卒業論文

題目

書込予測型バンクレジスタファイルの性能評価と予測機構の改良

指導教員

佐々木 敬泰 助教

2017年

三重大学 工学部 情報工学科
コンピュータアーキテクチャ研究室

北畠 奨太(413813)

内容梗概

近年，性能の向上のために，複数の命令を同時に実行することができるアウトオブオーダー型のスーパースカラプロセッサが主流になってきている．それらのプロセッサは同時に複数のデータを扱うために，多ポートのレジスタファイルが必要不可欠となっている．一般にレジスタファイルを構成しているSRAMの回路面積はポート数の2乗に比例するため，多ポートのレジスタファイルは非常に大きな回路面積となる．FPGAや安価なASICの設計フロー上では，様々な制約によって多ポートのレジスタファイルを実装することは困難である．そこで，SRAMの多重化を用いて多ポートのレジスタファイルを作成する手法が多く用いられている．しかし，SRAMの多重化手法は多ポートになるほど大量のSRAMを必要とするため，レジスタファイルのような多ポートのメモリは多重化によるオーバーヘッドが非常に大きい．これらの問題に対し，バンクメモリを用いた小面積のレジスタファイルが提案されている．バンクメモリとは，バンクと呼ばれる複数の領域にデータを分割し，管理するメモリのことである．これにより，各SRAMのポート数を減らすことができ，回路面積の削減に繋がる．しかし，1つのバンクに複数のアクセスが集中した場合，競合が発生し，メモリへのアクセスを1つずつ処理しなければならない．この競合をバンクコンフリクトという．レジスタファイルにバンクメモリを適用した場合，バンクコンフリクトが発生するといずれかのデータが書き込めず，性能が低下するおそれがある．先行研究として，小面積かつ予測機構を用いることにより性能の低下を抑えた書込予測型バンクレジスタファイルが挙げられる．先行研究では，書込予測機構を追加することによって，バンクコンフリクト発生の低減を目指している．しかし，先行研究の性能評価では，一部のベンチマークにおいて，実行時間が伸びてしまっている．これは先行研究の予測機構ではバンクコンフリクトが発生する命令列のパターンがあるためと考えられる．また，先行研究は，低IPC(Instruction per Cycle)環境での評価しか取られておらず，バンクコンフリクトが発生しやすい高IPC環境での評価が不十分といえる．そこで本研究では，書込予測型バンクレジスタファイルの性能の再評価と予測機構の改良を行う．書込予測型バンクレジスタファイルの性能の再評価では，スーパースカラプロセッサのシミュレータであるSimpleScalarに先行研究の動作を導入し，IPCの高い環境を構築し

た上で，評価を行う．評価を行った結果，マルチポートレジスタファイルより実行時間が伸びていた．これは先行研究の予測機構では，バンクの数と同等の命令が発行された時，依存関係により最後の命令が遅れなければバンクコンフリクトが発生するためだと考えられる．そこで，その原因を踏まえ，予測機構の改良を行った．予測機構の改良では，依存関係により実行のタイミングが遅れる命令を優先的にバンクに割り当てる．それにより，依存関係があった命令が割り当てられたバンクメモリにアクセスする命令の実行タイミングは前回アクセスした命令と同じように遅らせることができる．そのように各命令のバンクメモリへの割り当てを調整することにより，バンクコンフリクトの回避を狙う．評価結果では，通常のバンクレジスタファイルに比べ，実行時間を平均 0.85%，最大 2.67%削減することに成功した．

Abstract

In recent years, out-of-order superscalar processors which can simultaneously execute multiple instructions are used to improve performance of processors. Because these processors handle multiple data at the same time, multiport register files are indispensable. In general, the scale of SRAM circuit is proportional to the square of the number of ports. Therefore, the circuit scale of the multiport register file is very large. In FPGA or reasonable ASIC design flow, it is difficult to implement multiport register file due to various constraints. Therefore, a method of creating a multiport register file by using SRAM multiplexing is often used. However, in the SRAM multiplexing method, as the number of ports increases, a huge amount of SRAM is required. Therefore, the overhead due to multiplexing is very large in multiport memory such as register files. To solve these problems, a small scale register file using a bank memory has been proposed. The bank memory divides data across multiple areas called bank. Therefore, a bank memory can reduce circuit scale. However, if the multiple accesses concentrate on one bank, conflicts will occur. Therefore, the access to memory must be handled one by one. This conflict is called bank conflict. If bank conflict occurs in the banked register file, plural register values can not be written simultaneously and the performance is degraded. As a previous study, register port prediction for a banked register file can be proposed. They aim to reduce the occurrence of bank conflicts by prediction mechanism. However, in the performance evaluation of the previous study, the execution time has increased in some benchmarks. This is conceivable that there is a pattern of instruction sequence in which bank conflict occurs in the prediction mechanism of previous study. In addition, the evaluation in the previous study is done only in the low IPC (Instruction per Cycle) environment. However, the more IPC becomes higher, the more bank conflict occurs. Therefore, the evaluation in the high IPC environment is indispensable. In this study, reevaluation of the performance of the register port prediction for a banked register file and improve the prediction mechanism were carried out. In the reevaluation of the performance, this paper im-

plements the previous method on the SimpleScalar and evaluates it under the high IPC condition. As a result of reevaluation, execution time becomes longer than that of multiport register file. In the previous method, when instructions whose destination registers are assigned to the same bank are issued, it is considered that the bank conflict occurs, if the last instruction is not delayed by the dependency relationship. Therefore, to solve the problem, improvementation of the prediction mechanism were carried out. In the improvementation of the prediction mechanism, the dependency instruction is preferentially allocated to the bank. In that way, proposed prediction aims to avoid bank conflicts by adjusting the allocation of each instruction to the bank memory. As a result, compared with previous banked register file, proposed mechanism can reduce execution cycles by 0.85% in average and 2.67% in maximum.

