計算機アーキテクチャ 149－13
（２００２．８．２２）

最近の値の局所性に着目した共有化による物理レジスタ削減

大熊 穣[†] 片山清和[†] 小林良太郎[†]
安藤秀樹[†] 島田俊夫[†]

SMT プロセッサでは複数のスレッドを同時に実行するため大きなレジスタ・ファイルが必要であり，そのアクセスがクロック速度を決定するクリティカル・パスである．本論文では，頻繁に生成される値について物理レジスタを共有化することにより，必要な物理レジスタ数を削減する手法を提案する．本手法は，これまでの物理レジスタの共有化手法とは異なり単純なハードウェアで実現することができる．評価の結果，物理レジスタを共有しない従来手法では物理レジスタ数を削減すると急速に性能が低下するのに対し，本手法ではこの性能低下を大きく抑制できることがわかった．

Reducing Physical Registers with Sharing Techniques by Exploiting Recent-Value Locality

Minoru Okuma,[†] Kiyokazu Katayama,[†] Ryotaro Kobayashi,[†]
Hideki Ando[†] and Toshio Shimada[†]

Since SMT processors require a large register file to execute multiple threads simultaneously, the register file access is a critical path which determines the clock cycle time. This paper proposes a scheme that reduces the required number of physical registers by sharing physical registers for frequently-generated values. Our scheme can be implemented by simple hardware unlike previous sharing schemes. The evaluation results show that the performance of the processor without sharing techniques goes down rapidly with reducing the number of physical registers, but our scheme suppresses this performance degradation significantly.

1. はじめに

近年，次世代のアーキテクチャとして**SMT**(Simultaneous Multi-Threading) アーキテクチャ[1] が注目されている．SMT は，複数のスレッドの命令を，スーパスカラ・プロセッサの機能ユニットに毎サイクル同時に発行する技術である．これにより，SMT プロセッサではハードウェア資源を有効に活用することができる．しかし，SMT プロセッサでは複数のスレッドの論理レジスタを同時に扱わなければならないので，多くの物理レジスタが必要となる．最低限必要な物理レジスタ数は，論理レジスタ数に同時実行スレッド数を乗じたものになる．高い性能を得るためには，この数にレジスタ・リネーミングを行うための物理レジスタ数をさらに加える必要がある．

レジスタ・ファイルのアクセス時間は，物理レジスタ数とポート数の両方に依存する[2]．多くの物理レジスタ数とポート数を持つ現在のスーパスカラ・プロセッサではレジスタ・ファイルのアクセス時間が増加し，クロック速度を決定するクリティカル・パスとなっている．SMT プロセッサではスーパスカラ・プロセッサより多くの物理レジスタが必要になるため，レジスタ・ファイルのアクセス時間はさらに増加する．したがって，SMT プロセッサではレジスタ・ファイルのアクセスがクロック速度に大きな影響を与えることは明白であり，レジスタ・ファイルのアクセス時間の短縮は非常に重要な課題である．

レジスタ・ファイルのアクセスがクリティカル・パスとなることを避ける１つの方法は，レジスタ・ファイルのアクセスをパイプライン化することである．この方法は，回路設計のみで解決できるという点で良い方法であるが，バイパス論理が複雑になるという欠点がある[3]．また，パイプライン段数が深くなることにより分岐予測ミス・ペナルティが増加するという欠点もある．

これに対し，Monreal らは物理レジスタの割り当てをリネーム時ではなく命令の実行終了時に行い，レジスタの生存する期間を短縮することにより，物理レジスタ数を削減している[4]．この手法を拡張し物理レジスタをさらに削減する方法として，Jourdan らは**最近の値の局所性**を利用し，複数の命令間で物理レジスタを共有する手法を示している[5]．ここで最近の値の局所性とは，時間的に近い命令同士が同じ値を生成することである．彼らの手法は，高い命令レベル並列性を得るために必要な物理レジスタ数を大幅に削減できる可能性がある．しかし，彼らの手法では，ある命令が

