社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－ARC－167（25）
2006－HPC－105（25）
2006／2／28

タイミング違反を利用した省電力プロセッサにおける履歴を用いた性能低下抑制手法

千代延 昭宏[1]　美馬 和大[2]　佐藤 寿倫[3]

[1] 九州工業大学大学院 情報工学研究科
[2] 日立超 LSI システムズ
[3] 九州大学 システム LSI 研究センター

E-mail: [1] chiyo@mickey.ai.kyutech.jp, [3] toshinori.sato@computer.org

近年，半導体の微細化技術の向上によりタイミング制約が重要視されている中で，高性能かつ低電力のプロセッサが求められている．我々は，タイミング違反を利用した低電力方式の提案をしてきた．本稿では，この建設的タイミング違反方式を適用することで被る，性能へのペナルティを削減する方式を検討している．

Mitigating Performance Penalty of Constructive Timing Violation Technique

Akihiro Chiyonobu[1]　Kazuhiro Mima[2]　Toshinori Sato[3]

[1] Department of Artificial Intelligence, Kyushu Institute of Technology
[2] Hitachi ULSI Systems Co.,Ltd.
[3] System LSI Research Center, Kyushu University

E-mail: [1] chiyo@mickey.ai.kyutech.jp, [3] toshinori.sato@computer.org

While high-performance and low-power processors are still required, the wire delay problem due to the progress of semiconductor process technology is becoming serious, which results in severe timing constraints. To attack the problem, we proposed a typical-case-design methodology, which utilizes circuit-level timing speculation. We called it Constructive Timing Violation (CTV) technique. Unfortunately, processors, which utilize the CTV, suffer performance penalty due to verifying the circuit-level speculation. In this paper, we are trying to mitigate the penalty.

1. はじめに

半導体の微細化技術は着実に発展している．微細化技術の発展は，家電製品をはじめとする製品の高性能化・省電力化・高機能化・小型化に大きく貢献している．微細化技術の進展によって受ける恩恵は，宇宙産業の分野ではさらに重要な位置づけになる．宇宙ステーションなどの宇宙産業の拠点では，エネルギーや空間の制約が地上より厳しいため，半導体技術を用いたシステムの省電力化と小型化が強く要求されることは容易に想像できる．したがって半導体の微細化技術は将来にわたって要求されることは必至であり，発展していくと考えられる．

しかしながら，微細化技術の発展により上記の恩恵が得られる一方で，いくつかの問題が生じてきた．LSI の性能を決定する遅延時間の主要因は，回路を構成する素子の遅延時間（ゲート遅延）から配線遅延に移行しつつある．微細化によりゲート遅延は着実に小さくなるが，配線幅の細線化と高機能化によるチップ面積の増加に起因する配線長の増加が原因で配線遅延は小さくなるどころか逆に大きくなりつつある．高機能化によりチップ面積が拡大すればするほど配線遅延は大きくなり，チップの動作速度を低下させる．つまり高機能化と高速化はトレードオフの関係となり，微細化が進むと高機能かつ高速なチップの開発コストが増加すると予想できる．一方，発熱により内部から発生するノイズや宇宙線などの外部から発生するノイズへの耐性が，回路素子の微細化によって弱くなる問題がある．ノイズの影響で，回路が想定している動作をしない可能性がある．特に地上とは異なり宇宙空間では大気がないため宇宙線の影響を受け易く，宇宙空間で使用されるシステムには外部ノイズを防ぐための対策が強く要求される。したがって，対策のためにシステムが大規模になる可能性がある．そこで LSI チップとは別にノイズ対策を行うのではなく，チップを構成する回路レベルやアーキテクチャレベルでノイズ対策を施すことにより，システムを大規模にすることなくノイズ耐性を可能にすることが期待される．

このような状況下で，LSI 主要製品であるマイクロプロセッサは，高性能計算機から PDA 等の携帯情報端末に至るまで高性能かつ低消費電力なものの必要性が

増している．高性能計算機用のマイクロプロセッサでは，演算性能を向上させるためには高クロック動作させるのが最も単純な方法であるが，消費電力の増大が問題となる．一方携帯情報端末では，Java や動画像処理アプリケーションを動作させることが当然のようになっており，それらを処理するマイクロプロセッサでも高い性能が要求されている．同時にバッテリなどの制約により低消費電力化も必要である．また PC に限らず携帯情報端末でも商取引が盛んに行われるようになっており，過酷な環境で使用されることが多い携帯情報端末は商取引中に誤動作しないように信頼性が求められている．