目 次

1 はじめに **1**
1.1 研究背景 1
1.2 研究目的 2

2 スーパースカラプロセッサ **4**
2.1 パイプライン 4
2.2 スーパースカラ 7
2.3 アウトオブオーダー実行 8

3 レジスタファイルとバンクレジスタファイル **10**
3.1 SRAM の多重化 10
3.1.1 リードポートの多重化 11
3.1.2 ライトポートの多重化 12
3.1.3 リード・ライトポートの多重化 12
3.2 バンクレジスタファイル 15
3.3 バンクコンフリクト 15
3.4 バンクの割り当て手法 16

4 先行研究 **18**
4.1 概要 18
4.2 書込予測機構 19
4.3 問題点 23

5 書き込み予測機構の改良 **26**
5.1 概要 26
5.2 書き込み予測機構の詳細 26
5.3 提案手法実装時の動作 29

6 書込予測型バンクレジスタファイルの再評価 **31**

7 評価 **33**
7.1 評価環境 33
7.2 性能評価 34
7.3 考察 35

8 おわりに **37**

謝辞 **38**

参考文献 **39**

A　プログラムリスト **41**

B　評価用データ **41**

図 目 次

2.1 一般的なパイプライン構造 4
2.2 偽の依存関係 8
2.3 真の依存関係 8
3.4 リードポートの多重化 12
3.5 ライトポートの多重化 13
3.6 リードポート・ライトポートの多重化 14
3.7 バンクレジスタファイル 16
3.8 バンクコンフリクトが発生する様子 18
4.9 書込予測機構を用いたバンク割り当て 22
4.10 書込予測機構の問題点 24
5.11 提案手法の書込予測機構の動作 30
7.12 評価結果 35

表 目 次

7.1 評価を行った手法 33
7.2 プロセッサの構成 34

1　はじめに

1.1　研究背景

近年，プロセッサの高性能化のために命令を同時実行することができるスーパースカラプロセッサが主流になっている．このようなプロセッサでは，命令を同時に実行するために，複数のデータを扱える多ポートのレジスタファイルが必要となる．また，プロセッサの高性能化のためには，命令の同時実行数はさらに増えるため，レジスタファイルに必要なポート数も多くなる．レジスタファイルは SRAM で構成されており，1 命令あたり，2 つのリードポートと 1 つのライトポートを必要とする．4 つの命令を同時実行することができるスーパースカラプロセッサを考えたとき，それを構成するレジスタファイルのポート数の合計数は 12 となる．SRAM の回路面積は，ポート数の 2 乗に比例するため，レジスタファイルの回路面積は非常に大きいものとなる．これは，製造コストや消費電力，アクセス時間の増大といった問題を引き起こす．また，FPGA や安価な ASIC の設計フローでは，多ポートの SRAM を実装することは困難である．多ポートの SRAM の設計方法は論理合成や手設計といった方法があるが，設計に時間がかかるなどの問題がある．そこで，SRAM の多重化により多ポートメモリを作成する手法 [4] を用いて，多ポートのレ

ジスタファイルを設計する．しかし，SRAM の多重化手法では，多重化に必要な SRAM の個数はリードポート数とライトポート数の積に比例するため，多重化による面積の増大が問題となっている．

1.2 研究目的

SRAM の多重化による面積の増大を低減する方法の 1 つとして，レジスタのデータを複数の領域に分割し管理する，バンクメモリと呼ばれる手法がある．しかし，バンクメモリには第 3.4 章で述べるようなバンクコンフリクトが発生する可能性があり，性能低下の原因となっている．そこで，先行研究として，バンクへの書き込みを予測することでコンフリクトを回避する手法 [1] が提案されている．予測機構を用いることでバンクコンフリクトの回数を低減し，実行時間の削減に成功している．

しかし，先行研究の性能評価は低 IPC の環境で行われている．バンクコンフリクトは高 IPC の場合に発生しやすいと考えられるため、高 IPC の環境での効果の確認が必要である．また，一部のベンチマークでは性能が悪化している．これは先行研究の予測機構では予測できないような命令列があると考えられる．そこで，本研究では，プロセッサシミュレータである SimpleScalar [3] を用いて，高 IPC 環境で書き込み予測型バンク

レジスタファイルの性能評価を行うとともに，依存関係の情報を用いてバンクコンフリクトを回避する予測機構の改良を行う．

以降，本稿は次のように構成する．次章ではスーパースカラプロセッサの概要について，第 3 章では SRAM の多重化とバンクレジスタファイルについて，第 4 章では先行研究の概要とその問題点について，第 5 章では提案する予測機構の改良方法について，第 6 章では SimpleScalar の概要と再評価のための変更点について，第 7 章では評価を行う．

2　スーパースカラプロセッサ

本章では，本研究で想定している物理レジスタを用いるアウトオブオーダー型のスーパースカラプロセッサの概要について述べる．

2.1　パイプライン

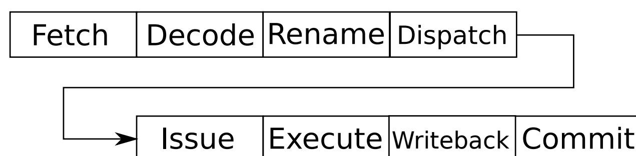

図 2.1: 一般的なパイプライン構造

図 2.1 に一般的なパイプライン構造を示す．高性能なプロセッサでは，命令の読み出しから，命令の解析・実行，結果の書き込みまでを 1 サイクルで行うようなことはしていない．一定の処理ずつに区切り，並列に実行させている．一定の処理を並列に実行することにより，1 サイクル中の処理量を減らすことができ，動作周波数を向上させることができる．そうすることにより，処理全体のスループットを向上させることができ，性能の向上にも繋がる．これをパイプライン処理という．以下に各パイプラインステージの動作を説明する．

Fetch

図 2.1 に示す Fetch ステージでは，プログラムカウンターの値に従って，命令キャッシュにアクセスし，命令を読み出す．その後，プログラムカウンターの値を更新する．

Decode

図 2.1 に示す Decode ステージでは，Fetch ステージで読み出した命令を解析し，命令に関する情報を得る．

Rename

物理レジスタ方式では，図 2.1 に示す Rename ステージで，命令セットで使用されている論理レジスタを物理レジスタに変換する．変換には，論理レジスタがどの物理レジスタを使っているか対応付けする Rename Map Table(RMT)，使われていない物理レジスタを管理する Free List を用いる．まず，RMT を参照し，Decode ステージでデコードされた命令のソースレジスタ番号が物理レジスタ番号に変換される．命令のデスティネーションレジスタ番号は Free List を参照し，得られた物理レジスタ番号を RMT に登録する．この一連の動作をレジスタリネーミングといい，レジスタリネーミングによっ

て命令間の偽の依存関係が取り除かれる．

Dispatch

図 2.1 に示す Dispatch ステージでは，Rename ステージでリネームされたデスティネーションレジスタ番号とそれが割り当てられた物理レジスタ番号を Active List に登録する．Active List に登録された情報は，例外が発生したとき，Rename Map Table の状態を復元するために活用される情報である．また，全ての命令は Issue Queue に書き込まれる．

Issue

図 2.1 に示す Issue ステージでは，Issue Queue にある命令のうち，ソースオペランドが揃い，実行可能になった命令からアウトオブオーダーに発行される．

Excute

図 2.1 に示す Excute ステージでは，演算器を用いて命令の実行を行う．

Writeback

図 2.1 に示す Writeback ステージでは，演算結果をデスティネーショ

ンレジスタ番号が割り当てられた物理レジスタに書き込み，Active List に命令が完了したことを伝える．

Commit

図 2.1 に示す Commit ステージでは，コミットされる命令を Active List から読み出し，その命令のデスティネーションレジスタ番号に割り当てられた物理レジスタ番号に対応する Architecture Map Table のエントリを更新する．使われなくなった物理レジスタを解放し，Free List に戻す.

