2006－ARC－169 (4)
2006／8／1

レジスタファイルの書き込み時タイミングエラーの検出・回復手法

荻野 健[††] 入江英嗣[†††,†]
五島正裕[†] 坂井修一[†]

プロセッサの過渡故障対策の研究が盛んに行なわれているが、その多くはデータのソフトエラーに着目している。しかしながら過渡故障にはタイミングエラーと呼ばれる、制御のエラーも含まれ、冗長性だけではレジスタファイルの書き込み時の過渡故障に対処することはできない。レジスタファイルは重要なアーキテクチャ・ステートを保持するため、レジスタファイルの書き込みを保証することは、全ての過渡故障対策の基点となる。

本論文では、レジスタファイルにおける書き込み時のタイミングエラーを検出、回復する手法について提案する。シミュレータによる評価の結果、提案手法は 8 エントリのバッファの追加で、実行オーバヘッドを 4.5%に抑え、レジスタファイルへの書き込みをタイミングエラーから保護できることが分かった。

Detection and Recovery of Register File Timing Errors

KEN OGINO,[††] HIDETSUGU IRIE,[†††,†] MASAHIRO GOSHIMA[†] and SHUICHI SAKAI[†]

Transient fault tolerance is increasingly becoming a popular reserch topic.Although prior research has mainly focused on soft errors, there is another kind of transient faults called timing errors which are caused by signal delay.

This paper addresses the timing errors in the processor register files(RF). We propose write assurance buffer(WAB). When a register is updated, the new value is also duplicated in WAB. Comparing the value of later read to the register with the value in WAB can detect timing errors.

Evaluation results show that WAB with eight-entry can detect timing errors while degrading performance by 4.5%.

1. はじめに

LSIの微細化に伴い、プロセッサの過渡故障(transient-fault)に対する耐性が低くなると予測されており[5]、その対策が広く研究されている。過渡故障とは、何らかの外的要因によりランダムに発生する瞬間的なエラーのことである。プロセッサにおける過渡故障の主な例としてソフトエラーがあり、これは中性子線の衝突によってシングルビットエラーを引き起こす。それによりプログラムが間違った結果を出したり、最悪の場合システムがダウンする危険性がある。プロセッサは今や個人の生命や財産に関わるような場面で多く使われており、過渡故障対策は今後益々重要になっていくものと考えられる。

既存の過渡故障対策の研究の多くは冗長性に基づいている。しかしながら過渡故障にはソフトエラーだけではなく、タイミングエラーと呼ばれる制御のエラーも存在する。タイミングエラーとは、何らかの原因でクロック、あるいはデータのタイミングがずれてしまい、結果間違った値を読み書きしてしまうことを言う。このようなエラーは、既存の冗長化手法では対処できないことが多い。例えば演算を冗長化して、それらの結果を比較することで過渡故障を検出することができても、結果の最終的な書き込み時にタイミングエラーが発生してしまう可能性がある。このような書き込みの正しさを保証するには一度書き込んだ値をもう一度読み出し、正しい値かどうかをチェックする必要がある。

タイミングエラーの要因としては、熱、製造ばらつき等が挙げられ、いずれも近年のプロセッサにおける主要な課題である。また、低電力化のためのDVS[6](Dynamic Voltage Scaling)も要因の一つである。

† 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†† 東京大学大学院 新領域創成科学研究科
Graduate School of Frontier Sciences, The University of Tokyo
††† 科学技術振興機構
Japan Science and Technology Agency

我々は、タイミングエラーを含めた全ての過渡故障に対して、確実にエラーから回復できるプロセッサアーキテクチャが今後重要になると考える。本論文では特に、レジスタファイル書き込み時のタイミングエラーを防ぐ手法を提案する。レジスタファイルは重要なアーキテクチャ・ステートを保持するため、レジスタの正しさの保証は高信頼化の基点となる。

本手法ではレジスタファイルとは別に小さなバッファを用意し、レジスタの値をそのバッファにも書き込む。後にバッファ内の値と対応するレジスタファイルの値を読み出して一致比較をし、書き込みミスを検出する。提案するバッファはレジスタファイルに比べて回路規模が小さく、タイミング・クリティカルパス上から外れるため、タイミングエラー耐性が高い。そのためバッファ内から回復を行なうことができる。本手法は命令のソース・レジスタの読み出しや、レジスタファイルの空きポートに着目し、それらを利用して一致比較することでレジスタファイルに余分なポートを追加することなく書き込みミスを検出できる。

