高信頼マイクロプロセッサ・アーキテクチャ

井上, 弘士
九州大学大学院システム情報科学研究院

出版情報：日本信頼性学会誌 : 信頼性. 30 (1), pp.27-35, 2008-01. 日本信頼性学会
バージョン：

高信頼マイクロプロセッサ・アーキテクチャ
Reliable Microprocessor Architectures

井上 弘士
Koji INOUE

概 要

近年，コンピュータの頭脳であるマイクロプロセッサの信頼性低下が極めて深刻な問題として注目されている．微細加工技術の進歩に伴い劇的な性能向上を達成してきた反面，耐故障性の低下により外部/内部雑音などの影響を受け易くなった．その結果，システムには不具合がなくとも，コンピュータが正しくプログラムを実行できないという極めて深刻な事態となる．このような背景に基づき，近年，マイクロプロセッサの信頼性向上を目的とした様々なアーキテクチャ技術が提案された．本稿では，信頼性向上戦略を整理すると共に，商用マイクロプロセッサの動向も踏まえてアーキテクチャ・レベルでの信頼性向上技術を解説する．

1. はじめに

順調に成長を続ける半導体微細化技術の進歩を背景に，コンピュータ・システムの頭脳であるマイクロプロセッサは目覚ましい発展を遂げてきた．LSI上に形成されるトランジスタを小型化することで，回路規模の拡大による高機能化，ならびに，高速スイッチングによる動作周波数の向上が可能となった．1985年から2000年前半においてインテル社が発表した代表的なマイクロプロセッサのトランジスタ数を図1に示す．例えば，1971年に開発されたインテル社の4ビット・マイクロプロセッサ4004では，動作周波数が108KHz，使用トランジスタ数は2,300個程度であった（最小加工寸法は10μm）．これに対し，2000年頃に登場したPentium4では，動作周波数が1.5GHz，使用トランジスタ数は約4,000万個にも達する（最小加工寸法は0.18μm）．また，テストチップではあるものの，2007年に

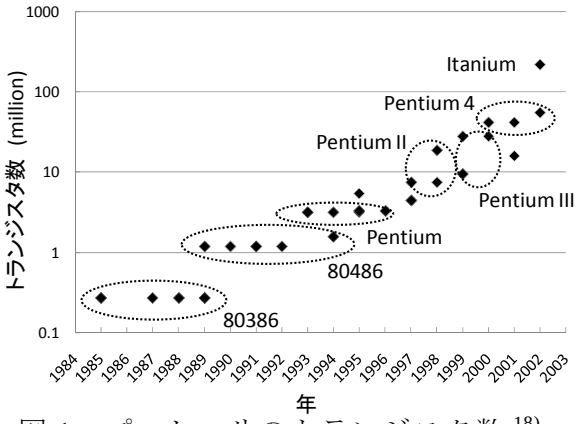

図 1：プロセッサのトランジスタ数 [18)]

発表されたインテル社の80コア・プロセッサでは約1億個のトランジスタを集積しており，最大動作周波数は5GHzを超える[17)]．

コンピュータ・システムにおけるマイクロプロセッサの役割は，「正しくプログラム（ソフトウェア）を実行する」ことにある．この大前提を満足するという制約の下，マイクロプロセッサはトランジスタ集積度の向上と共に進化してきた．しかしながら，近年，極めて深刻な問題が顕在化してきている．信頼性の低下である．

ハードウェア設計に不具合がなくとも，様々な要因により「正しくプログラムを実行する」ことができない状況が発生している．その代表的な原因として，LSI内部における一時的な故障の発生が挙げられる．予期しない不都合な物理的現象によりプロセッサが誤動作し,引いては,誤った実行結果を出力することになりかねない.以前よりLSIの故障は問題視されていた．しかしながら,近年の超微細化やクロックの高速化,低電源電圧化などに伴い十分な設計マージンを確保することが難しくなり，故障発生確率が高くなってきているのが現状である．

マイクロプロセッサの信頼性を高めるには,1)デバイス・レベルにて耐故障性に優れたトランジスタを形成する，2)回路レベルにて耐故障性を考慮した設計を行う，3)アーキテクチャ・レベルで故障の検出とプログラム実行の回復を可能にする，などが挙げられる．本稿では，特にアーキテクチャ・レベルに焦点を当て，商用マイクロプロセッサの動向も踏まえて高信頼化技術を解説する．