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

物理レジスタを共有可能であるかどうかを判定する動作が必要である．この判定動作は，連想検索によって実現されるため，クロック速度に悪影響を与える可能性がある．

そこで本論文では，**頻繁な値の局所性**[6] に着目し，連想検索を行うことなく物理レジスタを共有化する手法を提案する．ここで頻繁な値の局所性とは，命令が生成する結果には頻繁に出現する少数の値が存在することである．本論文では，頻出する値のみについて物理レジスタを共有化する手法を提案する．

本論文の構成は以下の通りである．2 章では，これまでに提案された物理レジスタの削減手法について述べる．3 章では，頻出する値のみについて物理レジスタを共有化する手法を提案する．4 章では評価を行い，5 章で本論文をまとめる．

2. 物理レジスタの削減手法

高い命令レベル並列性を少ない物理レジスタ数で達成するために，必要な物理レジスタ数を削減する研究がこれまでになされている．本章では，まずレジスタの生存期間を短縮する手法について述べ，次にこの手法を拡張して物理レジスタを共有化する手法について述べる．

2.1 レジスタの生存期間を短縮する手法

Monreal らは，物理レジスタの割り当てを命令の実行終了時まで遅らせることによりレジスタの生存期間を短縮し，必要な物理レジスタを削減する手法を提案している[4]．一般に，レジスタ・リネーミングには 2 つの目的がある．その 1 つはデータ依存関係をタグ付けすることであり，もう 1 つは論理レジスタを物理レジスタに割り当てることである．従来のレジスタ・リネーミング手法では，命令のデコード時に物理レジスタ番号を用いてデータ依存関係をタグ付けすることで，2 つの目的を同時に果たしている．この場合，物理レジスタが保持する値は命令が実行を終了するまで生成されないため，命令がデコードされてから実行を終了するまでの間，物理レジスタは記憶領域としては機能していない．これに対し，Monreal らの提案するレジスタ・リネーミング手法では，物理レジスタを命令の実行終了時に割り当てるために，命令のデコード時には仮想的に物理レジスタを想定しデータ依存関係をタグ付けすることで，2 つの目的を分離している．この手法では，命令のデコード時には論理レジスタと仮想物理レジスタの対応関係を作り，命令の実行終了時に仮想物理レジスタと物理レジスタの対応関係を作っている．物理レジスタが命令の実行終了時に割り当てられることにより，レジスタの生存する期間は短縮され，必要な物理レジスタ数を削減することができる．

2.2 物理レジスタを共有化する手法

Jourdan らは，Monreal らの手法よりさらに物理レジスタを削減する方法として，物理レジスタの共有化を行う手法をいくつか示している[5]．本論文では，これらのうち物理レジスタを最大限共有化するものとして，Monreal らのレジスタの生存期間を短縮する手法を拡張したものを評価対象とした．以下，この手法の概要を述べる．

Jourdan らは，最近の値の局所性を利用して，同じ値を生成する複数の命令間で物理レジスタを共有化する手法を示している．彼らは，時間的に近い命令同士が生成した結果値には相関があり，SPECint95 ベンチマークでは，ある命令が生成した結果値がその命令より以前の 128 個の命令が生成した結果値の中にある確率は，平均約 80% であるとしている．この局所性を最近の値の局所性と呼ぶ．

彼らの手法は，2 つの動作によって構成されている．1 つめの動作は，ある命令に物理レジスタを割り当てるときに，共有できる物理レジスタが存在するかどうかを判定する動作である．結果値と同じ値を保持する物理レジスタが存在し，その物理レジスタ番号を明らかにすることができれば，物理レジスタの共有が可能となる．この動作は，物理レジスタを連想検索することにより実現される．

2 つめの動作は，ある命令のコミット時に，その命令に対応する物理レジスタを解放してよいかどうか判定する動作である．物理レジスタの共有化を行わない場合には，物理レジスタの解放は対応する論理レジスタを再定義する命令のコミット時に無条件で行われる．しかし，物理レジスタを共有化する場合には，その物理レジスタを共有する命令が他に存在しない場合にのみ，物理レジスタは解放することができる．この動作は，物理レジスタにカウンタを追加し，その物理レジスタを共有する命令数を記録することで実現される．