2.2　スーパースカラ

スーパースカラとは，命令を処理できる回路を複数持たせることにより，複数の命令を同時に実行できるようにし，短時間で多くの命令を実行できるようにすることである．1 度に 1 つの命令しか処理できない非スーパースカラ型の場合，複数の処理を実行するのに，時間がかかってしまうため，近年のプロセッサはスーパースカラ型にする場合が多い．複数の命令を同時に処理するためには，複数のデータを同時に操作できなければならない．そのため，高性能なプロセッサには複数のデータに同時にアクセスすることができる多ポートのレジスタファイルが必要不可欠

となっている．

2.3 アウトオブオーダー実行

```
add R1, R2, R3
sub R2, R4, R5
```

Write After Read

```
add R1, R2, R3
sub R1, R4, R5
```

Write After Write

図 2.2: 偽の依存関係

```
add R1, R2, R3
sub R4, R1, R5
```

Read After Write

図 2.3: 真の依存関係

アウトオブオーダー実行とは，命令の発行順と実行順が異なることを許し，実行可能となった命令から順に命令を実行していくことで命令の待ち時間を削減する手法である．しかし，命令の依存関係により命令の実行順を入れ替えてはいけない命令列が存在する．図 2.2 と図 2.3 に依存関係がある命令列の例を示す．図 2.2 左の例では，2 番レジスタの値を入力とした加算命令の後に 2 番レジスタに値を代入する減算命令が行われている．ソースオペランドとして読み込まれた後にデスティネーションレジスタとして同じレジスタが使われる関係を Write After Read(WAR) という．同様に，図 2.2 右の例では，1 番レジスタに値を代入する加算命令の後に 1 番レジスタに値を代入する減算命令が行われている．デスティネーションレジスタとして使われた後にデスティネーションレジスタとして同じレジスタが使われる関係を Write After Write(WAW) という．こ

れら 2 つの例では，減算命令の結果の格納場所を変え，これより後の命令では変更後の格納場所の値を使うことにより，命令の実行順を入れ替えても命令通りの動きを行うことができる．このような依存関係を偽の依存関係といい，レジスタリネーミングを行うことで取り除くことができる．

図 2.3 の例では，1 番レジスタに値を代入する加算命令の後に 1 番レジスタの値を入力とした減算命令が行われている．デスティネーションレジスタとして使われた後にソースオペランドとして同じレジスタが読み込まれる関係を Read After Write(RAW) という．RAW のような依存関係を真の依存関係といい，レジスタリネーミングを行っても依存関係を取り除くことはできず，命令の発行順通りに実行しなければならない．

3 レジスタファイルとバンクレジスタファイル

本章では，FPGA や ASIC の設計フロー上で多ポートのレジスタファイルを設計するときに用いられている SRAM の多重化について述べる．その後，SRAM を多重化する際の問題点とその問題の解決法の一つであるバンクメモリについて述べる．

3.1 SRAM の多重化

通常，多ポートのメモリの設計には以下に示すような方法がある．

論理合成

Register Transfer Level(RTL) で多ポートメモリを設計し，ゲートレベル (AND ゲートや OR ゲートなど) のハードウェア要素を用いて設計を行う．

メモリコンパイラ

メモリを自動設計するツールを用いて，メモリの設計を行う．

手設計

全て手動で設計を行う．

SRAM の多重化

1R1W の SRAM の多重化を行い多ポートのメモリの設計を行う．

しかし，FPGA や ASIC ではそれぞれの手法には問題がある．論理合成によって設計された多ポートメモリは，回路規模が非常に大きくなる．メモリコンパイラでは，多ポートメモリには対応していない場合がある．手動で設計を行うと非常に時間がかかってしまう．比較的コストが低く，短時間で設計することができる SRAM の多重化が用いられている．

3.1.1 リードポートの多重化

図 3.4 に 2 リード・1 ライトポートの SRAM のブロック図を示す．リードポートの多重化は，図 3.4 で示す wr_data0 のような入力信号をそれぞれの SRAM のライトポートに繋ぐことで，同一のデータをそれぞれの SRAM が保持することになる．リードポートは SRAM によって別の場所を指すようにアドレスを入力することにより，図 3.4 で示す rd_data0 や rd_data1 の出力データは異なるアドレスから同時に呼び出すことが可能となる．

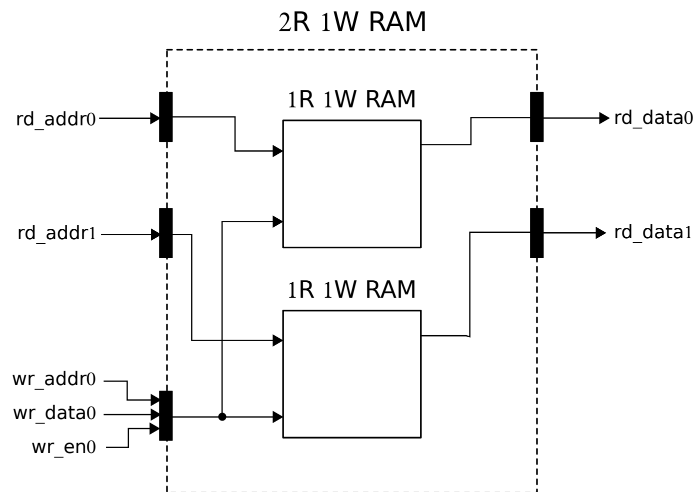

図 3.4: リードポートの多重化

3.1.2 ライトポートの多重化

図 3.5 に 1 リード・2 ライトポートの SRAM のブロック図を示す．リードポートの多重化と同様に複数の SRAM を並べて実現する．ライトポートの多重化では，図 3.5 で示す ram_select_vector のような各エントリの最新の情報を持つ SRAM を記憶する MRU メモリを必要とし，最新の情報を持つ SRAM からデータを引き出すためのセレクタも必要となるため，リードポートの多重化に比べ，必要となるハードウェアが多い．