以下、本論文は次のように構成される。2章でまずタイミングエラーについて述べる。続く3章で提案手法を説明し、4章でその評価を行なう。5章で関連研究を述べた後、最後に6章でまとめを行なう。

2. タイミングエラー

2.1 タイミングエラーとは

タイミングエラーとは、何らかの原因でクロック制御のタイミングがずれる結果起きるエラーである。クロック周波数はタイミング・クリティカルパス上の遅延に合わせて決定されるため、その上を流れるデータは他のパスに比べて正確なタイミングが要求される。したがって何らかの原因でタイミングのずれが生じると、間違った値を読み込んでしまうなどのエラーが起こる。近年では、タイミングのずれとなる要因が熱や製造ばらつき、DVS[6]など、動的なものが多くなってきた。そのため、設計段階だけでタイミングエラーに対処することは難しくなってきている。タイミングエラーの様子を図1に示す。各ビットは熱や製造ばらつきの影響でタイミングが一定ではない。図1ではbit1とbit4の遅延が大きく、クロックタイミングに間に合わなくなってしまっている。本来はbit1'とbit4'の値が取り込まれるはずだが、bit1とbit4の値が間違って取り込まれてしまう。

レジスタファイルの書き込み時のタイミングエラーは既存のECC(Error Correcting Code)や冗長化で防ぐことはできない。ECCが正しく計算できていても、タイミングエラーによって書き込まれる値はいかようにも変化しうるからである。冗長化も同様にタイミングエラーの根本的な解決にはならない。例え回路を2重に冗長化しても、同じ設計である以上、同じタイミングエラーが生じる可能性がある。

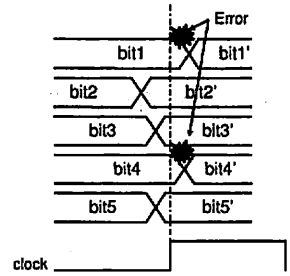

図1 タイミングエラーの例

過渡故障は検出、再実行により回復することができる。ただし、そのためにはレジスタファイルなどのアーキテクチャ・ステートが常に正しく保たれていなければならない。レジスタファイルへの書き込みでタイミングエラーが発生すると、アーキテクチャ・ステートの一部が失われ、回復が不可能となる。クロック周波数を落とさずに、レジスタの書き込みタイミングエラーから回復するためには、一度書き込んだ値をもう一度読み出し、書き込みが正しく行われたかをチェックしなければならない。

2.2 レジスタファイルのタイミングエラー検出方法

書き込まれた内容が正しいかどうかを判断するには、正しい値をどこかに保持しておく必要がある。本手法では、小容量のバッファを用意し、レジスタファイルの書き込みと同時にそのバッファにも書き込む。以後、本論文ではこのバッファを書き込み保証バッファと呼ぶ。書き込み保証バッファは回路規模が小さいがゆえにタイミングマージンを大きく設計することができ、正しく値を保持する。

次に、レジスタファイルに書き込んだ内容をもう一度読み出して、書き込み保障バッファの内容と一致比較をしなければならない。しかし、ただ単純にレジスタファイルにリードポートを追加し、読み出すのは得策ではない。回路面積はポート数の二乗に比例するため、リードポートを書き込み保証バッファの読み出し用に追加するのは大きなコストとなる。回路面積が増えた結果、新たなタイミング・クリティカルパスを作り、結果タイミングエラーが増えてしまうことも考えられる。そこで本論文では、レジスタファイルのリードポートを増やさずに、タイミングエラーを検出する

手法を提案する。

3. 提案手法

本節で提案手法について説明する。まず書き込み保証バッファについて説明する。その後レジスタファイルからどのようにして値を読み出してエラーを検出、回復するかについて述べる。

3.1 書き込み保証バッファ

書き込み保証バッファはレジスタファイルに書き込んだ値を、検証するまで正しく保持するためのものである。タイミングエラーから守るために、バッファのエントリ数は小さく抑える必要がある。ソフトエラーに関しては関連研究[1),7)-9)]を用いて対処されているものとする。

