1H-7

縮小レジスタを用いる先行実行プロセッサ

笹河 良介 † 吉瀬 謙二 ‡
東京工業大学 工学部情報工学科 † 東京工業大学大学院 情報理工学研究科 ‡

1 はじめに

逐次プログラムの高速化を図るために, 先行実行コアを利用したプロセッサの研究が行われている. 先行実行コアを実現するために, 投機的な実行を利用した方法がある.

本稿では縮小レジスタを用いた先行実行コアを提案する. また, レジスタの振る舞いと実際に縮小した時のプロセッサの動作を検証する.

2 先行実行コア

先行実行コアを利用したプロセッサの 1 つに Slipstream processors[2] がある.Slipstream processors の先行実行コアは十分な分岐予測が可能であると判断すると, それらの分岐命令および分岐命令のみに関係する命令を省いた命令列を実行する. これにより先行実行を可能にし, 先行実行で計算したデータフローや制御フローを値予測および分岐予測として, 冗長コアに渡すようになっている. 先行実行コアから来た制御フローが冗長コアの実際のフローと一致しなかったときは, 冗長コアからレジスタファイルやプログラムカウンタの値を先行実行コアへコピーしてリカバリを行う.

先行実行コアを利用する他の例としては, 長いストール時に後続の命令を投機的な値を用いて 1 度先行実行する run-ahead execution を先行実行コアに利用した Dual-Core Execution[3] がある.

冗長コアを使わず, マルチスレッディングを使ってデータのプリフェッチを行う Helper Threading[1] や, 先行実行によるロード命令のレイテンシ削減 [4] を行うものもあり, 先行実行のデータフローのみを使用して高速化を目指す研究もある.

3 縮小レジスタファイル

3.1 先行実行コアの提案

2 節で示したように, 先行実行コアの中には投機的な値を用いて先行実行を行うものがある. 正確な値を用いない先行実行コアでも, 制御フローが正しければその制御フローを分岐予測として利用できる. そこでレジスタファイルの bit 数を縮小した先行実行コアを提案する. このコアはデータパスの bit 幅減少により動作周波数を向上させることで先行実行を行う. bit を削減したことで不正確な値を生成・利用することになるので, 制御フローのみを利用することに焦点を置く.

A Pre-execution Processor with Shortened Regiters
Ryosuke SASAKAWA, and Kenji KISE
† Department of Computer Science, Tokyo Institute of Technology
‡ Graduate School of Information Science and Engineering, Tokyo Institute of Technology

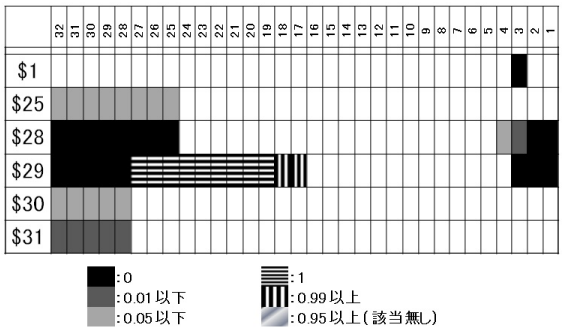

図 1: 各レジスタの各 bit における期待値の偏り

3.2 レジスタファイルの取りうる値

縮小レジスタファイルを用いた先行実行コアの検討にあたり, まずレジスタファイルがどのような値を取っているのか調査した. 今回は MIPS システムシミュレータ SimMips を用いて, レジスタの値が変更されたときに一体どのような値を取り得るのか, 各レジスタの bit ごとの期待値を取った. 使用したプログラムは SPEC CINT2006 より bzip2, gcc, gobmk, hmmer, libquantum, h264ref, omnetpp と, dhrystone ベンチマーク,FFT(単精度・倍精度), クイックソートである.

すべてのプログラムにおいて偏った期待値を示した bit を図 1 にまとめた. 図では期待値が 0.05 以下のものと 0.95 以上のものを表示してある. 図 1 に無いレジスタは特に偏りなく使用されている. スタックポインタレジスタである$29 については今回使用したアーキテクチャのメモリが 128MiB であったことから, 上位 bit で図にあるような偏りが生まれたと考えられる.

4 縮小レジスタを持つプロセッサの動作検証

4.1 疑似的な縮小レジスタファイルの構成

縮小レジスタファイルを用いた時にどの程度制御フローが一致しうるのか調べるため, シミュレーションを行う. レジスタから情報を **n**bit 分縮小する方法に以下の 3 つを用いた.

- set-0 - 最上位 bit の 32bit 目から **n**bit 分の bit を 0 にセットする
- signed-(32-n)bit - 32bit 目から **n**bit 分の bit を, それらの 1 つ下位の bit である (32-**n**)bit 目と同じ値にセットする.
- signed-32bit - 31bit 目から **n**bit 分の bit を, 最上位 bit である 32bit 目と同じ値にセットする.