3.1.3 リード・ライトポートの多重化

図 3.6 に 2 リード・2 ライトポートの SRAM のブロック図を示す．この時必要となる SRAM の個数はリードポートとライトポートの個数に比例

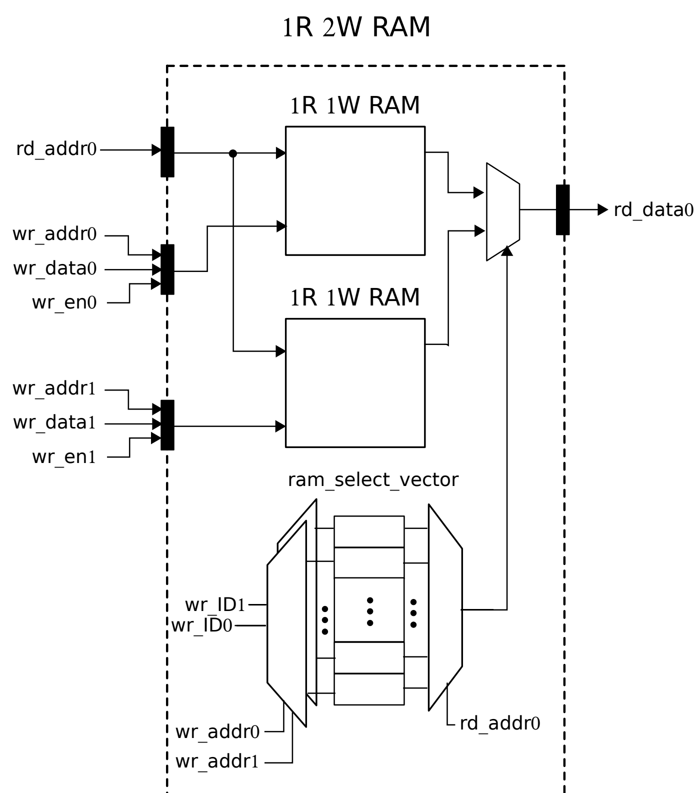

図 3.5: ライトポートの多重化

するため，図 3.6 が示すように，必要な SRAM の個数が多くなり，ハードウェア規模が大きくなる．また，第 3.1.2 章で述べたようにライトポートの多重化に必要なハードウェアは多い．そのため、多重化を用いてレジスタを多ポート化しようとするとハードウェア規模が非常に大きくなってしまう．

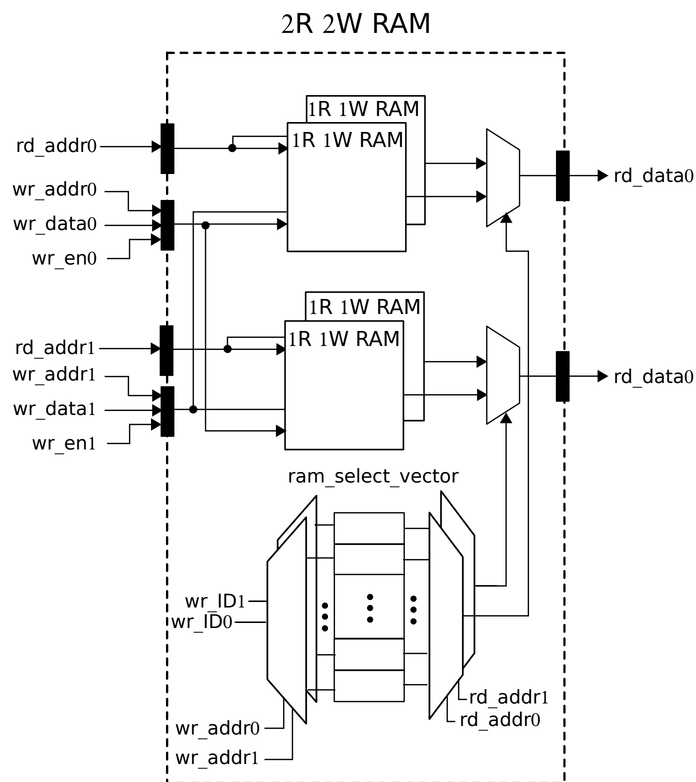

図 3.6: リードポート・ライトポートの多重化

3.2　バンクレジスタファイル

第 3.1 章で述べたように SRAM の多重化ではライトポートの多重化に大きなコストがかかる．そこで，バンクレジスタファイルを用いることにより，ライトポートの多重化に発生するコストを削減する．バンクレジスタファイルとは，バンクと呼ばれる複数の領域にデータを分割し，管理するレジスタファイルのことである．図 3.7 左に 8 エントリのレジスタファイル，図 3.7 右に 4 つの領域に分割した 4 バンクのレジスタファイルを示す．なお，各レジスタファイルは 8R4W，バンクは 8R1W とする．図 3.7 左は 8 エントリの SRAM を多重化しなければならないが，図 3.7 右は各バンクで，2 エントリの SRAM を多重化することで済むこととなり，結果的にレジスタファイルの回路面積を大幅に削減することができる．このようにマルチバンク化を行うことにより，同じ容量の SRAM をより小さいハードウェア規模で設計することができる．しかし，マルチバンク化することで，バンクコンフリクトという問題が発生する．

3.3　バンクコンフリクト

先述した図 3.7 の各レジスタファイルに複数のデータを同時に書き込むことを考える．このとき，図 3.7 左は 0 番と 4 番へ同時に書き込むこと

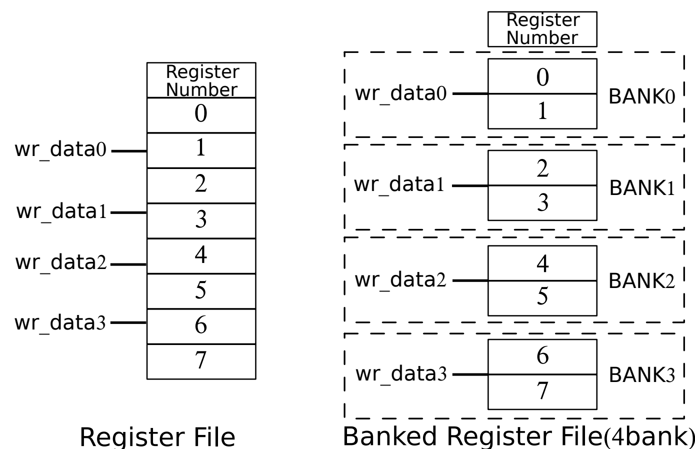

図 3.7: バンクレジスタファイル

が可能であるが，図 3.7 右は 1 つのバンクにはライトポートを 1 つしか持たないために，0 番と 4 番へ同時に書き込むことはできない．このようにマルチバンク化することにより，バンクへのアクセスに競合が発生してしまうことをバンクコンフリクトという．バンクコンフリクトが発生すると，どちらかの書き込みを遅延させなければならず，プロセッサの性能低下につながる．

3.4 バンクの割り当て手法

バンクコンフリクトを避けるための手段の 1 つとして，同時に発行される命令の書き込み先をそれぞれ違うバンクに割り当てる方法がある．しかし，実際には命令によって実行サイクル数が異なっていたり，第 2.3 章で述べたような真の依存関係により先行命令の完了を待たなければなら