書き込み保証バッファは整数レジスタと浮動小数点レジスタにそれぞれ用意する。ライトバック時、レジスタファイルへの書き込みは同時に書き込み保証バッファにも書き込まれる。この時、書き込んだ先の物理レジスタ番号も合わせて書き込み保証バッファに書き込まれる。レジスタファイルへの書き込みを全て保持するためには、書き込み保証バッファのエントリ数をレジスタファイルのライトポート数以上にする必要がある。

書き込み保証バッファ内のエントリに保持される値は、対応するレジスタファイルの値と比較する必要がある。比較した結果両者が一致したら、正しく書き込めたことが確認できたので、書き込み保証バッファのエントリを解放する。

書き込み保証バッファの空きエントリ数が足りない場合、バックエンドを 1 サイクルだけストールさせる。ストールの間に一致比較の結果が出てエントリが解放されるのを待つ。また、ストール中にレジスタファイルから値を読み出し、次のサイクルの比較に用いる。書き込みに充分な空きエントリが書き込み保証バッファに確保されるまで、この操作を繰り返す。

次に、どのようにしてレジスタファイルから値を読み出し、タイミングエラーを検出するかについて述べる。

3.2 レジスタファイルからの読み出し方法

レジスタファイルのリードポートを増やさずに、検証のためのレジスタ読み出しを行う手法は 2 種類存在する。本論文では 2 つの手法をそれぞれポート・シェア (port share)、ポート・スチール (port steal) と呼ぶ。

3.2.1 ポート・シェア

ポート・シェアは、まだ一致比較が行なわれていないレジスタが、ソース・オペランドとして読み出される時にその値を利用して比較を行なう方法である。レジスタに書き込まれた値は、近いうちにオペランドとして読み出される可能性が高いとの予測に基づいている。

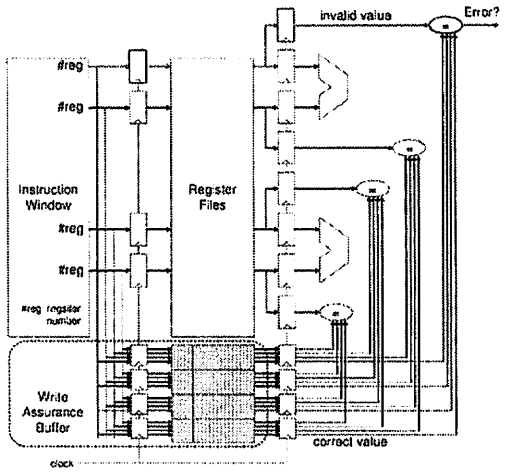

図 2 ポート・シェアの構成

図 2 にポート・シェアの構成を示す。この手法の場合、書き込み保証バッファは物理レジスタ番号をキー、レジスタの値をバリューとする CAM(Content Addressable Memory) で構成される。命令発行時、命令ウィンドウからレジスタファイルへ送られるリクエスト (物理レジスタ番号) をキーとして書き込み保証バッファ内を連想アクセスする。該当する物理レジスタ番号を持つエントリが存在すればレジスタから読み出されたオペランド値とエントリ内の値を比較する。しかし、オペランドがフォワーディングによって供給される場合、レジスタがバイパスされるため、ポート・シェアは行えない。

3.2.2 ポート・スチール

命令のソースオペランドの数が少ない時や、フォワーディングによって供給された場合、レジスタのリードポートに空きが生じる。ポート・スチールはその空きリードポートを利用して、レジスタファイルから値を読み出して比較する方法である。図 3 にポート・スチールの構成を示す。ポート・シェアと違い、この手法は連想アクセスをしないので、書き込み保証バッファは CAM でなくてもよい。命令発行時、書き込み保証バッファは命令ウィンドウと同じようにレジスタファイルにリクエスト (物理レジスタ番号) を送る。当然通常は命令ウィンドウからのリクエストが選択されるが、フォワーディングや元々ソースオペランドがないなど、