我々は，高性能・省電力・高信頼なマイクロプロセッサを実現するための技術として建設的タイミング違反方式（Constructive Timing Violation: CTV)を提案している[2]. マイクロプロセッサなどの LSI を正常に動作させるための設計制約（動作周波数，電源電圧等）を寛容に見積もることで起こる故障状態を，フォールトトレランス機構を備えることで動作を保証する方式である．本方式は，故障を検出するとフォールトトレランス機構が故障回復を行うが，故障確率が高いほど性能が低下する．本稿では，履歴を用いることで,マイクロプロセッサに CTV を適用する際に被る性能低下を抑制する方法を提案する．

2. 建設的タイミング違反方式

ディープサブミクロン化により，従来の保守的な設計手法である最悪ケースを考慮した方法では，近い将来 LSI の設計は不可能になると予想されている．ディープサブミクロン化したプロセス技術では，ノイズやプロセスばらつきが増大している．加えて電源電圧を下げる必要があり，さらにノイズ耐性の悪化に拍車をかけている．このような条件化では，最悪ケースを考慮するための設計マージンを確保することが非常に困難である．以上を踏まえ我々は，LSI の設計制約を楽観的に考えてタイミング違反による動作異常の状態を許容し，そのかわりにタイミング違反に対するフォールトトレランス機構を備えることを提案している．これが CTV と呼んでいる手法である[2].

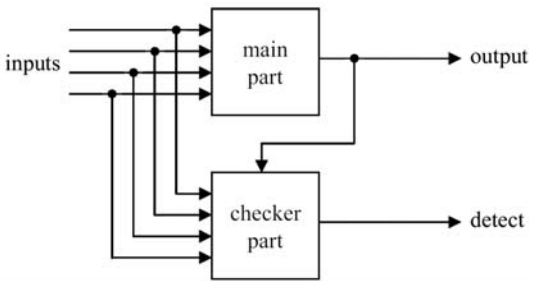

図 1：建設的タイミング違反方式

CTV の基本的な考え方を，図 1 で説明する．CTV では，設計時に検出されたタイミング違反が実行時には発生しないと仮定している．回路レベルで投機的実行を行っており，タイミング違反を検出する機構と違反検出時にプロセッサの状態を正常に回復させる機構とが必要となる．図 1 では回復機構は省略されている．図に示されている通り，CTV 方式で設計された回路はメイン部（図の main part）とチェック部（図の checker part）から構成される．メイン部は元の回路であり，低レイテンシかつ高スループットという高性能を維持できるように設計されるが，タイミング違反が発生する可能性を秘めている．チェック部はタイミング違反を生じないように設計されており，メイン部の入出力が入力されメイン部のタイミング違反を検出する．

3. 性能低下の抑制手法

3.1. 故障発生の特徴

信号遅延で生じる遅延故障は，実行される命令や CTV を適用した回路の特性によって遅延故障が生じる命令や演算に偏りがあると予想される．この偏りを把握し利用することで，遅延故障があらかじめ起こることを予測し遅延故障を起こさない対策をとることで,故障回復のペナルティを被ることを避ける．偏りを生じる要因となる可能性について，ALU 内の加算器に絞って検討した結果を以下に示す.

1. 同じ命令が何度も故障を起こす可能性が高い.
2. 同じ演算が何度も故障を起こす可能性が高い.
3. 加算器は演算ビット幅が大きくなるにしたがって回路のクリティカルパスが長くなる．したがって，オペランドのビット幅が小さな演算は遅延故障を発生しない可能性が高い.
4. 演算幅が大きくても，桁上げ伝播が途中で絶たれる場合には上位の桁と下位の桁は独立した演算となるので，遅延故障が発生しない.
5. 連続する演算が同じあるいは酷似する場合には，遅延故障を生じる演算であっても，先行する演算結果を利用できるために遅延故障を生じない.

要因 3 と 4 について評価した結果を示す．加算器には 32 ビット桁上げ選択加算器[3]を用いた. まず要因 3 を評価するために，故障を起こした演算の有効ビット幅を調査した．図 2 に故障を起こした演算の有効ビットが 16 ビット以下の割合を示す. 横軸はベンチマークを示しており，ベンチマークごとに 4 本のグラフがある．左から加算,減算，ロード，ストア命令中で行われる加減算で, 故障を生じた演算が有効ビット 16 ビット以下の割合を示す．各演算での遅延がクリティカルパス遅延の(1/1.5)倍を越える時に，故障を生じるとした.