ず，実行タイミングが遅れたりするため，バンクコンフリクトを大幅に削減することはできない．図 3.8 にバンクコンフリクトが起きる場合の例を示す．この例では，命令の実行サイクル数は全て 1 サイクルとし，同時に 4 つの命令を実行することができるプロセッサを考える．まず，プロセッサは 0 番から 3 番の命令を順番に異なるバンクに割り当てる．同様に，次のサイクルで 4 番から 7 番の命令を異なるバンクに割り当てる．このとき，1 番の命令には 0 番との依存関係があり，2 番の命令には 1 番の命令との依存関係が存在する．そのため，1 番と 2 番の命令は図 3.8 中央のようにパイプラインステージの実行ステージが遅れることがわかる．同様に，6 番と 7 番の命令も依存関係により実行ステージが遅れていることがわかる．このとき，1 番と 5 番の命令や 2 番と 6 番の命令に注目すると，同じサイクルで同じバンクに書き込もうとしていることがわかる．このように，命令の発行時に各命令の書き込み先を異なるバンクに割り当てる方法ではバンクコンフリクトが発生してしまう可能性がある．

instructions		pipeline stage	bank number
0:lui	gp, 0x4b	RN ------ EX WB	0
1:addiu	gp, gp, 30336	RN ------ EX WB	1
2:lw	a0, -32744(gp)	RN ------ EX WB	2
3:lw	a1, 0(sp)	RN ------ EX WB	3
4:addiu	a2, sp, 4	RN ------ EX WB	0
5:li	at, -8	RN ------ EX WB	1
6:and	sp, sp, at	RN ------ EX WB	2
7:addiu	sp, sp, -32	RN ------ EX WB	3

RN : Rename stage
EX : Execution stage
WB : Writeback stage

図 3.8: バンクコンフリクトが発生する様子

4 先行研究

本章では，先行研究である書込予測型バンクレジスタファイルの詳細について述べる．

4.1 概要

第 3.4 章で述べたように，バンクコンフリクトの発生は命令の書き込みが遅れ，性能の低下に繋がる．そこで先行研究では，同時にレジスタファイルに書き込みを行う命令を予測する書込予測機構を追加している．書込予測機構は命令の依存関係の情報を用いて，命令の実行順序を予測し，命令の書き込み先をそれぞれ異なるバンクに割り当てることでバンクコンフリクトを回避している．

4.2 書込予測機構

書込予測機構は同じサイクルでレジスタファイルに書き込みを行う可能性の高い命令をリネームステージ内で予測する．しかし，高精度な予測には，依存関係や先行命令の実行順序の情報を用いた複雑で大規模なハードウェアが必要となる．そこで，先行研究では，予測対象を ADD 命令などの実行サイクル数が決まっているシンプルな命令のみとすることで，予測機構のハードウェア規模を削減している．書込予測機構についてアルゴリズム 1 を用いて説明する．

先行研究の書込予測機構のアルゴリズムは以下の 3 つのステップで構成されている．また，$last_assigned_bank$ は 0 で初期化を行う．

Assign physical register

デスティネーションレジスタへの割り当ては割り当てるバンクの情報を用いて Free List から物理レジスタを割り当てる．ある命令 A が命令 B に依存関係を持つとき，命令 A の実行は命令 B が完了してからとなる．そのため，命令 A は次のサイクルでリネームされる命令列とバンクコンフリクトを引き起こす可能性が高くなる．よって，バンクの割り当ては命令 A が割り当てられたバンクの次のバンクから割り当てることにより，バンクコンフリクトの回避を図る．

Algorithm 1 レジスタ書込予測のアルゴリズム

```
Require: BANKS:バンクの数
Ensure: t ≥ 0 ∧ t < BANKS
  initialization
  last_assigned_bank ← 0
  1:Assign physical register
  t ← last_assigned_bank
  for i = 0 to renamed_width − 1 do
    DestinationRegister[i] ← pop(Freelist[t + +])
    if t ≥ BANKS then
      t ← 0
    end if
  end for
  2:Detect dependency
  for i = 1 to renamed_width − 1 do
    for j = 0 to i do
      if detect_dependency then
        depend_bank_number ← i
        break
      end if
    end for
  end for
  3:Propagate
  if detect_dependency then
    last_assigned_bank ← (depend_bank_number + 1) mod BANKS
  end if
```

Detect dependency

第 2.3 章で述べたように真の依存関係はレジスタリネーミングによって取り除くことができない．そこで，真の依存関係を持つ命令の書き込み先に割り当てたバンクを記憶することで，バンクコンフリクトの回避を図る．書込予測機構によって追加するハードウェアを単純にするため，ここでは初めに見つけた真の依存関係を持つ命令に割り当てられたバンクのみ保存する．

Propagate

記憶したバンクの番号をインクリメントし，保存する．次のサイクルの物理レジスタへの割り当ては保存したバンク番号から行う．

図 4.9 を用いて，書込予測機構によるバンクコンフリクトの回避について述べる．図 3.8 と同様に命令の実行サイクル数は全て 1 サイクルとし，同時に 4 つの命令を実行することができるプロセッサを考える．図 3.8 と同様に初めて発行される 0 番から 3 番の命令は 0 から順番に異なるバンクが割り当てられる．このとき，割り当てられた 0 番から 3 番の命令を順番に確認し，初めに見つけた真の依存関係がある 1 番の命令に割り当てられたバンク番号である 1 を $detect_dependency$ に保存する．最後に $detect_dependency$ に保存されたバンク番号をインクリメントし，

$last_assigned_bank$ に記憶する．次のサイクルで 4 番から 7 番の命令を割り当てるとき，$last_assigned_bank$ に記憶しておいたバンク番号の 2 から順番に異なるバンクを割り当てていく．ここで命令 0 がレジスタに書き込むタイミングを n サイクル目とすると，n+1 サイクル目には 1，4，5 番の命令が書き込みを行っている．しかし，それぞれの書き込み先のバンク番号は 1，2，3 と異なったバンクに書き込まれていることがわかる．同様に，n+2 サイクル目には 2，6 番の命令が書き込みを行っているがバンクコンフリクトは発生していないことがわかる．このように，先行研究の書込予測機構では真の依存関係がある命令の情報を使い，バンクコンフリクトの回避を図っている．

