Vol.2009-ARC-183 No.36
Vol.2009-OS-111 No.36
2009/4/24

物理レジスタ 2 段階解放方式の低消費電力化手法の評価

岩 本 健 吾[†1] 安 藤 秀 樹[†1]

物理レジスタ 2 段階解放による先行実行方式は，先行実行によるプリフェッチ効果により大きな性能向上を得られたが，エネルギ効率が悪かった．これは，可能な限り多くの命令を先行実行するため，無駄な先行実行が生じていたからである．これに対し，性能向上に寄与する命令のみを先行実行することにより，性能を維持しつつエネルギ消費を抑える手法が提案された．本稿では，この低消費電力化手法の，性能，エネルギ面での評価を行った．結果，先行実行命令数を 76 %削減し，2 %の性能低下で実行コアの消費エネルギを 7 %削減した．

Evaluation of Power Saving Scheme for Two-Step Physical Register Deallocation

Kengo Iwamoto[†1] and Hideki Ando[†1]

Instruction pre-execution is an effective way to prefetch data. Two-step physical register deallocation (TSD) is a pre-execution scheme. Although previous TSD study has successfully improved performance, it still has an inefficient energy consumption. This is because the TSD attempts to pre-execute as many instruction as possible. We previously proposed a power-saving scheme for the original, which pre-executes only those instructions that have great benefit. This paper evaluates the TSD and the power-saving schemes for the TSD regarding performance and energy. Our evaluation results show that our scheme reduces the dynamic pre-executed instruction count by 76%, compared with the original scheme. This reduction saves 7% energy consumption of the execution core with 2% performance degradation.

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

1. はじめに

メモリのアクセス速度の向上が，プロセッサのクロック周波数の向上に対し遅いため，ロード命令のレイテンシ (サイクル数) は増大を続けている．ロード・レイテンシを削減するため，キャッシュにより階層化したメモリ・システムにより，速度差を埋める手法が一般的に用いられているが，この手法は高コストであり，また，効果が不十分なことも多い．

この問題を解決する手法の一つに，データのプリフェッチがある．多くのハードウェアによるプリフェッチ手法が提案されている．命令の先行実行は不規則なパターンに対し効果的なプリフェッチ手法である．キャッシュ・ミスを引き起こすロード命令の先行実行により，データがメモリ階層の下位から上位に移動される．**物理レジスタ 2 段階解放**[1](**TSD**) は，命令の先行実行方式の一つである．TSD は，物理レジスタを仮に解放し，それを命令に割り当てることにより，物理レジスタ不足による，リネーム・ステージでのストールを回避している．命令が，仮に解放されたレジスタが真に解放されるのを待つ間に，ソース・オペランドが利用可能になれば，その命令は先行実行される．ただし，先行実行の結果はレジスタに書き込まれず，バイパス，またはフォワーディング・バッファ[2] と呼ばれる小さな完全連想バッファを介し，後続の依存命令に渡され，連続的に先行実行が行われる．後に，仮に解放された状態で割り当てられた物理レジスタが，真に解放されると，本実行が行われる．この際，先行実行でキャッシュ・ミスを起こしたロードは，キャッシュでヒットする．

TSD により，ロードのレイテンシは削減されたものの，従来の TSD 手法はエネルギ消費への配慮が欠けていた．TSD は可能な限り多くの命令を先行実行し，最大の性能向上を得ることを目指していた．しかし，ロードのレイテンシ削減に寄与する可能性のある命令は，キャッシュ・ミスを起こすロード命令と，それに直接または間接的に関係する命令だけである．この点を利用して，TSD を低消費電力化する手法が兵藤らにより提案された[3]．この手法では，**delinquent ロード**[4] と呼ばれる，頻繁にキャッシュ・ミスを起こす命令と，それが依存する命令列のみを選択的に先行実行する．さらに，兵藤らの手法では，先行実行する依存命令列の長さを動的に変更し，性能向上に寄与しない先行実行をさらに削減している．本論文では，この兵藤らの低消費電力化手法を，性能，エネルギ消費の面から評価する．

本論文では，第 2 節で関連研究を述べ，第 3 節で従来の TSD 手法の紹介を行う．第 4 節で兵藤らの手法の delinquent ロードとその依存命令列を判別する手法を紹介し，第 5 節では，先行実行命令列長の最適化手法の紹介をする．そして，第 6 節で評価を行い，第 7 節でまとめる．

2. 関 連 研 究

先行実行によるプリフェッチの研究は多数行われている．これらの手法のほとんどは，先行実行の対象を静的に決定し，スレッドとして実行する．

© 2009 Information Processing Society of Japan

マルチスレッドによる先行実行方式の中で，先行実行対象を動的に決定する手法に，Collinsらによる手法[5]がある．この手法も，エネルギ効率の点から，delinquent ロードに着目しているが，この手法は高性能化のために貪欲に先行実行し，省電力化のために先行実行命令列長を最適化するといったことはしていない．

