジスタ2段階解放による命令先行実行方式の低消費電力化

兵藤一永† 安藤秀樹†

近年のプロセッサはクロック速度とメモリ・アクセス速度の乖離が非常に大きく，性能に深刻な低下をもたらしている．データ・プリフェッチは，これを緩和する有効な手法である．我々はこれまで物理レジスタの2段階解放による命令の先行実行によるプリフェッチ手法を提案し，その有効性を示した．しかし，消費電力には注意が払われておらず，性能向上をもたらさない命令も先行実行され，電力効率に問題があった．そこで本論文では，この無駄な消費電力を削除する方式を提案する．まず，頻繁にL2キャッシュ・ミスするロードを動的に見つけ，これが直接もしくは間接的に依存している命令を抽出する．そして，これらのみを先行実行する．SPECfp2000ベンチマークを用いて評価した結果，平均10%の性能向上率の低下で72%の実行先行命令数削減可能であることを確認した．物理レジスタ2段階解放を組み込まないモデルに対し，平均11%の実行命令数の増加で80%の性能向上を達成している．

A Low-Power Design of Instruction Pre-Execution Mechanism with Two-Step Physical Register Deallocation

KAZUNAGA HYODO† and HIDEKI ANDO†

Recently the different between the clock speed of a processor and the memory access speed is large, and this adversely affects the performance seriously. Data prefetching is an effective scheme that alleviates this degradation. We previously proposed a prefetch scheme through instruction pre-execution with two-step physical register deallocation, and demonstrated the effectiveness. However, the scheme does not consider power consumption, and actually the scheme is inefficient in power consumption because instructions that do not contribute to the performance improvement are pre-executed. This paper proposes a scheme that reduces the wasted power. Our scheme dynamically finds a load that often occurs L2 cache misses, and marks the instructions that depend on the load directly or indirectly. Only marked instructions are then pre-executed. Our evaluation results using SPECfp2000 benchmark show that our scheme reduces the pre-executed instruction count by 72% with 10% performance degradation, compared with the original scheme. As a result, the speedup of 80% over a processor without pre-execution is achieved with the dynamic instruction count increase of 11%.

1. はじめに

ロードのレイテンシ (実行サイクル数) は，LSI技術の進歩と共に長くなっている．これは，メモリのアクセス速度の改善年率が，プロセッサのクロック速度の改善年率より悪いためである．このプロセッサ-メモリ間のギャップは，メモリ・ウォールとも呼ばれる．メモリ・ウォールによるレイテンシ増加を低減する最も一般的な方法は，ギャップを多くの階層で埋め，できるだけ高速な階層レベルでロード要求を満足させることである．しかしこの方法は，実現に高いコストを要するだけでなく，改善には限界があり十分でない．

この問題を解決する方法の1つに，データのプリフェッチがある．プリフェッチを実現するハードウェア手法として，これまでいくつかの方法が提案されている．これらの中で最も一般的な手法は，アクセス・パターンを予測し，キャッシュ・ミスをトリガとして，予測されたパターンでデータをプリフェッチする手法である (自動プリフェッチャ)．しかしこれらの方法は，予測が困難な不規則パターンには有効でない．不規則パターンを予測する手法も提案されているが，高コストな予測器を必要とするため現実的でない．

これらに対し我々は，命令を実際の実行より先行して実行させ，データのプリフェッチを実現する手法を提案した[3]．この方式は，予測ではなく実際に実行することにより，従来のプリフェッチャでは対応が困難であった複雑なアクセス・パターンに対してもデータのプリフェッチを行うことができるという特徴がある．

しかし，電力消費には注意が払われておらず，電力効率に問題があった．具体的には，性能向上への寄与が生じるのは，主として，キャッシュ・ミスを起すロードに関連する命令のみであるが，デスティネーション・レジスタを持つ全ての命令を先行実行していた．

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

本研究では，まず，性能を大きく低下させているイベントとして，L2キャッシュ・ミスに注目した．これまでの研究で，ごく少数のロードがキャッシュ・ミスの多くをカバーしていることが知られている[1]．そのようなロードは**delinquent ロード**と呼ばれる．そこで，本論文で提案する方式では，まず，delinquent ロードを動的に見つける．次に，それに先行する命令をバッファに格納し，そのバッファから delinquent ロードが直接あるいは間接的に依存する命令をデータフローを解析し抽出する．このような命令を我々は，**推移的生産者**あるいは**TP(transitive producer) 命令**と呼ぶ．TP 命令はマークされ，以後，それらのみを先行実行の対象とする．このようにして，性能に大きく寄与する命令のみを選択し先行実行する．