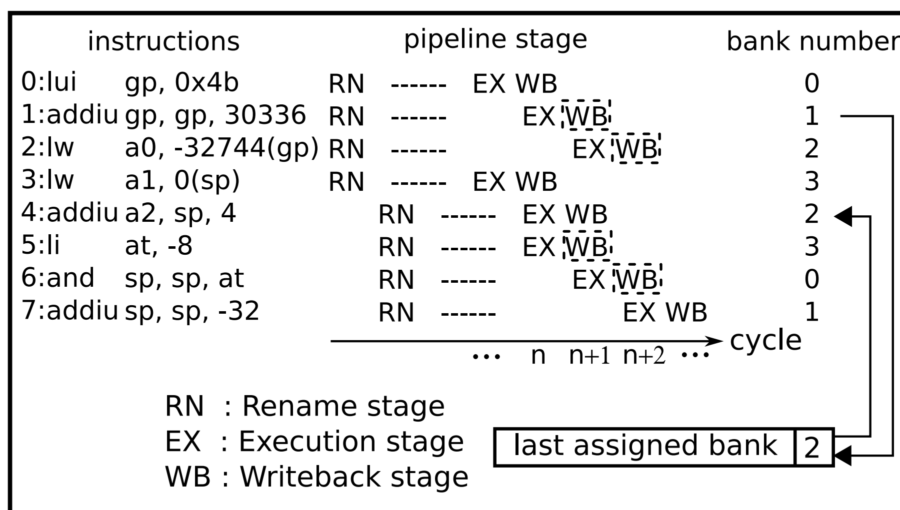

図 4.9: 書込予測機構を用いたバンク割り当て

4.3 問題点

先行研究には，書き込み予測による新たなバンクコンフリクトの発生と高 IPC 環境下での効果の 2 つの問題点が存在する．まず，前者について述べる．書込予測機構では，真の依存関係を持つ命令に割り当てられたバンク番号を保存し，次のサイクルでは，保存した番号の次のバンクから命令に割り当てる．そのため，次のサイクルで最後に割り当てるバンクは，予測機構に保存したバンクと同じになる．このとき，依存関係などにより，最後に割り当てた命令に実行の遅れがなければ，バンクコンフリクトが発生する．図 4.10 を用いて，この問題を説明する．図 4.9 と同様に命令の実行サイクル数は全て 1 サイクルとし，同時に 4 つの命令を実行することができるプロセッサを考える．まず，0 番から 3 番の命令を割り当てるとき，バンク 0 から順番に異なるバンクを割り当て，初めに見つけた真の依存関係がある命令に割り当てられたバンク番号をインクリメントした数字を $last_assigned_bank$ に記憶する．次の 4 番から 7 番の命令を割り当てるとき，$last_assigned_bank$ に記憶しておいたバンク番号の 2 から順番に異なるバンクを割り当てていく．ここで n+1 サイクル目に注目すると，1，4，7 番の命令が書き込みを行っている．その中で 1 番と 7 番の命令の書き込み先バンク番号はともに 1 番になっているこ

とがわかる．これは 4 番から 7 番の命令にバンクを割り当てるとき，真の依存関係があった命令のバンク番号である 1 をインクリメントした 2 から順番に割り当てるため，4 つめにリネームされる 7 番の命令には必然的にバンク 1 が割り当てられる．図 4.10 の例では 7 番の命令には依存関係がないため，先行命令のうち，依存関係があり実行タイミングが遅れてきた 1 番の命令の書き込みが重なってしまう．このように先行研究の書込予測機構では次のサイクルで割り当てられる命令がバンクと同じ数だけリネームされたとき，最後の命令に依存関係がなければバンクコンフリクトが発生してしまう．

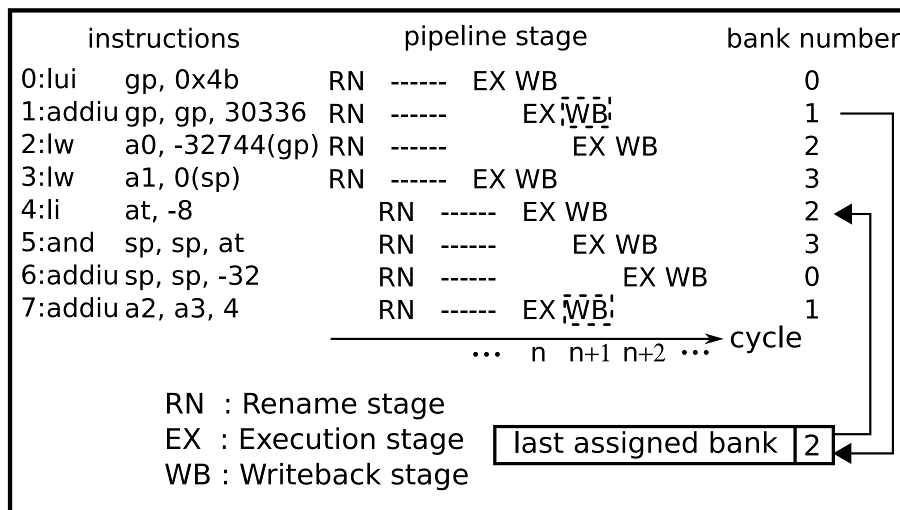

図 4.10: 書込予測機構の問題点

また，先行研究は任意のスーパースカラコアを生成することが出来る

ツールセットである FabScalar [2] を用いて性能の評価を行っている．バンクコンフリクトは 1 サイクルに書き込まれる命令が多いほど発生する確率が高くなるが，FabScalar で構築できるプロセッサの IPC は高いとはいえない．そのため，バンクコンフリクトが頻発している状況での効果が不明となっている．したがって，高 IPC の環境下で先行研究の効果を検証する必要がある．

5　書き込み予測機構の改良

本章では，本研究で提案する書込予測機構の詳細について述べる．

5.1　概要

第 4.3 章で述べたとおり，先行研究の書込予測機構では次のサイクルで割り当てられる最後の命令に依存関係がなかった場合，バンクコンフリクトが発生してしまう．そこで，本研究では先行研究の書込予測機構の予測方法を変更することにより，バンクコンフリクトの回避を図る．提案手法では，先行研究と同様に命令の依存関係の情報を用いる．依存関係がある命令を優先的に割り当てることにより，そのバンク内の命令の実行タイミングは全て同じように遅らせ，バンクコンフリクトの低減を目指す．

5.2　書き込み予測機構の詳細

書込予測機構をアルゴリズム 2 を用いて説明する．

Algorithm 2 レジスタ書込予測のアルゴリズム

Require: $BANKS$:バンクの数

Ensure: $t \geq 0 \wedge t < BANKS$

```
initialization
last_dependence_bank ← 0
last_assign_bank ← 0
1:Detect dependency
depend_bank_number ← -1
for i = 1 to renamed_width − 1 do
  for j = 0 to i do
    if detect_dependency then
      depend_bank_number ← i
      break
    end if
  end for
end for
2:Assign physical register
for i = 0 to renamed_width − 1 do
  if last_write_instruction < BANKS then
    t ← (i + last_assign_bank + 1) mod BANKS
    DestinationRegister[i] ← pop(Freelist[t])
  else
    t ← (last_dependence_bank + (BANKS + i − depend_bank_number)) mod BANKS
    DestinationRegister[i] ← pop(Freelist[t])
  end if
  if i = depend_bank_number then
    dependence_bank ← t
  end if
end for
last_dependence_bank ← dependence_bank
last_assign_bank ← t
```