マルチスレッド環境を必要としない先行実行手法に，runahead 実行[6]がある．この手法は，L2 キャッシュ・ミス発生時に，後続命令を先行実行することでメモリ・レベル並列性を利用するものである．

runahead 実行のエネルギ効率を改善する手法が，Mutlu らにより提案された[7]．しかし，これらの手法は delinquent ロードへの着目とも，先行実行命令列長の最適化とも違う手法であった．

また，TSD はレジスタ・ファイルを実効的に大きくする手法とも言える．同様の目的で，物理レジスタの遅延割り当て[8]や，早期解放[9]の手法が提案されている．早期解放手法は，投機的にレジスタを解放するため，投機に失敗した場合に大きなペナルティが生じる．遅延割り当ては，命令実行終了時に物理レジスタを割り当てる．アウト・オブ・オーダでの割り当てによるデッドロックを回避するため，機構が複雑化する問題がある．

3. TSD による先行実行

この節では，TSD 手法の説明を行う．

3.1 TSD の効果

まず，TSD による先行実行の効果を説明する．キャッシュ・ミスを起こすロード命令を含む 4 つの依存命令の実行のタイミングを，従来のプロセッサの場合を**図 1**(a) に，TSD の場合を図 1(b) に示す．図からわかるように，物理レジスタ不足によるストールが発生しないため，先行実行は，従来の実行より早く開始される．これにより，ロード命令によるキャッシュ・ミスが早期に発生し，この際データがメモリ階層の上位にコピーされる．その結果，本実行の際には L1 キャッシュでヒットし，性能向上が得られる．

3.2 TSD 手法による物理レジスタの解放

第 1 段階の解放

第 1 段階の解放はリネーム・ステージで行われる．マップ表，フリー・リストに加え，**Deallocation Table**(**DAT**) と呼ぶ表を用意する．DAT の各エントリは，物理レジスタに対応し，各物理レジスタを真に解放 (第 2 段階の解放) する命令に割り当てられた，ROB のエントリ番号を記憶する．

動作は次の様になる．まず，命令がリネーム・ステージに達すると，同じ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，仮に解放 (第 1 段階の解放) し，フリー・リストに加える．同時に，この命令に割り当てられた ROB のエントリ番号を，今仮の解放をされた物理レジスタに対応する DAT のエントリに記録する．これにより，第 1 段階の解放状態のレジスタが，どの命令のコミットで解放されるかが，DAT に記録される．

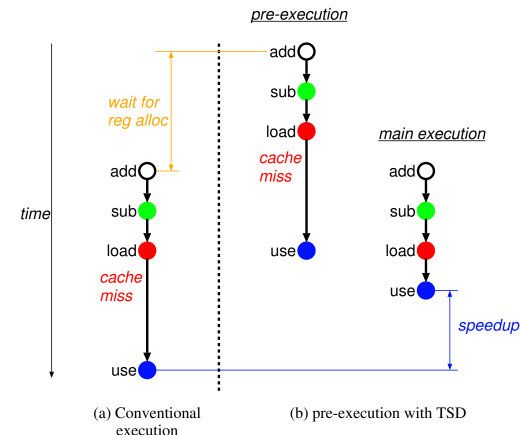


図 1 TSD による先行実行の効果

加えて，フリー・リストから利用可能な物理レジスタを取得し，論理デスティネーション・レジスタに，従来の手法と同様に割り当てる．この際，割り当てた物理レジスタに対応する DAT のエントリから，その物理レジスタを解放する命令の，ROB エントリ番号 (**ROBP**) を取得する．ROBP は，命令ウインドウ内で，デスティネーション・レジスタの第 2 段階の解放が行われたタイミングを知るためのタグとして用いられる．

第 2 段階の解放

リネームされた命令は命令ウインドウに挿入され，そのデスティネーション・レジスタの第 2 段階の解放を待つ．この解放は従来手法と同様，コミットされる命令の論理デスティネーション・レジスタに，以前に割り当てられていた物理レジスタの解放である．従来手法との違いは，コミットされた命令の ROB エントリ番号，つまり ROBP を，命令ウインドウに放送する点にある．放送された ROBP は，命令ウインドウ内の各命令が，リネーム時に取得した ROBP タグと比較される．一致したならば，その命令のデスティネーション・レジスタが真に解放されたということであり，実行結果の書き込みが許可され，発行が可能になる．

3.3 先 行 実 行

前節では，結果の書き込みが許可されるまで，その命令の発行はできないものとした．し

ⓒ 2009 Information Processing Society of Japan

Vol.2009-ARC-183 No.36
Vol.2009-OS-111 No.36
2009/4/24

かし，ソースが揃えばこれらの命令の実行は可能である．この場合，実行結果の書き込みはできないものの，バイパス論理やフォワーディング・バッファを介して，依存命令へ結果を渡すことは可能である．そのため，依存命令の列が，先行実行ストリームを形成する．物理レジスタによる資源制約がないため，本実行のストリームよりも速く実行が進む．

先行実行された命令のレディフラグは，発行の後リセットし，命令は命令ウインドウに残す．後に，レジスタが解放され，かつ，再びレディフラグがセットされれば，命令は再度実行され，結果が物理デスティネーション・レジスタに書き込まれる．