以上の bit の縮小方法は, 実際に冗長コアから先行実行コアへレジスタファイルの値をコピーする際にどう bit

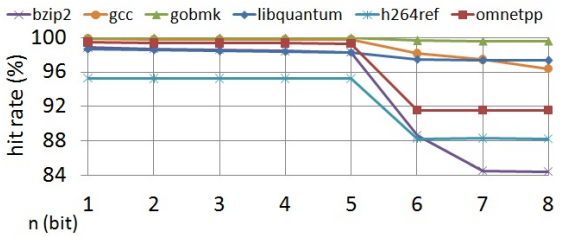

図 2: set-0 の分岐予測ヒット率

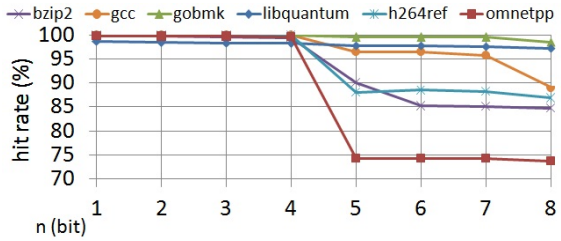

図 3: signed-(32-n)bit の分岐予測ヒット率

を削減するか, さらに先行実行コアで計算された値を冗長コアへ受け渡すときにどう 32bit に復元するべきかを決定する際のヒントになる.

4.2 実際の動作構成及び評価

今回の検証では bit を縮小しないプロセッサも同時に動作させ, それと比較することで縮小レジスタのプロセッサの制御フローを測定する. 2 つのプロセッサのプログラムカウンタが異なったときは, 命令をフェッチする前に正しいプロセッサからレジスタファイルを含むアーキテクチャステートとメモリの内容をコピーする. またレジスタを縮小したプロセッサの syscall 命令については nop 命令に置き換える. プログラムカウンタが異なる原因となるのはレジスタファイルから取ってきた値を条件式及び分岐先アドレスに使う分岐命令及びジャンプ命令である. 今回の検証は上記のような命令のうち, 正しく分岐した割合を分岐予測ヒット率として評価する. また bit の削減数 **n** は 1bit から 8bit までとする.

図 2,3,4 が bit を削減したレジスタファイルを用いたプロセッサの分岐予測ヒット率である. 縦軸がヒット率, 横軸が削減した bit 数となっている. どのプログラムでも set-0 では 5bit,signed-(32-**n**)bit 及び signed-32bit では 4bit 削減するまで高いヒット率を維持している. 特に set-0 及び signed-32bit に関しては,8bit 削減してもヒット率は 84%を超えている.

set-0 と signed-32bit では 27bit 目を削ってから大きくヒット率が下がっている. また signed-(32-**n**)bit では 0,1 をセットする値を決定する bit が 27bit 目になってから, ヒット率が悪化している. 以上から 28bit 目と 27bit 目の間に大きい壁があることが分かる.

5 考察

3 節で, レジスタファイルの各 bit における期待値の偏りの調査によって多くのレジスタは偏りなく使われ, スタックポインタとなる$29 はメモリサイズに依存する

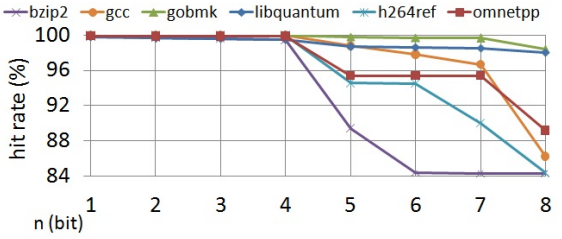

図 4: signed-32bit の分岐予測ヒット率

が, 入る値は偏りがあることが分かった. さらにスタックポインタが load/store 命令と関係していることから, 先行実行によるデータのプリフェッチへの可能性も検討できる. もし先行実行をデータのプリフェッチのみで活用するのであるのならば, 制御フローは大まかに一致していればよいので, 先行実行コアのリカバリによるペナルティを少なくすることも可能である.

4 節では, 有効な bit 数を削減することによって擬似的に縮小レジスタを構築し制御フローの一致具合を確かめ,bit の削減方法によっては分岐予測ヒット率が 84%以上となることが分かった. また,28bit 目と 27bit 目の間で大きくヒット率に差が出ることが分かった. ひとつの可能性としては, 図 1 におけるスタックポインタでの 28bit 目と 27bit 目との bit 期待値の差である. もしスタックポインタが関係する命令によってヒット率が下がったとするならば, これらに対してサポートを加えれば, ヒット率を改善し, より多くの bit を削減できると考えられる.

また 4 節でのシミュレーションは syscall 命令を nop 命令に置き換えたが,syscall 命令を省き, 浮動小数点演算に関する命令やユニットを削減することで, さらに先行させることが可能である. しかしその場合は実行命令数減少によるヒット率の低下を十分に考慮しなければいけない.

6 おわりに

本稿では, 先行実行コアを有したプロセッサとして, レジスタファイルの bit 数を縮小させたレジスタを用いた先行実行コアを提案し, 疑似的にその可能性を検証した. 今後は, より多くの bit の削減は可能なのか, 実際にどう先行させるのかといった構成やアーキテクチャを明確に決定し, さらに検討を行っていく.

参考文献

[1] Yonghong Song, et al. Design and Implementation of a Compiler Framework for Helper Threading on Multi-core Processors. In *PACT 2005*, 2005.

[2] K. Sundaramoorthy, et al. Slipstream Processors: Improving both Performance and Fault Tolerance. In *ASPLOS-9*, 2000.

[3] H. Zhou. Dual-core execution: Building a highly scalable single-thread instruction window. In *PACT 2005*, 2005.

[4] 山本哲弘ほか. 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 *SACSIS2006*, 2006.