ストア命令に関しては，ベンチマークプログラムに関係なく 16 ビット以下の割合が小さい．これはストア命令がアドレス計算で大きな値を計算することが多いからである．その他の命令に関しては 176.gcc を除いて 15%以下となっている．概して，有効ビット幅が小さいからと言って故障が生じないとは限らない．故障する演算の有効ビット幅には強い偏りは無いと言える．

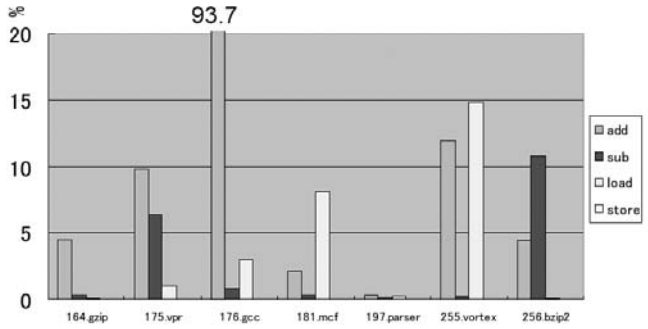

図 2：有効ビット数が 16 ビット以下の割合

続いて要因 4 を評価するために，故障を起こした演算でのキャリー伝播段数を調査した．これが大きいほどキャリー伝播に依存して演算が行われていることになる．もしこの値が 0 であれば演算の各桁はまったく前段からのキャリーの影響を受けず，各桁は独立して計算することが可能である．図 3 にキャリー伝播段数の分布を示す．伝播数が 0 から 4 までの割合を示している．最大 4 桁までの理由は，評価した桁上げ選択加算器[3]が 4 ビットのブロックで構成されているためである．結果として特に強い偏りは無いと言える．

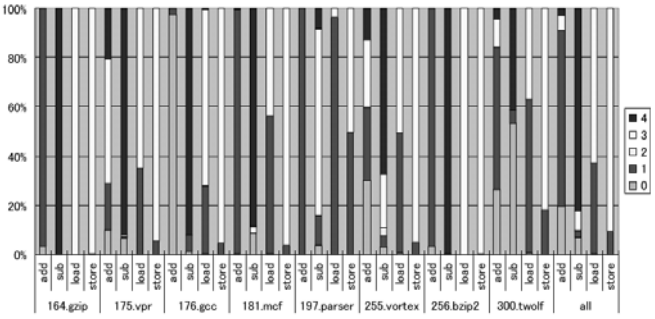

図 3：キャリー伝播段数

要因 5 の確認は今後の課題とする．以降では，要因 1 と 2 を利用した方式を検討する．

3.2. 演算結果キャッシュ

演算結果キャッシュ(Result cache: RC)は過去に行った演算を保持している[1]．図 4 に RC の構成を示す．キャッシュアクセスには演算オペランドをハッシュ関数にかけた値を用いる．ハッシュ値の下位ビットをインデクスとしキャッシュラインにアクセスする．RC のタグにはハッシュ値の上位ビットが格納されている．したがって，ハッシュ値の上位ビットとタグを比較して等しければキャッシュヒットとなりキャッシュに格納されている値を出力する．等しくなければミスヒットとなる．

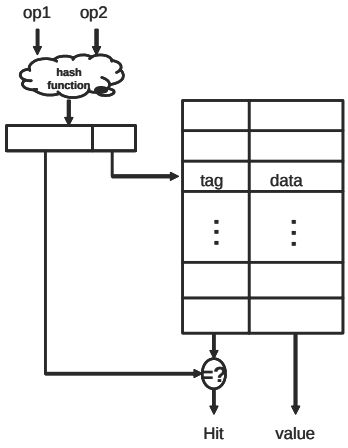

図 4：演算結果キャッシュ

キャッシュアクセスはメイン部 ALU の演算と同時に行われる．RC にヒットした場合，演算結果は RC から獲得された値を採用する．一方，チェック部は RC のヒット／ミスヒットにかかわらず故障検出を行う．キャッシュにヒットした場合，故障が起こるかの検証を行っても故障は検出されない．キャッシュにヒットした演算の検証は無駄のように思える．しかし RC にヒットするか否かで故障検出を行うか否かを決めるのは故障検出機構のパイプラインスケジューリングを複雑にし，故障検出機構の動作速度を低下させてしまう恐れがある．したがって故障の検証を行う必要がない場合でも検証を行うようにする．