彼らの手法では，これら 2 つの動作によって，最近の値の局所性を利用した物理レジスタの共有化を実現している．

3. 頻繁な値の局所性を利用した物理レジスタの共有化

従来の物理レジスタ共有化手法では，ある命令が物理レジスタを共有可能かどうかを，連想検索により判定する必要がある．連想検索が必要な理由は，値と物理レジスタ番号の対応関係が動的に変化するためである．ある命令が値を生成したときに，それと同じ値を保持する物理レジスタが存在するかどうかが不明であり，また存在するとしてもその物理レジスタ番号が不明であるため，連想検索が必要となる．仮に，値と物理レジスタ番号の対応関係を静的に固定することができれば，連想検索による判定動作の必要ない物理レジスタの共有化が実現できる．

これに対し一般に，命令が生成する結果には頻繁に出現する少数の値が存在することが知られている[6]．これを頻繁な値の局所性と呼ぶ．我々はこの性質に着目し，頻繁に出現する値のみに限定して物理レジスタを静的に割り当てることにより，少ないハードウェア量で効果的に物理レジスタを共有化できると考えた．本章では，まず頻繁な値の局所性について述べ，次にこの局所性を利用した共有化による物理レジスタの削減手法を提案する．

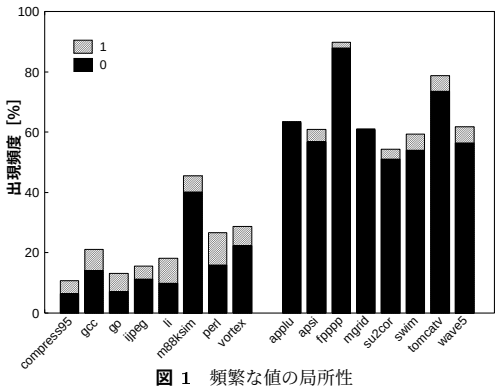


図 1 頻繁な値の局所性

3.1 頻繁な値の局所性

Zhang らは，メモリ・アクセスで参照される値には，出現頻度の高い値があるとしている．我々は，SPECint95 及び SPECfp95 ベンチマークにおいて，レジスタに書き込みを行う全ての命令について，書き込み値を調べ，頻繁な値の局所性を調査した．その結果，どのベンチマークにおいても共通して出現頻度が高い値は 0 と 1 であり，0 の出現頻度はどのベンチマークでも最も高かった．0 と 1 以外で出現頻度の高い値はベンチマークごとに異なり，共通して出現頻度の高い値はなかった．0 と 1 の出現頻度を**図 1** に示す．図の横軸はベンチマークプログラムであり，縦軸はレジスタに書き込まれる値の出現頻度である．各棒グラフは 2 つの部分からなり，上部，下部はそれぞれ 1, 0 の出現頻度である．0 の出現頻度は，SPECint95, SPECfp95 でそれぞれ平均 15.9%，63.0% であった．どのベンチマークにおいても共通して 0 と 1 の出現頻度が高いことから，0 と 1，または 0 値のみについて物理レジスタの割り当てを固定することにより，効率良く物理レジスタを共有化することが可能であると考えられる．

3.2 頻出値のみについて物理レジスタを共有化する手法

物理レジスタに書き込む値に頻出値があるという性質から，我々は，最も出現頻度の高い 0 と 1，または 0 値のみについて物理レジスタを共有化することによる物理レジスタの削減手法を提案する．この手法は，Monreal らの提案する物理レジスタを実行終了時に割り当てる機構に加え，物理レジスタの中に常にある一定の値を保持するレジスタを設け，その値が命令の結果値と一致するかどうかを判定する比較器を追加することで実現できる．