4. 選択的先行実行による低消費電力化

この節では，選択的先行実行による TSD の低消費電力化についての説明をする．

プログラム中のロード命令には，頻繁にキャッシュ・ミスを起こす命令が存在する．これらの命令は，プログラム中の静的な命令の内，わずかにしか存在しないが，キャッシュ・ミスの内，かなりの割合を引き起こしている．そのような頻繁にキャッシュ・ミスを起こすロード命令を，delinquent ロードと呼ぶ[4]．キャッシュ・ミスのうち，多くの割合を delinquent ロードが引き起こしているため，delinquent ロードに対し，先行実行ができればプリフェッチの効果により，高い性能向上が期待される．

提案する低消費電力化方式は，先行実行の対象を delinquent ロードと，それが直接または間接的に依存する命令 (**TP 命令**:**Transitive Producer**) に絞る．それにより，高い性能を維持しつつ，無駄な先行実行を避け消費エネルギを削減することを目的とする．

4.1 delinquent ロードの検出

delinquent ロードは，頻繁に L2 キャッシュ・ミスを起こす．delinquent ロードを検出するため，一定の期間内での各ロード命令の L2 キャッシュ・ミス回数をカウントする．このために，ロード命令の PC をインデクスとする表である **Miss Count Table**(**MCT**) と呼ぶ表を用意する．この表の各エントリは次のようなフィールドを持つ．

- 有効フラグ (V フラグ)
- L2 キャッシュ・ミス回数カウンタ
- TP 命令探索が既になされたか否かを示すフラグ (C フラグ)

ロード命令が L2 キャッシュ・ミスを起こした際，MCT の対応するエントリを読み出す．そのエントリが有効ならば，カウンタをインクリメントする．さもなくば，カウンタを 1 にし，C フラグをクリアし，V フラグをセットして初期化する．ロード命令がカウンタをインクリメントした際，カウンタ値が閾値を超え，かつ C フラグがセットされていなかった場合，そのロード命令を delinquent ロードとみなす．そして，その delinquent ロードの TP 命令の探索を開始する．

なお，MCT へのアクセスは，L2 キャッシュ・ミス発生時と，delinquent ロードがデコードされる際のみに行われる．これらの頻度は低いため，MCT へのアクセスによるエネルギ消費は小さいと思われる．

4.2 TP 命令の動的探索

delinquent ロードの TP 命令の探索 (**TP 探索**) のため，**Retired Instruction Buffer**(**RIB**) と呼ぶ FIFO バッファを用いる．TP 探索は 2 段階で行われる．

第 1 段階は，コミットされる命令を RIB に取り込むことである．RIB は通常，スリープ状態 (電力の供給を止めた状態) で待機している．トリガ命令としてマークされた命令のデコードをトリガとして，RIB は起動される．我々の実装では，delinquent ロードをトリガ命令とする．delinquent ロードが MCT により発見された際，トリガ命令としてマークされる．トリガ命令を示すため，命令キャッシュの各命令に **TG フラグ**と呼ばれるフラグを設ける．トリガ命令がデコードされた際，MCT がチェックされる．対応するエントリの C フラグがセットされていなければ，TP 探索がまだ行われていないということを示す．この場合，トリガ命令により RIB が起動され，トリガ命令がコミットされるまでの間，コミットされた命令は順に RIB に記録される．

第 2 段階は，トリガ命令がコミットされる際に開始される．delinquent ロードの TP 命令は，RIB 内の命令を末尾から先頭まで読み出し，データフローを解析することにより探索される．この解析は，概念的にはデータフロー・グラフを delinquent ロードのノードから逆順にたどることにより行われる．

図 2 にアルゴリズムを示す．最初に，delinquent ロードのソース・レジスタ ($sreg$) を，LIVE 集合の要素とする (第 1 行)．次に，RIB 内の命令を末尾から先頭まで読み出し，次のような手順を，LIVE 集合が空になるまで行う．もし，読み出した命令のデスティネーション・レジスタ ($dreg$) が LIVE 集合の要素ならば，その命令を TP 命令としてマークする (第 5 行)．次に，$dreg$ を LIVE 集合から取り除き (第 6 行)，$sreg$ を LIVE 集合に加える (第 7 行)．

LIVE 集合は，ビット・ベクタによって表現する．すなわち，もし，レジスタ i が LIVE 集合の要素ならば，第 i ビットをセットする．これにより，上記アルゴリズムの実行を単純なハードウェアで実装できる．

なお，TP 命令を示すため，命令キャッシュの各命令に，**TP フラグ**と呼ばれるフラグを設ける．TP または TG フラグのセットされた命令は，可能ならば先行実行される．

TP 探索完了後，対象の delinquent ロードに対応する MCT エントリの C フラグがセットされる．これにより，次に MCT がリセットされるまでの間，対応する delinquent ロードに対する TP 探索は行われない．そして，RIB はエネルギ消費を抑えるため，スリープ・モードにする．