本論文では，第2章でこれまでに提案した先行実行方式について説明し，第3章で電力効率を向上させる提案手法について述べる．第4章で評価を行ない，最後に第5章でまとめる．

2. 物理レジスタの2段階解放による命令の先行実行

この章では，以前に我々が提案した先行実行方式[3]について説明する．本方式の基本は，物理レジスタをパイプラインの異なるステージで2段階で解放する方式にある．まず，この方式を説明し，次にこれを拡張した先行実行の実現方法について述べる．

2.1 物理レジスタの2段階解放

1段階目の物理レジスタの解放はリネーム・ステージで行われる．リネーム・ステージには，従来のマップ表とフリー・リストの他，**解放表** (DAT: deallocation table) と呼ぶ表を用意する．この表の各エントリは物理レジスタに対応し，当該物理レジスタを解放する命令が登録されるリオーダ・バッファ(ROB: reorder buffer) のエントリ番号を保持する．

動作は次の通りである．まず，命令の論理デスティネーション・レジスタに現在割り当てられている物理レジスタを解放し，フリー・リストに追加する．この時，解放する物理レジスタに対応する DAT のエントリに，自身が割り当てられた ROB のエントリ番号を書き込む．また，従来と同じく，フリー・リストより空き物理レジスタを得，論理デスティネーション・レジスタに割り当てる．この時，DAT を参照し，当該物理レジスタを解放する ROB エントリ番号 ROBP を得る．この ROBP は，2段階目のレジスタの解放タイミングを知るための命令のタグとなる．

命令は，命令ウィンドウにおいて，自身のデスティネーション物理レジスタの2段階目の解放を待ち合わせる．2段階目の解放は，コミット時に行われる．この時解放される物理レジスタは，従来と同じく，コミットされる命令と同じ論理レジスタに以前に割り当てられていた物理レジスタである．従来と異なるところは，解放の際に，ROB のエントリ番号 ROBP を命令ウィンドウに放送する点である．放送された ROBP が，命令ウィンドウで待ち合わせている命令の ROBP タグと一致したなら，その命令は実行結果の書き込みを許可される．

2.2 命令の先行実行

前節では，命令ウィンドウで待機している命令は，書き込み許可が得られるまで発行しないとした．しかし，書き込み許可を得る前に，ソース・オペランドが利用可能となったとき一度だけ実行することで，本実行に先立つ先行実行ストリームを生成することができる．そこで，命令ウィンドウで待機している間，ソース・オペランドが利用可能となったとき，デスティネーション・レジスタの2段階目の解放が行われる前に，1度だけその命令を実行するよう制御する．レジスタへの書き込みは行えないので，実行を終了することはできないが，先行実行が実現される．先行実行は物理レジスタが十分にあるときの命令レベル並列性を利用して進むため，本実行よりも速く進む．

3. TP 命令の動的探索による先行実行命令数の削減

本手法は，1節で述べたように，先行実行命令数をできるだけ少なくしつつ，プリフェッチの効果を最大限得ることを目的としている．このために，先行実行する命令は delinquent ロードとその TP 命令とする．本機構は，これ自体が多く電力を消費しないよう考慮されなければならない．

本機構は，delinquent ロードの検出と，TP 命令の抽出と，TP 命令実行の機構からなる．以下，順に説明する．

3.1 delinquent ロードの検出

ロードが delinquent かどうかを判断するには，一定期間での L2 キャッシュ・ミス回数を計測すればよい．この計測は，ロードに至るパス毎に行う．その理由は，3.2 節で述べる TP 探索では，その時点での delinquent ロードに至る動的命令列から TP 命令を見つけるものなので，あるロードが delinquent かどうかは，制御パスによって区別されなければならないからである．

delinquent ロード検出のために，**図1**に示すように，ロードの命令アドレスをインデクスとする **MCT**(miss

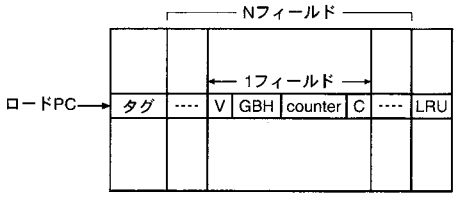


図 1 delinquent ロードを検出するための MCT

count table) と呼ぶ表を用意する．この表の各エントリは，複数のフィールドを持ち，各フィールドは当該ロードに至る広域分岐履歴 (GBH: global branch history) に対応している．1 つのフィールドは以下の内容を保持する：