この手法では，命令の実行終了時に結果値が 0 であったら常に 0 を保持するレジスタに割り当て，1 であったら常に 1 を保持するレジスタに割り当てを行う．結果値が 0 または 1 以外であったら通常の物理レジスタに割り当てを行う．0 値のみついて物理レジスタを共有化する場合は，上記の動作を 0 のみに限定して行う．この手法では従来の共有化手法とは異なり，結果値が 0 または 1 であると判定された時点で，共有する物理レジスタ番号が明らかである．したがって，物理レジスタを共有可能であるかどうかを連想検索によって判定する必要がない．また，共有化する値を保持するレジスタは，常にその値を保持しており解放する必要がないため，従来の共有化手法で必要であった物理レジスタを解放してよいかどうかの判定動作も必要ない．

4. 評　　価

本章では，まず評価環境，評価モデル，測定条件について述べる．次に評価結果について述べる．

4.1 評 価 環 境

SimpleScalar Tool Set Version 3.0a[7] をベースに SMT プロセッサのシミュレータを作成し，物理レジスタ共有化機構を評価した．命令セットとして MIPS R10000 を拡張した SimpleScalar/PISA を使用した．ベンチマークプログラムは，SPECint95 の全 8 種類と SPECfp95 の 8 種類 (applu, apsi, fpppp, mgrid, su2cor, swim, tomcatv, wave5) を使用した．

評価は SPECint95, SPECfp95 それぞれに対し，同時実行スレッド数 1,2,4,8 の場合について行った．同時実行スレッド数が 2 以上の場合は，全てのプログラムの実行を同時に開始し，いずれかのプログラムの実行が終了するまで測定を行った．同時実行スレッド数が 2 の場合は，プログラムの組合せにより性能が変わりやすいため，SPECint95 と SPECfp95 それぞれに対してすべての組合せ (${}_8C_2 = 28$ 種類) について測定を行った．同時実行スレッド数が 4 の場合は，8 種類の組合せについて測定を行った．これは，同時実行スレッド数が 4 の場合は 1 つの組合せについての測定時間が長く，すべての組合せを測定するのは現実的に困難であるためである．しかし，同時実行スレッド数が 4 の場合は，プログラムの組合せによる性能差が小さいため問題はない．同時実行スレッド数が 8 の場合は，SPECint95, SPECfp95 それぞれ全てのプログラムを含めば良いので，組合せの問題はない．

4.2 評価モデル

以下のモデルについて評価した．

- **base モデル**
  いかなる物理レジスタの削減手法も適用しないモデルである．
- **no-share モデル**
  物理レジスタの共有化は行わず，Monreal らの提案した命令の実行終了時に物理レジスタの割り当てを行う機構を備えたモデルである．
- **zero モデル**
  命令の実行終了時に物理レジスタの割り当てを行い，我々の提案する 0 値についてのみ物理レジスタを共有化する機構を備えたモデルである．
- **binary モデル**
  命令の実行終了時に物理レジスタの割り当てを行い，我々の提案する 0 値または 1 値についてのみ物理レジスタを共有化する機構を備えたモデルである．
- **full モデル**

表 1 測定条件

命令デコード幅	最大 8 命令
命令発行幅	最大 8 命令
命令コミット幅	最大 8 命令
命令フェッチ手法	ICOUNT.2.8[3]
命令ウィンドウ	RUU(Register Update Unit) 128 エントリ LSQ(Load/Store Queue) 64 エントリ
機能ユニット	8 iALU, 4 iMULT/DIV, 4 Ld/St, 6 fpALU, 4 fpMULT/DIV/SQRT
命令キャッシュ	32KB 2 ウェイ・セット・アソシアティブ, ライン長 32 バイト, ヒット・レイテンシ 1 サイクル
データキャッシュ	32KB 2 ウェイ・セット・アソシアティブ, ライン長 64 バイト, 4 ポート, ヒット・レイテンシ 1 サイクル
2 次キャッシュ	統合, 1MB 2 ウェイ・セット・アソシアティブ, ライン長 64 バイト, ヒット・レイテンシ 12 サイクル
分岐予測機構	gshare 17 ビット履歴 256K エントリ PHT, 分岐予測ミス・ペナルティ10 サイクル