全ての TP，TG フラグは，定期的にクリアされる．これにより，プログラムの実行フェイズの変化に追従する．

我々は delinquent ロードをトリガ命令としたものの，どの命令をトリガ命令とするかは，

© 2009 Information Processing Society of Japan

```
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:  }
```

図 2 TP 探索のアルゴリズム

性能に影響を与えうる．しかし，理想的条件として，十分に大きな RIB(1K エントリ) を常に作動させていた場合でも，性能は 0.5 %しか変わらなかった (RIB サイズによらず，TP 探索に時間はかからないものとした)．このため，我々の設定したトリガのタイミングは十分に早いといえる．一方，早期にトリガをかけることにより，必要以上に TP 命令が発見される可能性がある．しかし，不必要な TP 命令は，次節に示す最適化により，刈り取られる．

5. TP 命令列長の最適化

プリフェッチの効果は，delinquent ロードの先行度に依存する．先行度とは，delinquent ロードの先行実行から本実行までのサイクル数である．TSD では，物理レジスタによる資源制約を取り除くことによって先行度を得ている．しかし，そのような効果をもたらさない TP 命令も存在する．このような命令の先行実行は，エネルギの無駄である．この先行実行をなくすため，命令列長の最適化を行う．

5.1 最適化アルゴリズム

TP 命令列長の最適化は，TP 命令列の先頭から，適切な数の TP 命令を刈り取る (prune) ことによって行う．これは，命令列の先頭の命令 (**HD 命令**) から，特定の命令数，先行実行を抑制することにより行われる．この実装のため，まず，命令キャッシュの各命令に，HD 命令を示すフラグである，**HD フラグ**を設ける．さらに，**TP Pruning Table**(**TPPT**) と呼ばれる表を用意する．TPPT の各エントリは，HD 命令に関連付けられ，*nprune*，*max_pcy*，*ntp* の 3 つのフィールドを持つ．*nprune* は刈り取る命令数を示す．*max_pcy* は，HD 命令に対応する delinquent ロードの過去の実行時の最大先行度を記憶する．*ntp* は，TP 命令列に含まれる TP 命令の数を示す．

```
1:  pcy := precedence cycles of delinquent load;
2:  if (pcy ≤ max_pcy) {
3:    if (max_pcy − pcy < PCY_th)
4:      nprune := min(nprune + Δt, ntp);
5:    else
6:      nprune := max(nprune − Δt, 0);
7:  } else {
8:    nprune := max(nprune − Δt, 0);
9:    max_pcy := pcy;
10: }
```

図 3 TP 命令刈り取りの学習アルゴリズム

TP 探索終了時に，探索した TP 命令列の HD 命令に対応する TPPT のエントリを初期化する．*nprune* と *max_pcy* を 0 にし，*ntp* には探索により得た TP 命令の数を書き込む．

刈り取る命令数は，学習により決定される．アルゴリズムを**図 3** に示す．delinquent ロードの実行が終了した際，先行度 *pcy* が得られる (第 1 行)．もし *pcy* が現在の *max_pcy* と同じかまたは小さい場合，*pcy* が *max_pcy* からどれだけ減少したかを評価する．減少量が閾値 PCY_{th} より小さい場合 (第 3 行)，刈り取り命令数をさらに増やしても先行度に影響はないものとみなし，*nprune* を定数 Δ_t だけ増加させる (第 4 行)．減少量が閾値 PCY_{th} より大きい場合，過去に過剰な刈り取りが行われたものとみなし，*nprune* を Δ_t だけ減少させる (第 6 行)．*pcy* が *max_pcy* より大きい場合 (第 7 行)，刈り取りの基準値が変わったと検知し，刈り取り命令数を徐々に初期状態に戻してゆく．つまり，*nprune* を Δ_t だけ減少させ (第 8 行)，*max_pcy* を更新する (第 9 行)．

この最適化アルゴリズムの実行によるエネルギ消費のほとんどは，TPPT へのアクセスによるものである．TPPT へのアクセスは，TP 探索完了時，HD 命令フェッチ時，delinquent ロード実行後の，このアルゴリズムの開始と終了時に行われる．これらのイベントの頻度は低いため，エネルギ消費は小さいと思われる．

6. 評　価

評価には，SimpleScalar Tool Set version 3.0a[10] をベースにシミュレータを作成し，使用した．命令セットは，MIPS R10000 ISA の拡張版である，SimpleScalar/PISA を使用した．ベンチマーク・プログラムは，SPECfp2000 から 8 つを使用した．プログラムのコ

© 2009 Information Processing Society of Japan

表 1 キャッシュ及びメモリ・アクセス統計

program	miss rate			memory
	L1 D-cache	stream buffer	L2	access rate
ammp	9%	59%	29%	1.59%
applu	5%	68%	29%	0.96%
apsi	1%	20%	3%	0.00%
art	48%	31%	85%	12.79%
equake	4%	31%	22%	0.29%
mesa	1%	18%	10%	0.02%
mgrid	3%	3%	9%	0.01%
swim	13%	59%	13%	1.03%