- GBH
- L2 キャッシュ・ミス回数を数えるカウンタ
- delinquent ロードと判定され，TP 探索が行われたかどうかを示すビット (C フラグ)
- このフィールドの有効ビット (V フラグ)

この他，フィールドの入れ替えのためにアクセス順序である LRU 情報と，キャッシュと同様のタグを持つ．

ロードが L2 キャッシュ・ミスを生じたら，MCT の対応するエントリが保持している有効な GBH を参照する．当該ロードに至る分岐履歴が一致する GBH が存在する場合，その GBH に対応するカウンタをアップする．一致しない場合，空きフィールドがあれば，そこに本分岐履歴を書き込み，カウンタを 1 にセットし，C フラグをクリアする．空きがなければ，LRU でフィールドを置き換える．

L2 キャッシュ・ミスの頻度を測るために，一定間隔で全有効ビットをクリアし，MCT をリセットする．ロードがカウンタを 1 増加させる際，カウンタ値があらかじめ定めた閾値に到達し，C フラグがセットされていなければ，そのロードを当該 GBH において delinquent と判定し，3.2 節で述べる TP 探索を起動する．TP 探索が終了したら，C フラグをセットし，以後，現在のインターバルにおいては，再び TP 探索しないよう制御する．

ここで，MCT が参照されるのは，ロードが L2 キャッシュ・ミスを生じた時と，3.2 節で述べるトリガ命令 (これは delinquent ロードであり，L2 キャッシュ・ミスを起すロードより出現頻度は低い) をフェッチしたときだけであることに注意されたい．L2 キャッシュ・ミスの頻度は低いので，MCT アクセスにより消費される電力は小さい．

3.2 TP 命令の動的探索

delinquent ロードの TP 命令を探索するために，コミットされた命令 (実際には，PC と命令のデスティネーション・レジスタ番号およびソース・レジスタ番号) を順に蓄える **RIB**(retired instruction buffer) と呼ぶ FIFO を用意する．

TP 命令の探索は 2 段階で行われる．第 1 段階は，RIB を起動し，コミットされた命令を格納する段階である．RIB は通常は低電力モード (電源電圧降下か電源停止) にあり，トリガ信号により起動し，コミットされた命令を格納していく．そして，第 2 段階の TP 命令抽出が終了したら低電力モードに遷移する．このように必要に応じて起動することにより RIB による無駄な電力消費を抑える．トリガは次のようにしてかける．まず，MCT によって delinquent ロードが検出されたら，その命令をトリガとしてマークする．これは，命令キャッシュにそのマークを保持するフラグ (**TG フラグ**) を用意して記憶する．その後，トリガ命令がフェッチされたら MCT を参照する．分岐履歴がマッチする GBH が存在し，C フラグがセットされていなければ，この命令はトリガとされた delinquent ロードである☆．これが実行時に再度 L2 キャッシュ・ミスを生じたら，RIB を起動し，以後，トリガ命令がコミットされるまで，コミットされる命令を RIB に書き込む．

第 2 段階は，トリガ命令がコミットされると開始される．この段階では，RIB を末尾から先頭に向かって読み出し，データフロー解析を行い，TP 命令を抽出する．この解析では，データフロー・グラフを delinquent ロード (この場合トリガ命令) を始点としてエッジを逆方向に遡り，至ったノードに対応する命令を TP 命令としてマークする．次のようなアルゴリズムである．

```
1:  LIVE := sreg of delinquent load;
2:  foreach inst ∈ RIB (from tail to head)
3:    if (LIVE = φ) break;
4:    if (dreg of inst ∈ LIVE) {
5:      mark inst as "TP";
6:      LIVE := LIVE − dreg of inst;
7:      LIVE := LIVE ∪ sreg(s) of inst;
8:    }
9:  }
```

まず，delinquent ロードのソース・レジスタ (*sreg*) を集合 *LIVE* の初期値とする (1 行目)．次に，RIB を末尾から先頭に向かって順番に読み出し，*LIVE* が空になるまで以下を行う．読み出した命令のデスティ