命令の実行終了時に物理レジスタの割り当てを行い，Jourdan らの提案した全ての結果値について物理レジスタを共有化する機構を備えたモデルである．

4.3 測定条件

各評価モデルにおいて共通する測定条件を**表 1** に示す．整数用と浮動小数点用の物理レジスタ数は同数とした．キャッシュと分岐予測器は，スレッドごとに用意した．これは，本論文では物理レジスタと性能の関係に注目しており，同時実行スレッド数の増加によるキャッシュのヒット率や分岐予測精度の変化が，性能に影響することを避けるためである．物理レジスタを実行終了時に割り当てる動作，及び物理レジスタを共有化する動作は，ライトバック時に 1 サイクルで行うとした．

4.4 物理レジスタの削減効果

本節では，単一スレッド実行時における各手法の物理レジスタの削減効果について評価する．評価では十分な数 (600 個) の物理レジスタを用意した．割り当てられている物理レジスタ数を各サイクルにおいてカウントし，その数の全サイクル数に対する相対累積度数を求めた．**図 2** に，測定した相対累積度数分布 (ベンチマークの平均) を示す．

図より，base モデルに対して，no-share モデル, zero モデル, binary モデル, full モデルの順に割り当てられている物理レジスタ数が少なくなっており，full モデルの物理レジスタの削減効果が最も大きいことがわかる．ここで，性能低下をほとんど起こさない必要物理レジスタ数を，相対累積度数が 80% での数とすると，base モデルでは必要物理レジスタ数は SPECint95, SPECfp95 でそれぞれ 106, 158 となった．これに対し，no-share モデルではそれぞれ 70, 92 であり，base モデルに対する物理レジスタの削減率はそれぞれ 34.0%, 41.8% となった．full モデルでは，必要物理レジスタ数はそれぞれ 33, 13 であり，base モデルに対する物理レジスタの削減率はそれぞれ 68.9%, 91.8%