ンパイルは，gcc ver.2.7.2.3 に，-O6 -funroll-loops のオプションを付けて行った．**表 1** に，後述する base モデルでの実行時の，各ベンチマークのメモリ・アクセス統計を示す．ただし，L2 ミス率と，メモリ・アクセス率には，ストライド・プリフェッチャによるアクセスは含まれていない．

また，エネルギ消費の評価は次のように行った．ある部品によるエネルギ消費は，一度の使用で消費されるエネルギに，総使用回数を乗ずることにより求める．各部品の使用回数は，SimpleScalar によるシミュレーションにより得られる．各部品の一度の使用で消費されるエネルギは，アレイ構造と CAM については，CACTI[11] を，より現実的な配線パラメータを使用するよう修正したものを用いて求めた．その他の部品については Wattch[12] のモデルを用い，使用するプロセス・ルールに合わせてエネルギをスケーリングしたものを使用した．プロセス・ルールは 70nm プロセス，電源電圧は 1.0V を使用した．ただし，本研究で作成したシミュレータでは，静的電力は測定できない．そのため，評価は動的電力のみで行われている．

次の 2 つのモデルを評価した．一つは，TSD により可能な限り多くの命令を先行実行する full モデル．もう一つは，第 4 節および第 5 節で述べた手法により選択的に先行実行を行う energy-efficient モデルである．また，これらとの比較対象として，TSD を使用しない base モデルについても測定した．

3 つのモデルのプロセッサ共通のパラメータを**表 2** に示す．TSD のプリフェッチ効果を過大評価しないよう，per-load stride predictor を用いたストライド・プリフェッチャを搭載し，L1 データ・キャッシュの汚染を避けるため，L1 データ・キャッシュと L2 キャッシュの間にストリーム・バッファを搭載した．また，メモリ曖昧性の除去を完全としている．TSD は，ロード命令のアドレス計算を先行実行することにより，メモリ曖昧性の除去について良い影響を与える．この効果を除いて公平な評価をするため，ベースのシミュレータにメモリ依存予測器を搭載する代わりに，メモリ曖昧性の除去を完全とした．

TSD の効果は，ROB サイズと物理レジスタ数のバランスに強い影響を受ける．このバランスをとるため，次の式を用いる．

$$Npregs = \alpha \times ROBsize + Nlregs \quad (1)$$

$Npregs$ 及び $Nlregs$ は，それぞれ物理，論理レジスタ数を表す．係数 α は，整数または浮動小数点のデスティネーション・レジスタを持つ動的命令と，全動的命令数の比を表す．式 (1) から導き出される ROB サイズと物理レジスタ数は，次の 2 点から，バランスが取れていると言える．第 1 に，ROB サイズがインフライト命令数の上限となるため，$\alpha \times ROBsize$ のインフライト命令が物理レジスタを必要とする．第 2 に，コミットされた各論理デスティネーション・レジスタは，物理レジスタを必要とする．次に，物理レジスタ数は，整数と浮動小数点で同数に分けた．これは，使用したベンチマーク・プログラムにおいて，整数及び浮動小数点それぞれのデスティネーション・レジスタを持つ動的命令数が，同程度であったためである．**表 3** に，デスティネーション・レジスタの種類ごとの動的命令数の割合を示した．

表より，α は 0.83 とした．ROB サイズは 128 であるので，$Npreg$ は 170($128 \times 0.83 + 64$) となる．これを，最も近い 8 の倍数に丸め，物理レジスタ数は 168 と決定した．

full 及び energy-efficient モデルそれぞれの，固有のパラメータを**表 4** に示す．

また，TP 探索には，RIB の 1 エントリ毎に，10 サイクルを要するものと仮定した．さらに TP 探索中には，TP フラグをセットするため，命令キャッシュへのアクセスが生じる．そのため，TP 探索中はフラグの書き込みを行うか否かにかかわらず，命令のフェッチがストールするものと保守的に仮定した．

表 2 基本パラメータ

Pipeline width	4-instruction wide for each of fetch, decode, issue, and commit
Branch prediction	6-bit history gshare, 8K-entry PHT, 512-entry, 4-way BTB
	10-cycle misprediction penalty
ROB	128 entries
LSQ	64 entries
Instruction window	64 entries
Physical registers	168 (int 84, fp 84)
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
Data prefetcher	stride prefetcher, incremental prefetching
Stream buffer	8-way4KB each, 32B line, 1-cycle hit latency
Mem. disambiguation	perfect

© 2009 Information Processing Society of Japan

Vol.2009-ARC-183 No.36
Vol.2009-OS-111 No.36
2009/4/24

表 3 デスティネーション・レジスタの種類による動的命令数割合

program	type of destination register			
	int	fp	other	none
ammp	24%	55%	1%	20%
applu	60%	31%	1%	8%
apsi	59%	22%	9%	10%
art	37%	39%	2%	22%
equake	58%	16%	0%	26%
mesa	48%	21%	4%	27%
mgrid	43%	55%	1%	2%
swim	48%	48%	0%	4%
AVG	47%	36%	2%	15%

表 4 full 及び energy-efficient モデルのパラメータ

common	8-entry, fully-associative forwarding buffer
energy-efficient model	1024-entry tagless MCT. 64-entry RIB. Threshold of miss count to identify delinquent load is 16. Reset interval of MCT and TP, TG, and HD flags is 1M cycles. 128-entry tagless TPPT. PCY_{th} is 30 cycles. Δ_t is 3 cycles.