提案手法の書込予測機構は，以下の 2 ステップで構成されている．

Detect dependency

1 サイクルでリネーミングされる命令のうち，真の依存関係を持つ命令が何番目にリネーミングされたかを記憶する．先行研究の書込予測機構と同様に，追加するハードウェアを単純化するために，初めに発見した真の依存関係を持つ命令のみに注目している．

Assign physical register

提案手法では，Detect dependency で発見した真の依存関係を持つ命令を前のサイクルで依存関係があった命令が割り当てられたバンクを記憶している $last_dependence_bank$ の番号に割り当てるように順番を操作する．例えば，前のサイクルで依存関係があった命令がバンク 0 に割り当てられ、現在リネームされた命令列で 1 番の命令に真の依存関係が存在したとき，0 番の命令はバンク 3 へ，1 番の命令はバンク 0 へ，2 番の命令はバンク 1 へ，3 番の命令はバンク 2 へ割り当てられる．リネームされた命令列に依存関係がなかった場合，0 番の命令を $last_dependence_bank + 1$ のバンクへ，それ以降の命令を順番に割り当てるようになっている．また，前のサイクルの書込命令数がバンクの数に満たない場合，最後に割り当てられたバンク

を記憶している $last_assign_bank$ をインクリメントした数から順番に割り当てる．このとき，命令間の依存関係を確認し，真の依存関係があれば $last_dependence_bank$ を更新する．デスティネーションレジスタへの割り当ては割り当てたバンクの情報を用いて Free List から物理レジスタを割り当てる．

5.3 提案手法実装時の動作

図 5.11 を用いて，提案手法の書込予測機構の動作について説明する．図 4.10 と同様に命令の実行サイクル数は全て 1 サイクルとし，同時に 4 つの命令を実行することができるプロセッサを考える．まず，0 番から 3 番の命令を割り当てるとき，先に依存関係がないか確認する．この例では，1 番の命令に依存関係があったため，1 番の命令を $last_dependence_bank$ の番号のバンクに割り当て，他の命令は順番にバンクに割り当てる．次の 4 番から 7 番の命令を割り当てるとき，先ほどと同様にバンクを割り当てる前に依存関係の有無を確認する．この例では，5 番の命令に依存関係があったため，5 番の命令を $last_dependence_bank$ の番号のバンクに割り当て，他の命令は順番にバンクに割り当てる．ここで，先行研究でバンクコンフリクトが発生していた n+1 サイクル目に注目すると，書き込

みを行う命令は1，4，7番で，それぞれ割り当てられたバンク番号は0，3，2とそれぞれ違うバンク番号を指していることがわかる．同様に他のサイクルでも同じタイミングで同じバンクに書き込みを行っている命令は存在しないことから，バンクコンフリクトを回避していることがわかる．このように依存関係により，実行タイミングが遅れるような命令を優先的にバンクに割り当てることにより，バンクコンフリクトの回避を図る．

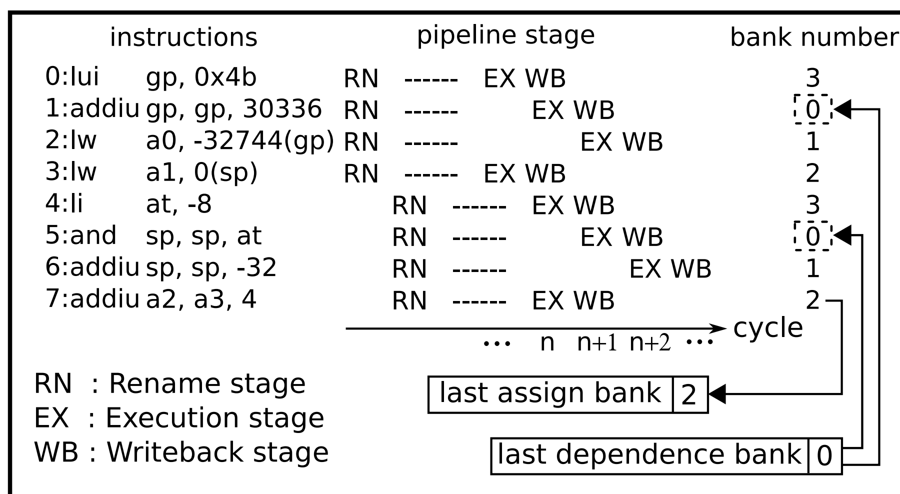

図 5.11: 提案手法の書込予測機構の動作

6　書込予測型バンクレジスタファイルの再評価

第 4.3 章で述べたように，高 IPC の環境下での先行研究の評価が必要となる．そこで，スーパースカラプロセッサのシミュレータの 1 つである SimpleScalar を用いて，高 IPC 環境下での書き込み予測型バンクレジスタファイルの評価を行う．本研究では，SimpleScalar のツールセットのうち，命令スケジューリングを行う sim-outorder 上に先行研究の機構を実装する．SimpleScalar は Tomasulo 方式を採用しているが，バンクレジスタファイルは物理レジスタ方式が対象である．そこで，本研究では，SimpleScalar 上に仮想的に物理レジスタや RMT，Free list などを作成する．これにより，デスティネーションレジスタの物理レジスタへの割り当てや，コミット時の物理レジスタの解放といった物理レジスタ方式の動作を模倣する．

また，プロセッサのシミュレーション評価に用いるベンチマークの命令数は非常に多いため，その全ての命令をシミュレータで実行するには膨大な時間がかかる．そのため，ベンチマークのシミュレーションには，シミュレーション・ポイントを用いる場合がある．シミュレーション・ポイントとは，そのポイントだけを実行することにより，プログラム全体の振る舞いが推定できるようなプログラムの実行列の一部分のことである．

シミュレーション・ポイントを選出するツールの一つとしてSimPoint [5] が挙げられる．SimPointはプログラムを実行したときの動的な命令列を一定の区間で区切る．この区間をインターバルとよび，それぞれのインターバルに含まれる基本ブロック(分岐や合流を含まない命令のまとまり)を基にk-means法を用いてクラスタリングを行う．それぞれのクラスタで代表的なインターバルを取り出すことで，シミュレーション・ポイントを選出している．本研究では，SimPointを用いて選出されたシミュレーション・ポイントにしたがい，ベンチマークの性能評価を行った．