☆ 分岐履歴は限られたパス情報しか保持していないので，正確にはトリガとされた delinquent ロードでない可能性はあるが，その頻度は低いと思われる．

ネーション・レジスタ (*dreg*) が *LIVE* に属しているなら，その命令を TP としてマークする (5 行目)．そして，*LIVE* から *dreg* を除き (6 行目)，*sreg* を *LIVE* に加える (7 行目)．

集合 *LIVE* はビット・ベクタで表す．つまり，レジスタ i が *LIVE* に属していれば，第 i 番目のビットを 1 とする．こうすれば，集合演算はビット毎の論理演算という単純なハードウェアで行える．また，TP としてのマークは，具体的には，命令キャッシュにそれを保持するビット (**TP フラグ**) を用意して記憶する．

以上の TP 探索が終了すると，今回の TP 探索をトリガした delinquent ロードの TG フラグをクリアし，TP 探索終了を示す MCT の C ビットをセットする．また，RIB を低電力モードとし，電力消費を抑える．

TP フラグは，MCT リセットの際に，全てクリアする．これば，TP 命令集合が不必要に大きくなり，不要な先行実行が生じることを防ぐためである．

命令キャッシュには，TG フラグおよび TP フラグが追加されるため，命令毎に 2 ビット新たに SRAM セルが加わることになる．しかし，これは命令長 32 ビットに対し，相対的に十分に小さく，命令キャッシュのアクセス時間や消費電力をほとんど増加させることはないと考える．

3.3 TP 命令の先行実行

フェッチされた命令が TP とマークされているなら，それらの命令は，可能であれば先行実行され，データ・プリフェッチに寄与する．これは，我々の先行実行を実現する命令ウィンドウへの命令の挿入において，先行実行を許可する既存の T-FF の初期化を制御するだけで実現できる．

4. 評価測定

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，データ・セットの大きなプログラムを多く含む SPECfp2000 の 8 本を使用した．**表 1** に，後述の base モデルにおける各ベンチマーク・プログラムの L1 データ・キャッシュおよび L2 キャッシュのミス率を示す．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た．

以下のモデルについて評価を行なった：

- **base**：通常のレジスタ解放を行ない，先行実行を行わないモデル

表 1 キャッシュ・ミス率

program	L1 miss rate	L2 miss rate
ammp	9%	32%
applu	5%	49%
apsi	1%	31%
art	48%	44%
equake	4%	41%
mesa	1%	41%
mgrid	3%	30%
swim	13%	37%

- **full**：物理レジスタ 2 段階解放により可能な限り全ての命令の先行実行を行なうモデル
- **low power**：物理レジスタ 2 段階解放による先行実行方式において，本論文で提案した手法により，先行実行命令を制限したモデル

base モデルのプロセッサの構成を**表 2** に示す．low power モデルでは，特に断らない限り，MCT は 1024 エントリ，ダイレクト・マップとし，各エントリは 4 ビットの GBH を含むフィールドを 8 つ持つとした．delinquent ロードと判別する L2 キャッシュ・ミス回数の閾値は 8 回とした．MCT と命令キャッシュの TP フラグは，1M サイクルごとにリセットする．RIB のエントリ数は 128 とし，トリガ命令により 0 サイクルで低電力モードより起動するとした．TP 命令の抽出には，RIB の 1 エントリ当り 2 サイクルを消費するとした．TP フラグを書き込むために命令キャッシュはアクセスされ，命令フェッチが行えなくなるため，実験では悲観的に，TP フラグの書き込みの有無に関わらず，RIB からの TP 命令抽出の間継続的に命令フェッチは行えないとした．

full モデル，low power モデルにおいて，データ・アクセスの空間的局所性を考慮し，先行実行時にはロード命令が要求しているデータを含むラインに加え，後続の 1 ラインを合わせてプリフェッチする．さらに，我々の以前の研究[3] では，先行実行においては，オペランドはバイパス論理によってのみ後続の命令に渡されるとしていた．このため，連続して実行される命令間でしかオペランドは受け渡せられなかった．これに対し本論文では，最近の実行結果をいくつか蓄える小容量のバッファ(**フォワーディング・バッファ**) を用意し，そこからオペランドを供給できると仮定した．本評価では楽観的に，フォワーディング・バッファのエントリ数を無限とした．

4.2 命令先行実行率

図 2 に，full モデルと low power モデルにおける全実行命令に対する先行実行した命令の割合 (**命令先行実行率**) を示す．図からわかるように，全てのベンチ