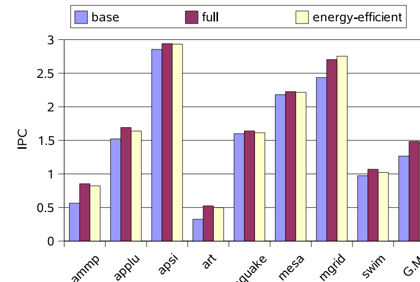


図 4 IPC

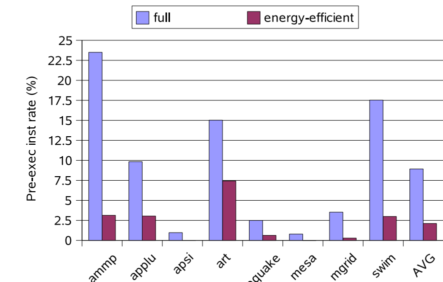


図 5 命令先行実行率

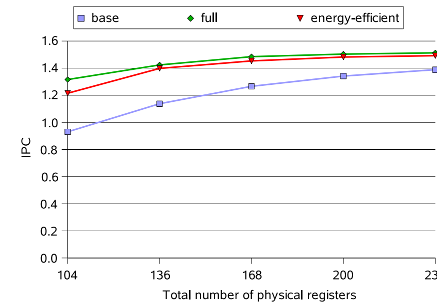


図 6 物理レジスタ数を変えた場合の IPC

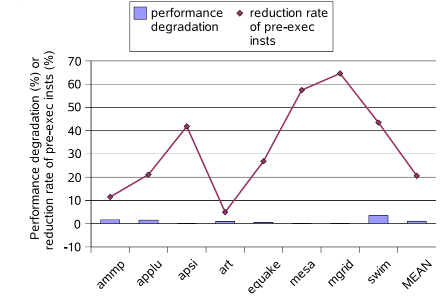


図 7 TP 命令列長最適化の効果

6.1 先行実行率

図 5 に full，energy-efficient 両モデルの命令の先行実行率を示した．先行実行率とは，コミットされた命令のうち，先行実行された命令の割合である．図からわかるように，energy-efficient モデルは，先行実行率が顕著に下がっている．また，先行実行命令の削減率は，メモリ・アクセス率と負の相関がある (表 1)．たとえば，apsi，mesa，mgrid では，削減率が 90 %を超えているのに対し，メモリ・アクセス率は非常に低い．対照的に，art では，削減率が 51 %と，あまり高くないの対し，メモリ・アクセス率は非常に高い．

6.2 性　　能

図 4 に，base，full，energy-efficient モデルの IPC を示す．energy-efficient モデルの，full モデルに対する性能低下率は，平均 2 %と小さい．この性能低下の理由は，3 つ考えられる．まず第 1 に，この手法では，稀にしか L2 キャッシュ・ミスを起こさない命令のレイテンシを隠蔽することはできない．しかし，このことが性能に与える影響は，**図 9** から見て取れるように小さい．delinquent ロードと判定する閾値を 1 から 64 まで変えても，性能低下率はほとんど変わっていない．第 2 に，この手法ではほとんどの L1 キャッシュ・ミス・ペナルティを隠蔽できない．第 3 に，この手法は，TSD の機能の一つである，アドレス計算とロードの間の真の依存を取り除く効果からほとんど恩恵を受けられない．TSD では，アドレス計算命令を先行実行した際，結果を LSQ に書き込む．そのため依存するロード命令は，計算された値を用い，アドレス計算を待つことなく本実行できる．これによりアドレス計算とロードの真の依存が取り除かれる．しかし，本手法はこの利点のほとんどを失っている．

energy-efficient モデルの性能は，full モデルに対してはわずかに低下しているが，base モデルに対しては依然として 15 %高い性能を発揮している．

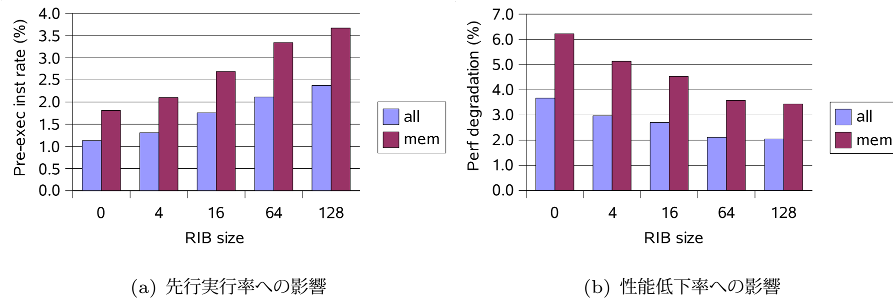
図 6 に 3 つのモデルの，物理レジスタ数を変えた際の IPC を示す．物理レジスタ数の増加に従い，2 つの TSD モデルの base モデルに対する性能向上率が低下している．しかし，energy-efficient の性能は，物理レジスタ数 232 の場合においても，base モデルに対し，7 %高い性能を維持している．