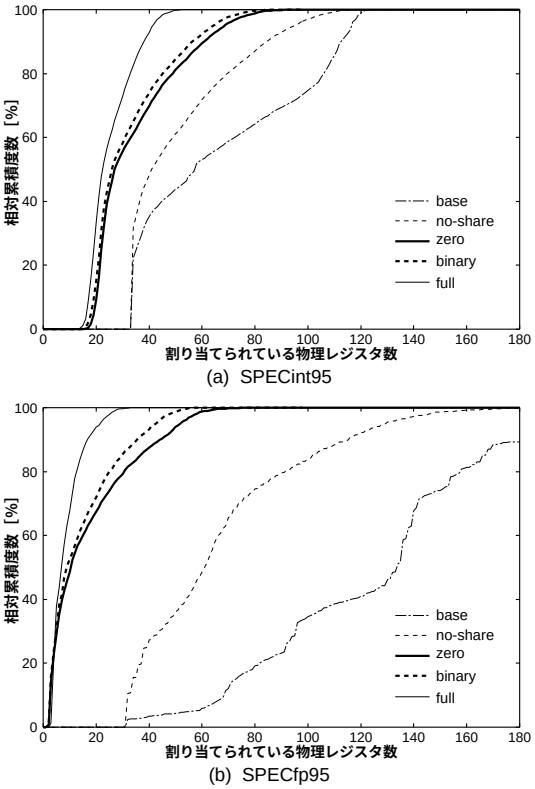


図 2 相対累積度数分布

と非常に大きいことがわかる．no-share モデルに対する物理レジスタの削減率でも，それぞれ 52.9%, 85.9% と大きい．

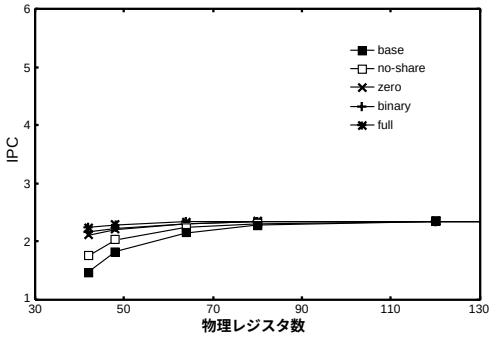
一方，binary モデルでは，必要物理レジスタ数は SPECint95, SPECfp95 でそれぞれ 45, 25 であり，base モデルに対する物理レジスタの削減率はそれぞれ 57.5%, 84.2% となった．また no-share モデルに対する物理レジスタの削減率はそれぞれ 35.7%, 72.8% となった．これらの物理レジスタの削減率は full モデルの削減率と大差なく高い値であり，binary モデルは物理レジスタの削減効果が大きく，特に SPECfp95 で削減効果が大きいことがわかる．

zero モデルについては，必要物理レジスタ数は SPECint95, SPECfp95 でそれぞれ 49, 31 であり，binary モデルとの差が小さいことから，binary モデルとほぼ同一の物理レジスタ削減効果があることがわかる．

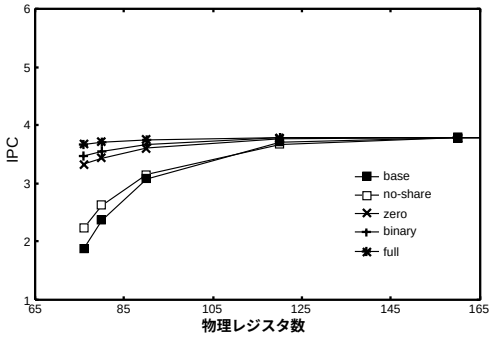
これらのことより，頻出値のみ物理レジスタを共有化する提案手法は，従来の共有化手法とは異なり連想検索を行うことなく，同等の高い物理レジスタの削減効果を得ることができると言える．

4.5 性能に対する影響

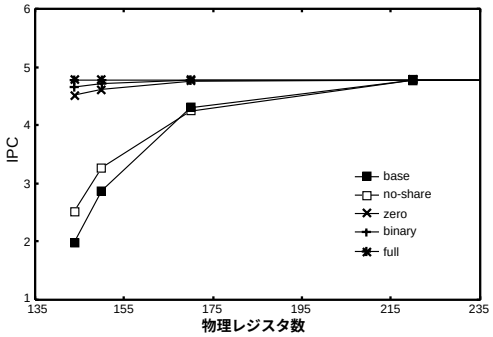
本節では，物理レジスタ数を変化させた時の性能を評価した．同時実行スレッド数が 1, 2, 4, 8 の場合の性能を測定した結果を，SPECint95 及び SPECfp95 について，それぞれ**図 3**, **図 4** に示す．同時実行スレッ