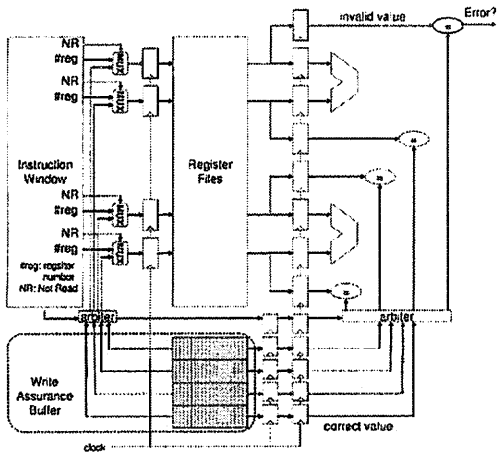

図 3　ポート・スチールの構成

レジスタを読み込まない場合は書き込み保証バッファのリクエストが選択される。その後レジスタファイルから読み出されてきた値と書き込み保証バッファ内の値を比較する。

3.3 回復方法

書き込みミスが検出された場合、パイプラインをフラッシュし、書き込み保証バッファが保持する正しい値をレジスタにコピーすることで回復を行なう。ただし、書き込みミスをした値がすでに後続命令で使われ、コミットされていないことを保証しなければならない。

ポート・シェアの場合、それは保証されている。ポート・シェアはソース・オペランド読み出しに乗じて一致比較を行なう手法であるから、書き込みミスした値が最初に読み出された時に検出できる。

一方ポート・スチールだけの場合、検出が間に合わずに間違えた値のままコミットされてしまうこともありうる。そこでポート・スチールの場合は全てのソースオペランドが一致比較されている命令のみコミットできる、という制約が必要となる。

4. 評　価

本節では書き込みエラー検出を行なうことによるオーバヘッドを評価する。まず評価環境、評価モデルについて述べ、その後評価結果を説明する。

4.1 評価環境

SimpleScalar Tool Set Version 3.0d[2] の sim-outorder シミュレータに提案手法を実装し, 評価を行なった。ベンチマークは SPEC2000 の CINT、CFP から用いた。表 1 に使用したベンチマークと入力についてまとめた。

表 1　ベンチマークプログラムと入力

プログラム名	入力	実行命令数
ammp	ammp.in	1G
apsi		1G
art	-scanfile c756hel.in -trainfile1 a10.img -trainfile2 hc.img -stride 2 -startx 110 -starty 200 -endx 160 -endy 240 -objects 10	1G
mgrid	mgrid.in	1G
mesa	-frames 1000 -meshfile mesa.in -ppmfile mesa.ppm	1G
swim	swim.in	1G
wupwise		1G
bzip2	input.random	1G
crafty	crafty.in	1G
gcc	cccp.i -o cccp.s	1G
gzip	input.compressed 2	1G

表 2　測定条件

Simulator	SimpleScalar(sim-outorder)
way 数	4
命令セット	Alpha 21264 ISA
命令ウィンドウ	32 エントリ
LSQ	16 エントリ
機能ユニット	4 iALU, 1 iMULT/DIV 2 Ld/St, 1 fpALU 1 fpMULT/DIV/SQRT
レジスタファイル	1 cycle access latency
L1 ICache	32KB, LRU, 4way, 32Bytes line, 2cycle access latency
L1 DCache	32KB, LRU, 2way, 32Bytes line, 2cycle access latency
L2 Cache	512KB, LRU, 4way, 64Bytes line, 12cycle access latency

4.2 評価モデル

ベースモデルはアウトオブオーダ実行を行なうスーパスカラプロセッサで、各モデル共通の条件を表 2 に示す。評価モデルは以下の 3 つを用意した。

- share: ポート・シェアのみを採用したもの
- steal: ポート・スチールのみを採用したもの
- unified: ポート・シェアとポート・スチールの両方を採用したもの

4.3 評価結果

図 4、5、6 に評価結果を示す。それぞれ share、steal、unified モデルの結果である。プロセッサ性能の指標には相対 IPC を用いており、ベースモデルの IPC を 1 として正規化している。グラフには 3 本一組のバーが 12 あり、左の各組が SPEC2000 のプログラムに相当する。一番右の組は調和平均を表す。3 本のバーは左から順に 8、16、32 と書き込み保証バッファのエントリ数が変化する。