本手法で利用する RC は故障を回避する目的で利用する．したがって，キャッシュには過去に故障を起こした演算の正しい結果のみを保持する．もしキャッシュに存在する演算を実行するならば，RC 内の値を実行結果として用いる．これにより正しい結果を得られるため故障を回避できる．ただし性能低下を避けるためには，演算器と同じレイテンシでアクセスできる必要がある．

3.3. 故障履歴バッファ

故障履歴バッファ(Fault History Buffer: FHB)は過去に故障を起こした命令のプログラムカウンタ（PC）を保持するバッファである．図 5 に構成を示す．PC の下位ビットをインデックスとしてバッファにアクセスする．FHB のタグには PC の上位ビットが格納されている．したがって PC の上位ビットとタグを比較して等しければキャッシュヒットとなり，等しくなければミ

スヒットとなる．ミスヒットの場合は通常の実行を行い故障検出機構で検証を行う．一方 FHB 内に実行される命令の PC が存在する場合には，故障を起こす命令と判断して，安全で低速な故障検出機構に搭載されている ALU で実行する．これにより演算レイテンシは増加するが，故障を回避することができる．FHB へのエントリ追加は故障検出機構で故障が検出された場合に行う．

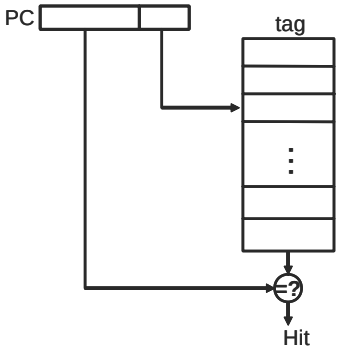

図 5：故障履歴バッファ

4. 評価方法

4.1. プロセッサモデル

Alpha 命令セットシミュレータを構築した．基準モデルは表 1 のとおりで，アウトオブオーダー実行を行うスーパスカラプロセッサである．

RC は CTV を適用した ALU と同速度で動作しなければならないので，大容量のものを用いることはできない．したがって，RC は本研究では 64 エントリまでとする．FHTB は BTB に構造が似ているためエントリ数を最大 1024 までとする．

故障検出の対象は，整数 ALU で実行される加算命令と減算命令とする．故障回復には故障した命令以降の命令を全て破棄し再実行させる命令破棄を利用する．

表 1：プロセッサモデル

命令フェッチ幅	4 命令
分岐予測機構	bimodal,1024 エントリダイレクトマップ BTB, コミット時更新 リターンアドレス・スタック 8 エントリ, ミスペナルティ 3 サイクル
命令ウィンドウ	リザベーションステーション 16 エントリ リオーダバッファ 16 エントリ, ロード・ストアキュー 8 エントリ
命令ディスパッチ幅	4 命令
命令コミット幅	4 命令
機能ユニット数	4iALU's,1iMUL/DIV,2Ld/St,4fpALU's,2fpMUL/DIVs
レイテンシ (全/投入間隔)	iALU 1/1,iMUL 7/1,iDIV 12/9,Ld/St 1/1,fADD 4/1 fCMP 4/1,fCVT 3/1,fMUL 4/1,fDIV 12/9,fSQRT 18/15
レジスタファイル	32 ビット整数レジスタ 32 本，32 ビット浮動小数点レジスタ 32 本
データキャッシュ	16K4 ウェイ・セットアソシアティブ，ライン幅 32 バイト
命令キャッシュ	16K ダイレクトマップ，ライン幅 32 バイト
2 次キャッシュ	共用，256K4 ウェイ・セットアソシアティブ，ライン幅 64 バイト

4.2. ベンチマークプログラム

評価には SPEC CINT2000 から表に示す 7 個を用いた．各プログラムの入力には SPEC から配布されている ref ファイルを使用した．各プログラムのシミュレーションは 10 億命令実行後の 1 億命令を対象として行った．

5. 結果

5.1. 故障検出機構におけるレイテンシ増の影響

CTV を適用していない従来の方法と CTV を適用したプロセッサの性能を評価する．評価の指標には，サイクル当りにコミットされた命令数(Instructions Per Cycle: IPC)を用いる．後者の場合，故障検出機構のレイテンシは 2〜4 までを考慮する．図 6 に故障検出機構のレイテンシを変化させたときの IPC を示す．この評価では故障率は 0%である．つまり故障回復による性能低下は含まれていない．