表 2　base モデルの構成

Pipeline width	8-instruction wide for each of fetch, decode, issue, and commit
ROB	128 entries
LSQ	64 entries
Instruction window	64 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	6-bit history gshare, 8K-entry PHT, 10-cycle misprediction penalty
Physical register	total 128 (64 for each of int and fp)
Mem. disambiguation	perfect

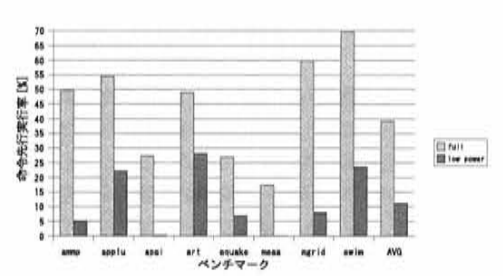


図 2　命令先行実行率

マークにおいて命令先行実行率を大きく削減することができた．特に apsi，mesa では先行実行率を 1%以下にしている．表 1 から分かる通り，これらのベンチマークの L1 キャッシュ・ミス率は 1%であり，L2 キャッシュまでアクセスする命令は少ない．よって，L2 キャッシュ・ミスが起こることも少なく，先行実行の必要性はほとどない．逆に art では L1 キャッシュ・ミス率，L2 キャッシュ・ミス率が共に高く，多くの命令が L2 キャッシュ・ミスをする．これを本実行時に回避するため多くの命令が先行実行されている．applu，swim ではキャッシュ・ミス率に対し，相対的に命令先行実行率が高い．これは少数の delinquent ロードが何度も繰り返し実行され，その都度先行実行が起動されているためと考えられる．

命令の先行実行は，命令発行論理，機能ユニット，データ・キャッシュなどを余分に動作させ，電力を消費する．特に，命令発行論理は多くの電力を消費している[2]．先行実行命令数を減らせば，それによる命令のウェイクアップ，選択の動作回数が減少し，発行論理の電力は大きく削減される．

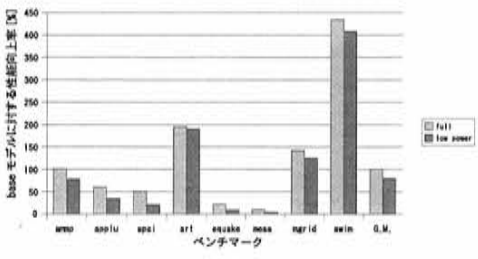


図 3　正規化した IPC

4.3 IPC

図 3 に，full モデル，low power モデルの base モデルに対する性能向上率を示す．図から分かるように，full モデルは base モデルに対し平均で 99%の性能向上率 (つまり，1.99 倍の性能) を達成しているが，low power モデルの full モデルに対する性能低下は平均で 10%と小さい．full モデルに対する性能向上率低下の原因は 3 つある．1 つは，電力効率を重視し，先行実行を delinquent ロードに対するプリフェッチのために限っていることである．このため，L2 キャッシュ・ミスの頻度が低いロードについては，プリフェッチによるレイテンシ隠蔽は行われない．2 つめは，L1 データ・キャッシュのレイテンシ隠蔽が狙われていない点である．3 つめは，本先行実行方式に備わっているロードに対する真の依存解消効果が狙われていない点である．本論文では述べなかったが，本先行実行方式では，ロード・アドレスを先行実行により計算し，本実行時にそれを利用することが可能である[3]．これによりアドレス計算とメモリ・アクセスの間の真の依存を除去できる．