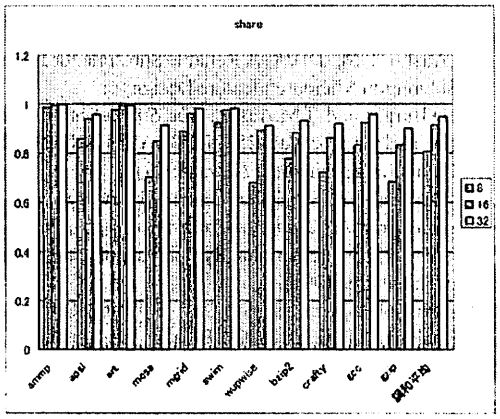

図 4　ベンチマーク毎の正規化した IPC(share)

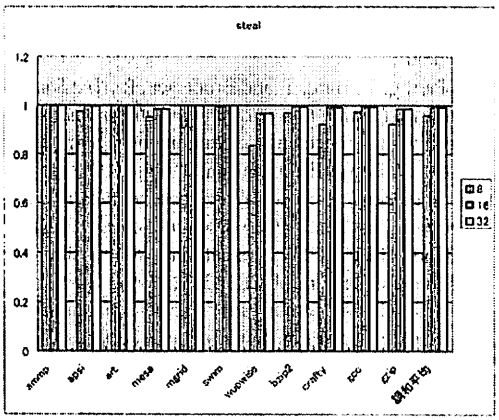

図 5　ベンチマーク毎の正規化した IPC(steal)

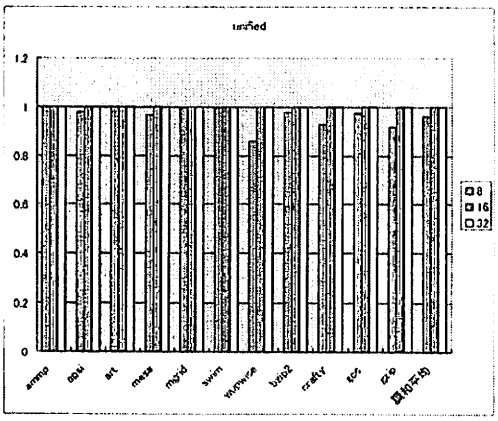

図 6　ベンチマーク毎の正規化した IPC(unified)

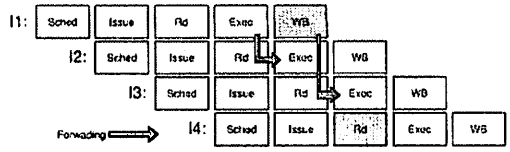

図 7　da

結果より、steal と unified の書き込み保証バッファのエントリ数が 16 ならば、相対 IPC の平均はそれぞれ 0.991、1 でありほとんどオーバヘッドはない。また、8 エントリの場合でもそれぞれ 0.955、0.961 であり、オーバヘッドは 4.5%以下に抑えられている。それに対して share は相対 IPC が低い。share の相対 IPC の平均は 16 エントリで 0.912、8 エントリで 0.805 となっている。

share のオーバーヘッドが大きい理由は、フォワーディングによってソースレジスタ読み出しの数が減り、ストールしやすいためである。今回の評価ではレジスタファイルのアクセスレイテンシを 1 サイクルに設定した。そのため、例えば図 7 にあるように back-to-back で実行される命令列があった場合、先頭命令 I1 の結果がレジスタから読み出されるのは最低 3 サイクル後の命令 I4 である。命令 I2、I3 はフォワーディングによってオペランドを得るのでポート・シェアを行なうことができない。先頭命令 I1 に依存している命令が、命令 I4 以降にある割合は少ないと考えられ、ポート・シェアが有効に利用されなかったといえる。

以上をまとめると、提案しているタイミングエラー検出・回復手法では、8 エントリ、または 16 エントリ程度のバッファでも、実行オーバヘッドを小さく抑えることができることがわかった。特に steal と unified においては 8 エントリの場合でもオーバヘッドは平均して 4.5%以下に抑えられている。