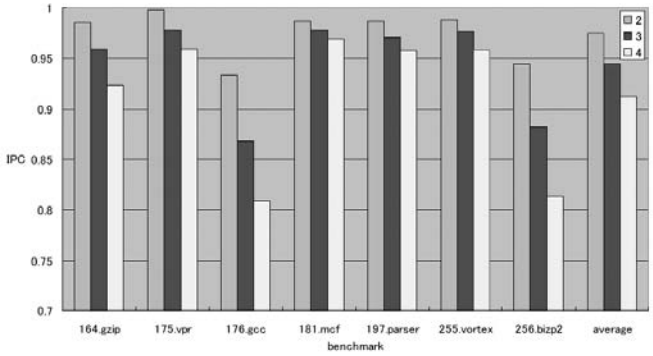

図 6：検出レイテンシの性能への影響

グラフの値は，CTV を適用しない基準モデルの IPC を 1 として正規化している．故障検出機構のレイテンシが 2 の場合には，性能が平均で 2.5%，最大 6.6%（176.gcc）低下している．故障検出機構のレイテンシが 4 の場合には，性能が平均で 9.6%，最大 19.0%（176.gcc）低下する．これは，故障を生じなくても，CTV を適用しただけで性能が低下することを表している．この原因は，故障検出機構が検証を終えるまで命令がコミットされないためである．命令がコミットされないとリオーダバッファのエントリが開放されないためにリソース不足となり，リソースが確保できるまでプロセッサが停止してしまう．故障検出機構のレイテンシが増加すると命令がコミットされるまでのレイテンシが増加しリソース不足が頻繁になる．

5.2. 演算結果キャッシュの効果

RC を適用すると性能低下をどの程度抑制できるかを調べるために，RC のエントリ数を 2 から 64 まで変化させて IPC を測定した．故障はランダムで与えられ，故障率を 30%とした．図 7 に結果を示す．横軸はベンチマーク名である．縦軸は，基準モデルの IPC を 1 とした時の，RC を適用したモデル（RC モデル）の IPC である．ベンチマークごとに 6 本のグラフがある．左端は CTV のみを適用したモデル（CTV モデル）の，残る 5 本は CTV に RC モデルの性能である．この評価で用いた RC はフルアソシアティブであり,全ての整数

演算器で共有されている．書き込み時にエントリに空きがない場合は，FIFO 方式で入れ替える．

2 エントリの場合には，最大で 3.4%（255.vortex），平均で 1.8%の性能低下抑制を達成できている．64 エントリ時には，最大で 13.5%（255.vortex），平均で 7.0%の性能低下抑制が達成できている．平均を見ると，RC のエントリ数増加におよそ比例して，性能低下抑制の効果も増加している．

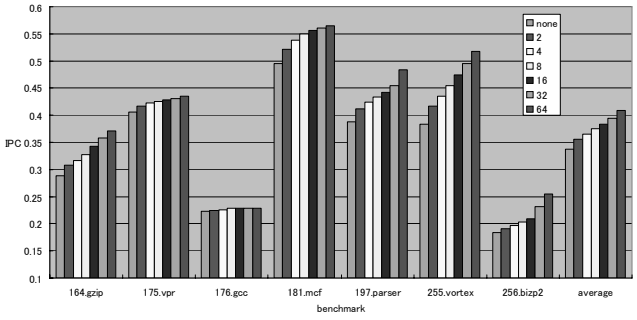

図 7：演算結果キャッシュの効果

今回の評価では故障はランダムに発生しているため，故障を発生させる演算に偏りはない．それにも関わらず，RC のエントリ数を増加させた時に，性能低下が抑制できるものとそうではないものが見られる．前者の例は 197.parser と 255.vortex であり，後者の例が 175.vpr と 176.gcc である．前者が現れる理由は，故障を生じる／生じないに関係なく，同じ演算が繰り返し実行されるためと考えられる．

5.3. 故障履歴バッファの効果

FHB の効果を調べるために，FHB のエントリ数を 2 から 1024 まで変化させたときの IPC を測定した．前節と同様に，故障はランダムで与えられ，故障率を 30%とした．図 8 に結果を示す．縦軸は，基準モデルの IPC を 1 とした時の，FHB を適用したモデル（FHB モデル）の IPC である．ベンチマークごとに 11 本のグラフがある．左端のグラフは CTV モデルの，残る 10 本は FHB モデルの性能である．