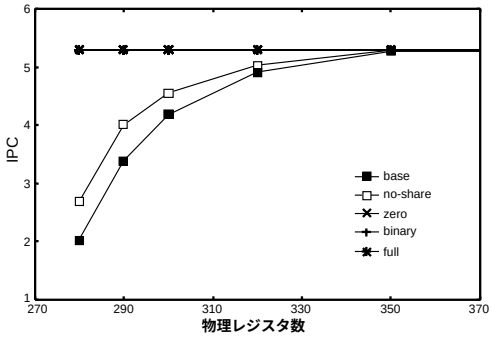
(a) 1スレッド



(b) 2スレッド

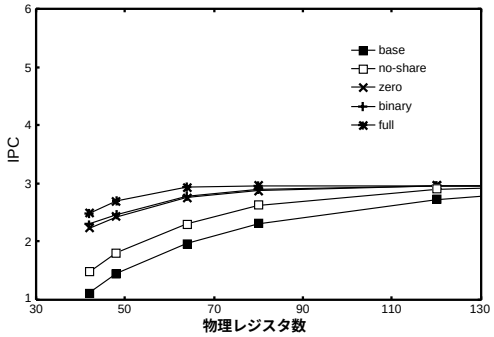


(c) 4スレッド

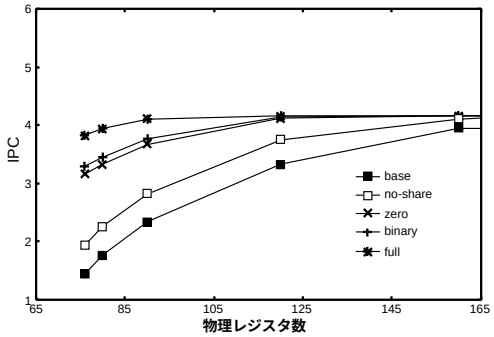


(d) 8スレッド

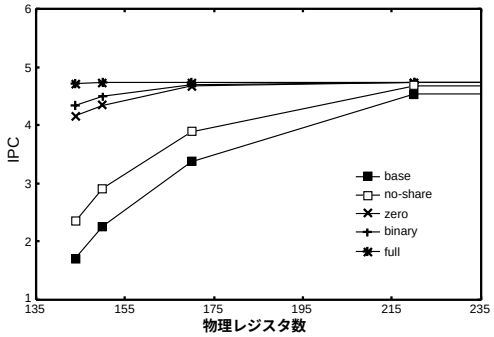
図 3 性能向上 (SPECint95)



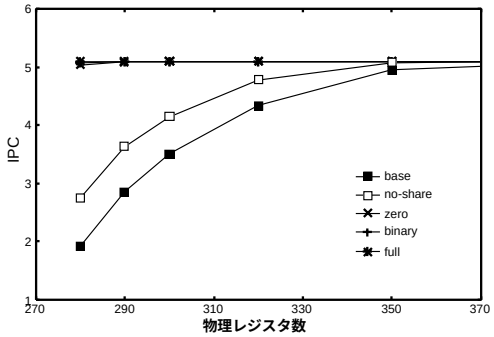
(a) 1スレッド



(b) 2スレッド



(c) 4スレッド



(d) 8スレッド

図 4 性能向上 (SPECfp95)

ド数が 1 の場合の IPC は，各ベンチマークにおける IPC の調和平均であり，同時実行スレッド数が 2, 4 の場合の IPC は，ベンチマークの各組合せにおける IPC の調和平均である．測定した物理レジスタ数の最小値は，論理レジスタ数にスレッド数を乗じたものに，リネーム用として 8 を加えた数とした．ここで，論理レジスタ数は整数用が 34 個 (0 番を除く 31 個の汎用レジスタ，2 個の乗算/除算レジスタ，1 個の浮動小数点条件コードレジスタ)，浮動少数点用が 32 個である．

図より，物理レジスタを共有しない base モデルと no-share モデルでは，SPECint95, SPECfp95 ともに，物理レジスタ数の減少に伴い IPC が大きく減少していることがわかる．ここで，最大性能を物理レジスタが十分な数ある場合の base モデルの性能とすると，base モデルでは，最大性能に対して物理レジスタ数が最小値の場合に，単一スレッド実行時には SPECint95, SPECfp95 でそれぞれ 37.0%, 62.2%，8 スレッド同時実行時にはそれぞれ 61.9%, 62.4% もの性能低下率となっている．no-share モデルでは，単一スレッド実行時にそれぞれ 25.1%, 50.0%，8 スレッド同時実行時にはそれぞれ 49.2%, 46.1% の性能低下率となっており，性能低下を若干抑制しているが，十分でないことがわかる．物理レジスタを共有化する full モデルでは，単一スレッド実行時にそれぞれ 4.7%, 15.9% の性能低下率となっており，同時実行スレッド数が 4 以上の場合にはほとんど性能が低下しないことがわかる．これより，full モデルでは，物理レジスタを共有化することにより，同時実行スレッド数が 4 以上の場合に，最小値の物理レジスタ数で最大性能を得ることができると言える．