6.3 TP 命令列長の最適化の効果

図 7 に TP 命令列長の最適化の効果を示す．折れ線は先行実行命令数削減率を，棒グラフは最適化なしのモデルに対する最適化を行ったモデルの性能低下率を示す．グラフの一番右の値は，先行実行命令数削減率と性能低下率の，それぞれ算術平均と幾何平均である．

© 2009 Information Processing Society of Japan

Vol.2009-ARC-183 No.36
Vol.2009-OS-111 No.36
2009/4/24



(a) 先行実行率への影響　(b) 性能低下率への影響

図 8　RIB サイズによる影響

図からわかるように，最適化は，先行実行命令数の削減には非常に効果的である (平均 21 %削減)．また，性能に対する悪影響もわずかである (平均 1 %低下)．

6.4　RIB サイズの影響

energy-efficient モデルの RIB サイズを変化させた場合の先行実行率と，full モデルに対する性能低下率を，それぞれ**図 8**(a) および (b) に示す．各 RIB サイズで，左の棒グラフはすべてのベンチマークの平均を，右の棒グラフは，メモリ・インテンシブなプログラム (ammp，applu，art，equake，swim) のみの平均を表している．

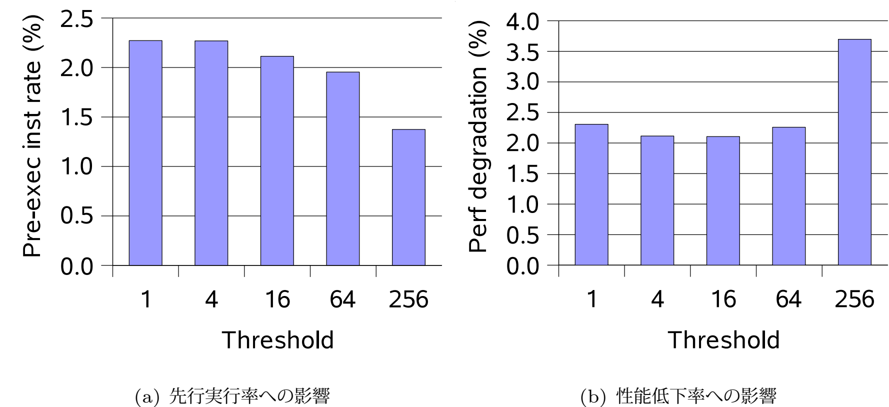
RIB サイズが増加するに従い，先行実行率は高くなっている．この率が高くなるにつれ，プリフェッチの的確さも向上し，その結果性能低下率が低下する．これらの傾向は，メモリ・インテンシブなプログラムでは，より顕著になる．

6.5　Delinquent ロード判定閾値による影響

energy-efficient モデルの delinquent ロード判定閾値を変化させた場合の先行実行率と，full モデルに対する性能低下率を，それぞれ図 9(a) および (b) に示す．一般に，閾値を大きくするにつれ，delinquent ロードと判断される命令は減少し，判定のための学習時間が長くなる．このため，図からわかるように，閾値を増加させて行くと大きな性能低下が生じるが，先行実行率は低下する．閾値を 1 から 16 にした際，わずかではあるが性能が上がっている．これは，TP 探索中に生じる命令フェッチのストールが閾値を増やしたことにより減少し，その利点がプリフェッチの減少による損失を上回ったことによるものである．

6.6　エネルギ消費

本手法により，先行実行率が削減され，命令ウインドウ，データ・キャッシュ，機能ユニットといった，命令実行に関わる資源のエネルギ消費は減少する．しかし，この手法は追加のハードウェアを必要とし，それらが追加のエネルギを消費する．このエネルギ消費のほとんどは，命令キャッシュに追加したフラグ，MCT，RIB，TPPT によって生じる．しかし，CACTI によるシミュレーションを行った結果，フラグによる命令キャッシュの消費エネルギの増加は，6 %にとどまることがわかった．**表 5** に，MCT，RIB，TPPT それぞれのサイズと動作率を示す．動作率とは，全実行サイクル数に対し，各ハードウェアが動作していたサイクル数の割合である．表から判るように，どのハードウェアもサイズが小さく，動作率も低い．このため，これらの消費するエネルギは小さいことが予想される．



(a) 先行実行率への影響　(b) 性能低下率への影響

図 9　Delinquent ロード判定閾値による影響

表 6 に，TSD 手法に関係する部品の平均エネルギ消費量を，base，full，energy-efficient の各モデルについて示す．各部品は，実行コア，TSD オーバーヘッド，energy-efficient 手法のオーバーヘッドの 3 種に分類されている．各部品の消費エネルギは，base モデルの実行コアの消費エネルギで正規化されている．