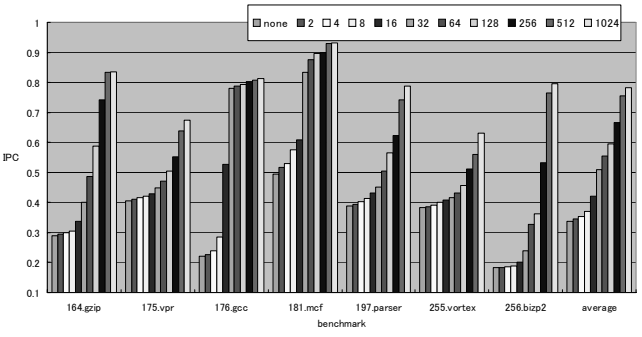

図 8：故障履歴バッファの効果

平均を見ると，FHB のエントリ数が 2 から 8 の間では性能はほとんど改善されない．8 エントリの時で 3.2%の改善である．エントリ数が 16 から 512 に増加すると，エントリ数増加にみあった効果が確認できる．512 エントリの時には 41.6%の改善が達成され，基準モデルの 75.4%までに性能が回復する．さらにエントリ数を 1024 に増やしても，512 の場合と比較して 2.8%しか改善せず，エントリ数増加による性能改善はあまり見られなかった．

164.gzip，176.gcc と 256.bzip2 では FHB のエントリ数を増加させたときの性能の改善が著しい．例えば 512 エントリでは 176.gcc で最も効果が高く，58.6%の改善率である．これらのベンチマークプログラムは，CTV モデルでの性能低下が著しく，かつ RC を適用した場合では効果を得られなかった．しかし，FHB を適用した場合は，大きな効果が確認できる．これらの 3 個のベンチマークプログラムでは実行される命令の時間的局所性が大きく，FHB のヒット率が大きいことが要因である．一方 197.parser と 255.vortex では，RC の効果は大きいが，FHB の効果は小さい．これらの 2 個のベンチマークプログラムは上記 3 個のベンチマークプログラムと異なり，実行される命令の時間的局所性が小さく，FHB のヒット率が小さいことが要因である．

5.4. 二方式を組み合わせた効果

前節までの評価では，それぞれの方式で効果的に性能低下を抑制できるプログラムとそうでないものが観察された．したがって，RC と FHB を組み合わせることでお互いの手法の弱点を補うことを考える．この評価では，各々の手法において，エントリあたりの改善効果が高いエントリ数を採用する．つまり，RC では 64 エントリ，FHB では 512 エントリとする．

RC と FHB へのアクセス方法は以下のとおりである．まず RC へアクセスし，ヒットしない場合のみ FHB へアクセスする．RC ヒット時には，RC から獲得された値を演算結果とする．FHB ヒット時には，故障検出機構で演算を行った結果を採用する．RC にヒットしない場合には，メイン部の演算結果を故障検出機構で検証する．この時に故障が検出されれば，RC と FHB の両方にエントリを追加する．

図 9 に結果を示す．このときの故障率は前節までの評価と同様に 30%である．ベンチマークごとに 4 本のグラフがある．左から順に，CTV モデル，RC モデル，FHB モデル，二方式を組み合わせたモデル（both モデル）の性能である．基準モデルの IPC を 1 とした時の，各モデルの IPC で表している．Both モデルの効果は，RC モデルの性能改善率と FHB モデルの性能改善率を足した結果となっているのが理想である．197.parser

と 255.vortex では，それぞれの改善率をほぼ足した結果となっており，両方式を組み合わせることでシナジー効果が得られることが確認できる．一方 164.gzip, 181.mcf と 256.bzip2 では，二方式を組み合わせることによるシナジー効果は確認できない．しかし平均では，both モデルの性能改善率は，RC モデルの改善率と FHB モデルの改善率をほぼ足した値となっている．このときの性能は基準モデルの 78.9%までに回復できている．