5. 関連研究

過渡故障対策として様々な手法が提案されている。

IRI[8] は命令を 2 回発行して結果を比較することで過渡故障を検出している。また、データ投機実行に用いられる命令再発行機構を利用することで、過渡故障を起こした命令に依存する命令だけを選択的に再実行させることができる。レジスタファイルの保護は ECC で行なうのみで、タイミングエラーによる書き込みミスについては考慮していない。

AR-SMT[7] はスレッドを複製し、SMT[3] プロセッサ上で 2 つのスレッドをある程度の時間差をもって並列に実行する。先行スレッドの結果と後続スレッドの結

果を比較することでエラーを検出する。先行スレッドの結果を後続スレッドが分岐予測やデータ予測に使うことで、後続スレッドの実行効率を高めている。

これらの手法が同じハードウェアで複数回実行して過渡故障を検出しているのに対し、DIVA[1]は2つのコアを持ち、一方の演算結果をもう一方のチェック専用のコアがチェックして検出を行なう。チェック用のコアは演算結果のチェックに徹することで、パフォーマンスはある程度低くてもよい。したがって設計マージンを大きくとることができる。

タイミングエラーに対する技術としてRAZOR[4]がある。これは回路レベルでタイミングエラーを検出、訂正する手法である。RAZORは通常のフリップフロップ(Main)に加えてもう一つ、ShadowLatchと呼ばれるフリップフロップを用いる。ShadowLatchはメインのクロックに比べて一定値だけ位相の遅れた遅延クロックで動作する。したがって、例えMainがタイミングエラーで誤った値を取り込んでも、ShadowLatchは正しい値を得ることができる。この手法はロジック回路に対しては有効であるが、レジスタファイルなどのようなメモリセルで構成されるユニットに対しては使うことができない。

6. おわりに

本論文では、ソフトエラーのような値の反転だけではなく、タイミングエラーなども含んだ全ての過渡故障に対応できるアーキテクチャが今後重要であることを指摘した。本論文は、重要なアーキテクチャ・ステートを保持するレジスタファイルに着目し、その書き込み時におけるタイミングエラーを検出、回復する手法を提案した。

タイミングエラーを検出するためには、一度書き込んだ内容をもう一度読み出して正しい書き込み値と比較する必要がある。そのため、本手法ではタイミングエラー耐性の高い小容量のバッファに書き込み値を保持してエラーの検出、回復に用いた。

SPEC2000ベンチマークでの評価の結果、このバッファが8エントリと小さくても、実行オーバヘッドは4.5%以内と小さく抑えられることがわかった。

謝辞 本論文の研究は，一部21世紀COE「情報技術戦略コア」、及び科学技術振興機構CREST「ディペンダブル情報基盤」による。

参考文献

1) Austin, T.M.: DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design.
2) D.Burger and T.M.Austin: *The SimpleScalar tool set, version 2.0*, Vol. 25, No. 3, ACM SIGARCH Computer Architecture News (1997).
3) D.M.Tullsen, S.J.Eggers and H.M.Levy: *Simultaneous multithreading: maximizing on-chip parallelism*, 22nd International Symposium on Computer Architecture (1995).
4) Ernst, D., Kim, N. S., Das, S., Pant, S., Rao, R., Pham, T., Ziesler, C., Blaauw, D., Austin, T., Flautner, K. and Mudge, T.: Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation, *International Symposium on Microarchitecture*, pp. 7–18 (2003).
5) L.Amghel and M.Nicolaidis: *Cost reduction and evaluation of a temporary faults detecting technique*, Design, Automation and Test in Europe Conference (2000).
6) Pering, T., Burd, T. and Brodersen, R.: *The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms*, Proceedings of International Symposium on Low Power Electronic and Design (1998).
7) Rotenberg, E.: *AR-SMT:A Microarchitectural Approach to Fault Tolerance in Microprocessors*, 32nd IEEE/ACM International Symposium on Microarchitecture(MICRO-32) (1999).
8) Sato, T.: A Transparent Transient Faults Tolerance Mechanism for Superscalar Processors, *IEICE Transaction on Information and Systems*, Vol. E86-D.
9) Vijaykumar, T. N., Pomeranz, I. and Cheng, K.: Transient-Fault Recovery Using Simultaneous Multithread ing, *International Symposium on Computer Architecture*, pp. 87–98 (2002).