表から，energy-efficient モデルの実行コアの消費エネルギは，base モデルよりまだ 4 %大きいが，full モデルに対しては 7 %削減されている．また，energy-efficient モデルのエネルギのオーバーヘッドは，実行コアの 2 %でしかなく，非常に小さい．

TSD によるエネルギのオーバーヘッドは，full，energy-efficient 共に，base モデルの実行コアの消費エネルギの 12 %である．さらにエネルギ効率を上げるため，これらのオーバーヘッドを削減することが今後の課題である．

6.7　命令キャッシュのアクセス時間のオーバーヘッド

energy-efficient 手法では，TG，TP，HD の 3 つのフラグを命令キャッシュ内の各命令に追加している．これにより，命令キャッシュのアクセス時間が延びる可能性がある．これ

© 2009 Information Processing Society of Japan

表 5 MCT，RIB，TPPT のサイズ及び動作率

	MCT	RIB	TPPT
size	0.8KB	0.4KB	0.3KB
operation rate	6.3%	0.9%	5.4%

表 6 実行コアの消費エネルギで正規化した消費エネルギ

category	component	model		
		base	full	energy-efficient
execution core	instruction window	0.32	0.36	0.34
	LSQ	0.07	0.08	0.07
	register files	0.09	0.10	0.10
	function units	0.28	0.31	0.29
	D-cache	0.23	0.27	0.24
	D-TLB	0.00	0.01	0.01
	total	1.00	1.12	1.04
TSD overhead	DAT		0.05	0.05
	ROBP broadcast/match		0.06	0.06
	forwarding buffer		0.00	0.00
	total		0.12	0.12
energy-efficient scheme overhead	I-cache flags			0.01
	MCT			0.01
	RIB			0.00
	TPPT			0.00
	total			0.02

に対し，CACTI により，アクセス時間を評価した．その結果，データ・アレイのワード線遅延が，キャッシュのアクセス時間の 2 %増加した．しかし，キャッシュ・アクセスのクリティカル・パスは，タグ・アレイへのアクセス経路である．そのため，この遅延の増加は，キャッシュのアクセス時間に影響しない．

7. ま と め

本稿では，TSD の低消費電力化手法の評価を行った．この手法は，性能向上に大きく寄与する命令を，選択的に先行実行する．SPECfp2000 による評価の結果，先行実行率の大幅な減少が見られ，従来の TSD 手法に対し平均 76 %減少した．また，性能面では，従来の TSD 手法に対し，平均 2 %の性能低下が生じたものの，TSD を行わないモデルに対しては，15 %高い性能を発揮している．エネルギ面では，TSD を行わないモデルに対し，実行コアの消費エネルギが 4 %増加しているものの，従来の TSD モデルに対しては，7 %の減少となった．しかし，TSD によるエネルギのオーバーヘッドは，依然無視できない量であり，その削減が今後の課題である．

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 19500041）による補助のもとで行われた．

参 考 文 献

1) Yamamoto, A. et al.: Data Prefetching and Address Pre-Calculation through Instruction Pre-Execution with Two-Step Physical Register Deallocation, *Proceedings of the Eighth Workshop on Memory Performance: Dealing with Applications, Systems and Architectures*, pp.41–48 (2007).
2) Borch, E. et al.: Loose Loops Sink Chips, *Proceedings of the 8th Annual International Symposium on High-Performance Computer Architecture*, pp.299–310 (2002).
3) 兵藤一永，安藤秀樹：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，情報処理学会研究報告， No.2007-ARC-174, pp.169–174 (2007).
4) Collins, J.D. et al.: Speculative precomputation: long-range prefetching of delinquent loads, *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pp.14–25 (2001).
5) Collins, J.D. et al.: Dynamic speculative precomputation, *Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 306–317 (2001).
6) Mutlu, O. et al.: Runahead Execution: An Effective Alternative to Large Instruction Windows, *IEEE Micro*, Vol.23, No.6, pp.20–25 (2003).
7) Mutlu, O. et al.: Techniques for Efficient Processing in Runahead Execution Engines, *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pp.370–381 (2005).
8) Monreal, T. et al.: Delaying Physical Register Allocation Through Virtual-Physical Registers, *Proceedings of the 32th Annual International Symposium on Microarchitecture*, pp.186–192 (1999).
9) Moudgill, M. et al.: Register renaming and dynamic speculation: An alternative approach, *Proceedings of the 26th Annual International Symposium on Microarchitecture*, pp.202–213 (1993).
10) SimpleScalar LLC: SimpleScalar LLC, http://www.simplescalar.com/.
11) Shivakumar, P. and Jouppi, N.P.: CACTI 3.0:An Integrated Cache Timing, Power, and Area Model, *WRL Research Report 2001/2* (2001).
12) Brooks, D., Tiwari, V. and Martonosi, M.: Wattch: a Framework for Architectural-Level Power Analysis and Optimizations, *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp.83–94 (2000).

© 2009 Information Processing Society of Japan