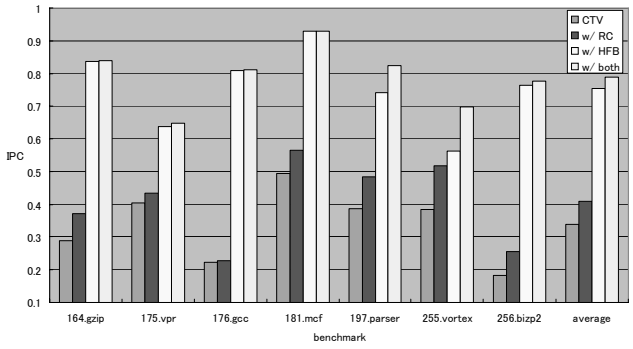

図 9：二方式を組み合わせた効果

表 2 に，各モデルでの，RC と FHB のそれぞれのヒット率を示している．括弧内の数字はエントリ数である．二方式を同時を適用した場合を各々の方式を個別に適用した場合を比較すると，FHB のヒット率はほとんど変わらないことが判る．しかし，RC のヒット率は，175.vpr を除いて改善している．RC は 1 サイクルで正しい演算結果を出力できるため，RC のヒット率が上昇すれば性能が改善することが期待できる．しかし，いくつかのベンチマークで二方式を組み合わせたときに想定する性能の改善ができてない．その理由は，RC にヒットする演算数が少なくなっているためであると考えられる．

表 2：ヒット率

program	RC	FHB	both	
	(64)	(512)	RC(64)	FHB(512)
164.gzip	32.54	59.23	98.73	55.54
175.vpr	11.27	37.96	7.70	38.01
176.cc1	3.45	92.26	35.79	92.36
181.mcf	33.83	62.25	39.30	62.31
197.parser	34.46	53.88	60.16	54.15
255.vortex	38.89	32.43	48.52	32.12
256.bzip2	38.46	67.01	90.83	65.25

5.5. 故障率と性能低下抑制の関係

図 10 は，CTV モデル，RC モデル，FHB モデル，both モデルの性能が，故障率にどのような影響を被るかを示している．横軸は故障率である．縦軸は，基準モデルの IPC を 1 とした時の，各モデルの IPC である．グラフは 7 個のベンチマークの平均値である．この評価に用いる RC と FHB のエントリ数は前節で用いた値と同じである．

故障率が上昇すると，CTV モデルでは性能が著しく低下するが，FHB モデルと both モデルでは，故障率が 100%の場合でも高々35%の性能低下である．RC モデルではエントリ数が 64 と小さいため，CTV モデルと比較して約 7%しか性能を改善できない．both モデルでは，故障率 30%以上では，二つのモデルの改善効果を足した改善効果が確認できる．

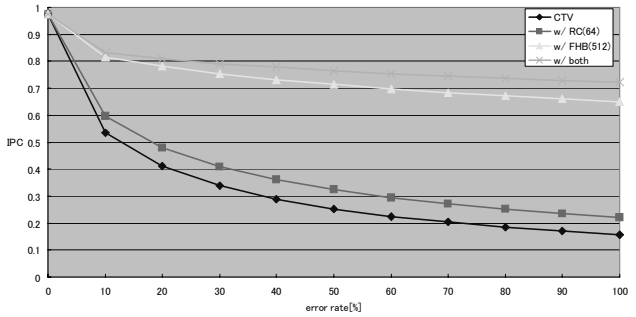

図 10：故障率の性能への影響

6. まとめ

CTV を適用したマイクロプロセッサにおいて，故障検出のために生じる性能低下を抑制する手法の提案した．シミュレータを用いて評価した結果，30%の故障率を想定した場合，64 エントリの RC を用いた場合で平均 7%の，512 エントリの FHB を用いた場合で平均 41.6%の，64 エントリの RC と 512 エントリの FHB を組み合わせて用いた場合で平均 45.1%の性能低下を抑制できたことが確認できた．また故障率 100%の場合において，64 エントリの RC および 512 エントリの FHB を用いれば，性能低下を平均 27.8%に抑えることができることが確認された．

性能低下を抑制した時の消費電力への影響を調査することが，今後の課題である．

謝辞

本研究の一部は，科学研究費補助金（No.16300019, No.176549）の援助によるものです．

文　献

[1] S. E. Richardson: Exploiting trivial and redundant computation, 11 th Int. Symp. on Computer Arithmetic (June 1993)

[2] T. Sato, I. Arita: Potential of constructive timing-violation, IEICE Trans. Electron., vol.E85-C, no.2 (February 2002)

[3] 谷野，佐藤，有田: 建設的タイミング違反方式に基づく ALU の HDL 設計とその評価，信学技報 ICD2002-212 (March 2003)