7　評価

本章では，先行研究の書込予測機構と提案手法の書込予測機構の性能評価を行う．

7.1　評価環境

先行研究と提案手法の予測機構の効果を調べるため，SimpleScalar を用いたサイクルレベルシミュレーションにより評価を行う．評価プログラムとして，SPEC2000INT より，gzip，vpr，gcc，mcf，crafty，gap，bzip2 を用いた．また，SimPoint のインターバルサイズを 100M に設定し，選出されたシミュレーション・ポイントを用いて評価を行った．表 7.1 に評価を行った手法を示し，表 7.2 にプロセッサの構成を示す．

表 7.1: 評価を行った手法

手法
4 ライトポートのマルチポートレジスタファイル
予測機構なしの 4 バンクレジスタファイル
先行研究を用いた 4 バンクレジスタファイル
提案手法を用いた 4 バンクレジスタファイル

表 7.2: プロセッサの構成

Fetch, Dispatch, Issue, Commit width	4 inst
RUU size	128 entry
Load/Store queue	64 entry
memory port	2
Integer ALUs	4
Integer MUL/DIV	1
Branch prediction	combined with 1024 entries 2level predictorwith 8bits history width
BTB	2048 sets
Memory latency	first 100 cycles, inter 2 cycles
L1 I Cache	32KB, 64B/line, 4way 1cycle latency
L1 D Cache	32KB, 64B/line, 4way 1cycle latency
unified L2 Cache	512KB, 128B/line, 4way 6cycle latency

7.2 性能評価

図 7.12 に，各手法でのベンチマークの実行時間を示す．各実行時間はマルチポートレジスタファイルを 1 として正規化している．

先行研究では，通常のバンクレジスタファイルと比べ，実行時間を平均 0.71%，最大 1.78%削減している．提案手法では，通常のバンクレジスタファイルと比べ，実行時間を平均 0.85%，最大 2.67%削減している．また，gzip や crafty では，IPC が 2 程度になっていることから，評価を取るうえで十分な環境で評価を行うことができていると考えられる．

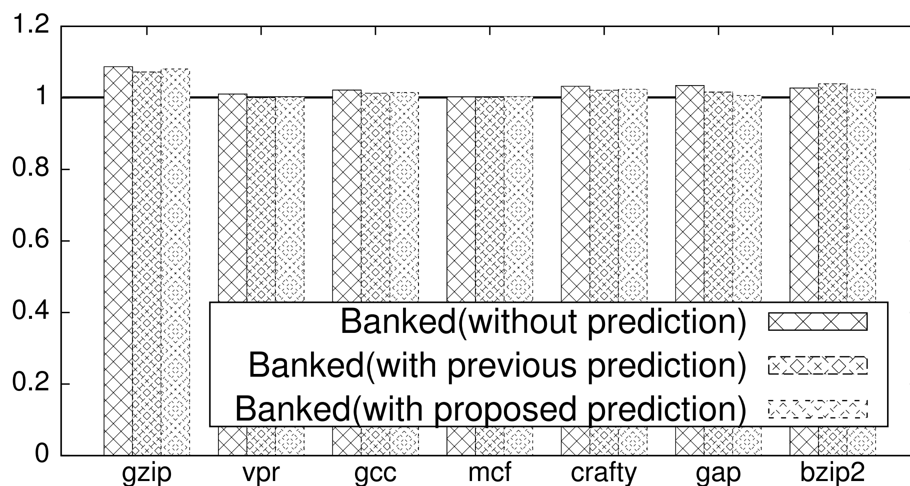

図 7.12: 評価結果

7.3 考察

本研究で先行研究の再評価を行った結果，予測機構の追加による性能の向上は十分にみられた.

gap では，提案手法を用いることにより，通常のバンクレジスタファイルに比べ，実行時間を大きく削減することができた．これは，ベンチマークに依存関係をもつ命令が多く存在し，予測機構によってバンクコンフリクトを回避できたと考えられる．bzip2 では，通常のバンクレジスタファイルに比べ，先行研究の予測機構では実行時間が増加していたが，提案手法では削減することができた．これは，実行サイクル数が異なる命令が多く，先行研究では命令の実行サイクル数は考慮していない設計のため，

実行時間が増加したものと考えられる．提案手法では，依存関係がある命令の両方が同じ実行サイクル数の時は対応できるようになっているため，実行時間を削減することができたと考えられる．gzip では，バンク化することにより，他のベンチマークと比べ，実行時間が大きく増大している．物理レジスタ方式では分岐予測ミスなどの例外処理をするとき，物理レジスタの状態を復帰するために命令が発行されてからコミットされるまでの生存期間を記録する Active List を必要としており，命令を発行するためには Active List が空いていなければならない．しかし，シミュレータの実行結果から gzip では Active List が詰まっている状態が多いことがわかった．そのため，バンクコンフリクトによって，命令をコミットできなくなったときに後続の命令に与える影響が大きいため，他のベンチマークと比べ，実行時間が増大したと考えられる．

8 おわりに

本稿では，レジスタ書込ポート予測を用いたバンクレジスタファイルの性能の再評価と，別の方法でバンクコンフリクトを回避する書込予測機構を提案した．提案手法では命令間の依存関係の情報を使い，正しくバンクに振り分けることで，バンクコンフリクトの発生を抑えることができる．評価結果によると，提案手法の書込予測機構は通常のバンクレジスタファイルに比べ，ベンチマークプログラムの実行時間を平均 0.85%，最大 2.67%削減することに成功した．今後の展望として，書込予測機構のアルゴリズムをさらに発展させ，性能を向上させることが挙げられる．

謝辞

本研究を進めるにあたり，多数のご指導，ご助言を頂いた佐々木敬泰助教，近藤利夫教授，並びに深澤祐樹研究員に深く感謝いたします．また，様々な場面で支えていただいた修士１年の萱室高樹先輩やコンピュータアーキテクチャ研究室の皆様にも心より感謝いたします．

参考文献

[1] Hiroaki Kawashima, et al.:"Register port prediction for a banked register file", Proc. of the Third International Symposium on Computing and Networking, pp.551-555, Dec.2015.

[2] Niket Kumar Choudhary, et al.:"FabScalar: composing synthesizable RTL designs arbitrary cores within a canonical superscalar template", Proc. of the 38th Annual International Symposium on Computer Architecture, pp.11-22, June.2011.

[3] Austin, Todd, Eric Larson, and Dan Ernst. "SimpleScalar: An infrastructure for computer system modeling.", Computer 35.2, pp.59-67, Feb.2002.

[4] Dwiel, Brandon H., Niket K. Choudhary, and Eric Rotenberg. "FPGA modeling of diverse superscalar processors" Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on. IEEE, Apr.2012.

[5] Hamerly, Greg, et al. "Simpoint 3.0: Faster and more flexible program phase analysis.", Journal of Instruction Level Parallelism 7.4,

pp.1-28, Jun.2005.

A　プログラムリスト

B　評価用データ