一方，binary モデルでは，物理レジスタ数の減少に伴い性能は低下するものの，その度合は base モデルと no-share モデルより小さいことがわかる．binary モデルでは，物理レジスタ数が最小値の場合の性能が，最大性能に対して単一スレッド実行時に SPECint95, SPECfp95 でそれぞれ 7.7%, 22.3%，4 スレッド同時実行時にはそれぞれ 2.7%, 8.2% の性能低下率となっており，性能低下を大きく抑制していることがわかる．また 8 スレッド同時実行時には，物理レジスタ数が最小値の場合でもほとんど性能が低下していないことがわかる．full モデルに対する性能低下率は，最も性能差が大きい 2 スレッド同時実行時の物理レジスタ数が最小値の場合でも，SPECint95, SPECfp95 でそれぞれ 5.4%, 13.9% となっている．特に SPECint95 で性能低下が小さく，8 スレッド同時実行時には SPECint95, SPECfp95 とも full モデルとほぼ同一の性能となっている．

zero モデルについては，binary モデルに対する性能低下率が，性能差が最も大きい場合でも SPECint95, SPECfp95 でそれぞれ 4.0%, 4.1% と性能低下は小さく，zero モデルと binary モデルの性能はほぼ同じであることがわかる．

以上より，我々が提案する頻出値のみ物理レジスタを共有化する手法は，物理レジスタ数の減少に伴う性能低下を大きく抑制できることから有効性は高いと言える．また，同時実行スレッド数が多い場合には，連想検索が必要な従来の共有化手法とほぼ同一の性能を得ることができ，十分な数の物理レジスタがある場合に得られる性能を，最小値に近い物理レジスタ数で得られることから，その有効性が特に高いと言える．

5. ま と め

本論文では，連想検索など複雑度の増加が避けられない従来の共有化手法に対して，複雑度を増加させることなく共有化を実現する方法として，頻出値のみ物理レジスタを共有化する手法を提案した．

評価の結果，物理レジスタ数が最小値に近い場合，物理レジスタを共有化しない手法では物理レジスタが十分な数ある場合に対して，性能が非常に大きく低下する．これに対し提案手法では，物理レジスタ数が最小値に近い場合でも，物理レジスタが十分な数ある場合に対する性能低下率は単一スレッド実行時に SPECint95, SPECfp95 でそれぞれ 7.7%, 22.3%，4 スレッド同時実行時にはそれぞれ 2.7%, 8.2%，8 スレッド同時実行時には 1% 以下であり，性能低下を大きく抑制できることがわかった．また，連想検索など複雑度の増加が避けられない従来の共有化手法に対する性能低下はほとんどなく，頻出値のみ物理レジスタを共有化する提案手法の有効性は高いことがわかった．特に 8 スレッドを同時実行する場合には，従来の共有化手法とほぼ同一の性能が得られ，最小値に近い物理レジスタ数で，十分な数の物理レジスタがある場合とほぼ同じ性能が得られることがわかった．

参 考 文 献

1) D. M. Tullsen, et al., “Simultaneous Multithreading: Maximizing On-Chip Parallelism,” In *Proc. ISCA-22*, pp.392-403, Jun. 1995.
2) K. I. Farkas, et al., “Register File Design Considerations in Dynamically Scheduled Processors,” In *Proc. HPCA-2*, pp.40-51, Jan. 1996.
3) D. M. Tullsen, et al., “Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor,” In *Proc. ISCA-23*, pp.191-202, May 1996.
4) T. Monreal, et al., “Delaying Physical Register Allocation Through Virtual-Physical Registers,” In *Proc. MICRO-32*, pp.186-192, Nov. 1999.
5) S. Jourdan, et al., “A Novel Renaming Scheme to Exploit Value Temporal Locality Through Physical Register Reuse and Unification,” In *Proc. MICRO-31*, pp.216-225, Nov. 1998.
6) Y. Zhang, et al., “Frequent Value Locality and Value-Centric Data Cache Design,” In *Proc. ASPLOS-IX*, pp.150-159, Nov. 2000.
7) D. Burger, et al., “The SimpleScalar Tool Set Version 2.0,” Technical Report 1342, Computer Sciences Department, University of Wisconsin, Jun. 1997.