4.4 追加機構の稼働率

本手法では，新たに CMT，RIB という機構を組み込んでいる．本節ではこれらが稼働している時間を評価することにより消費する電力について検討する．

図 4 に全実行サイクルに対する CMT のアクセス回数および，RIB の稼働サイクル数の割合を示す．RIB

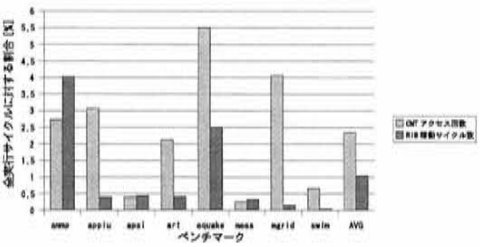
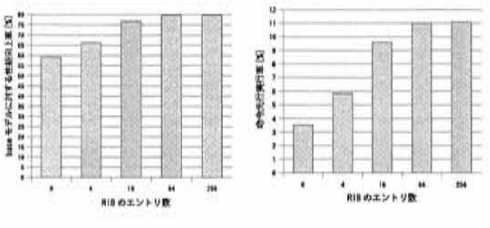
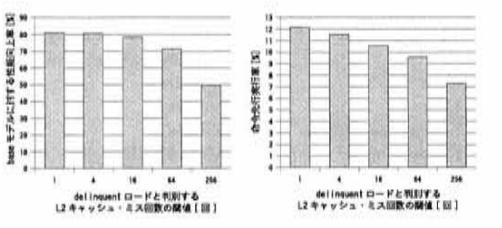


図 4　追加機構の稼働割合



(a) 性能向上率の平均　　(b) 命令先行実行率の平均

図 5　RIB のサイズによる性能向上率の変化



(a) 性能向上率の平均　　(b) 命令先行実行率の平均

図 6　delinquent ロードと判別する L2 キャッシュ・ミス回数の閾値による性能向上率の変化

の稼働期間は，3.2 節で述べたように，トリガ命令による RIB の起動から TP 探索終了までである．

図から分かるように，CMT のアクセス頻度と RIB の稼働率は極端に少ない．これは追加した機構の消費電力が小さいことを示唆している．

4.5 RIB エントリ数への依存性

RIB のエントリ数を変化させた場合の base モデルに対する性能向上率および命令先行実行率の変化を**図 5** に示す．エントリ数 0 の時は，TP 探索は行われず，delinquent ロードのみが先行実行される．一般に，RIB のエントリ数が少すぎる場合，delinquent ロードに対し，十分に先行する TP 命令を検出できなくなり，プリフェッチのタイムリー性が悪くなる．図から分かるように，RIB エントリ数 0 でも性能向上率は 59%あるが，エントリ数を増加させれば命令先行実行率は向上していき，プリフェッチのタイムリー性が良くなる．その傾向は，64 エントリで飽和し，性能向上率も 80%で飽和する．

4.6 delinquent ロードと判別する L2 キャッシュ・ミス回数の閾値への依存性

delinquent ロードと判別する L2 キャッシュ・ミス回数の閾値を変化させた場合の base モデルに対する性能向上率および命令先行実行率の変化を**図 6** に示す．一般に，閾値が大きいほど，先行実行命令数は少なくなる．これは，delinquent ロードを発見するための学習時間が長くなると共に，プリフェッチの効果を得ることができるロードを見逃す可能性が高くなるためである．図から分かるように，L2 ミス・ペナルティ隠蔽の効果をほぼ最大限得るには，閾値を 16 回程度以下にしなければならないことがわかる．

閾値が 1 回の場合，図 3 に示した full モデルの性能向上率と比較すると，9%性能向上率が低下している．この場合，1 度でも L2 キャッシュ・ミスを起せば delinquent ロードと判定されるから，この性能低下のほとんどは，L1 データ・キャッシュ・ミス隠蔽およびアドレス早期生成効果が狙われていないことにあると思われる．

5. ま と め

本論文では物理レジスタの 2 段階解放による命令先行実行方式に対し，大きな性能向上をもたらす命令のみを選択的に先行実行を行なう手法を提案した．SPECfp2000 ベンチマークを用い，評価を行なった結果，性能低下を小さく抑えつつ，先行実行命令数を大きく削減することができた．平均 10%の性能向上率の低下で，命令先行実行率を 72%削減できることを確認した．物理レジスタ 2 段階解放を組み込まないモデルに対し，平均 11%の実行命令数の増加で 80%の性能向上を達成している．

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 19500041）による補助のもとで行われた．

参 考 文 献

[1] J.D. Collins, H.Kong, D.M. Tullsen, C.Hughes, Y.-F. Lee, D. Lavery, and J. P. Shen, Speculative precomputation: Long-range prefetching of delinquent loads, In *ISCA-28*, pp. 144–154, July 2001.

[2] D. Folegnani and A. González, Energy-effective issue logic, In *ISCA-28*, pp. 230–239, July 2001.

[3] 山本哲弘, 安藤秀樹, 島田俊夫, 先行実行を利用したロード命令のレイテンシ削減および正確なスケジューリング手法, In *SACSIS 2006*, pp. 403–410, 2006 年 5 月.