2. マイクロプロセッサの信頼性

2. 1. LSI での故障

LSI内部で発生する故障は，主に以下の2種類に大別される．

- **永久故障**：長期間に渡って存在する故障．発生要因としては，半導体接合の破壊や回路短絡，断線などが挙げられる．
- **過渡故障**：システム稼動サイクルと比較して非常に短い期間に存在する故障．α粒子や宇宙線などの外部的要因，さらには，電源電圧の変動といった内部的要因などにより一時的に回路の誤動作が発生する．

ここで，「信頼できるマイクロプロセッサ」とは，LSI内部で永久故障や過渡故障が発生した場合でもプログラム実行能力（機能）を維持できることとする．したがって，プロセッサの信頼性を向上するためには，以下の2つの機能が必要となる．

- **故障検出能力**：プログラム実行に悪影響を及ぼす故障を検出する．
- **実行回復能力**：故障によって引き起こされた実行結果の誤りを訂正し，正しくプログラム実行を継続する．

すなわち，プログラム実行中，1）永久故障や過渡故障の検出，2）その悪影響の排除，ならびに，3）プログラムの再実行をサポートする必要がある．

一般に，永久故障は再現性が高いため，プログラム実行前の診断などにより検出できる．これに対し，過渡故障は再現性がないため，プログラム実行中でのオンライン診断が必要となり,様々な性能/面積オーバヘッドを伴う．以降，本稿では,過渡故障の検出/回復を可能にする各種アーキテクチャ技術に焦点を当てる．

2. 2. 高信頼化の基本方針

第1節で説明したように，マイクロプロセッサが満たすべき機能制約は「正しくプログラムを実行する」ことにある．つまり,

- 命令を正しい順序で実行する（分岐などの制御命令を正しく実行する）
- 命令実行で必要となる各種演算を正しく行う（指定されたオペランドに対して正しい演算を行う）
- データを正しく転送/記憶する（レジスタ・ファイルやメモリなどに正しくデータを保持する）

ことにより，プログラム実行結果に影響を与える全ての命令実行の正しさを保証する必要がある．これを実現する代表的な手段として以下の

方法が挙げられる．

- **情報冗長化**：誤り検出/訂正符号により命令実行に必要となる各種データを保護する．現在多くの商用プロセッサにおいてパリティ符号やECC（Error Correcting Code）が採用されている．
- **空間冗長化**：ハードウェア要素を複数搭載し，これらで同一命令を並列実行する．実行結果を比較することで故障を検出できる．多重化されたハードウェアからの出力結果に関して多数決を採ることにより実行回復が可能となる．
- **時間冗長化**：単一ハードウェアにおいて同一命令を複数回実行する．実行結果を比較することで故障を検出できる．多重化された実行結果に関して多数決などを採ることにより実行回復も可能となる．ただし，単一ハードウェアを使用する場合は永久故障の検出は困難である．

2．3．オーバヘッド削減の重要性

マイクロプロセッサの信頼性を表す指標として，MITF（Mean Instructions To Failure）がある[14]．これは，システム信頼性の指標として広く認知されているMTTF（Mean Time To Failure）をマイクロプロセッサでのプログラム実行という観点から見直したものである．つまり，MITFは，障害発生間隔において処理可能な有効命令数を表しており，以下の式で定義される．

$$\mathrm{MITF} = \mathrm{IPC} \times \mathrm{MTTF} \times \mathrm{F} = \frac{\mathrm{IPC} \times \mathrm{F}}{\mathrm{AVF} \times \mathrm{SER}} \qquad (1)$$

ここで，IPCはクロック・サイクル当りの平均実行命令数（コミットされた命令の数），Fは動作周波数，SERはプロセッサにおけるソフトエラー発生確率，ならびに，AVF（Architectural Vulnerability Factor）はプロセッサ内部で発生した故障が実行結果に伝搬する確率を表す（詳細は文献[14]を参照）．ここで，式（1）において以下のトレードオフが存在する．

- **AVF vs. IPCとF**：第2.2節で説明したように，一般的には冗長化によって命令実行の信頼性を向上できる（つまり，AVFを小さくできる）．しかしながら，冗長化によるオーバヘッドが顕在化する場合には，必ずしもMITFを改善できるとは限らない．例えば，時間冗長化を施した際にはIPCが低下する傾向にある．また，空間冗長化や情報冗長化により追加した回路がマイクロプロセッサの動作周波数Fに悪影響を及ぼすことも考えられる．
- **AVF vs. SER**：前述したように，空間冗長化や情報冗長化の実施はマイクロプロセッサの回路規模を増大する傾向にある．その結果，使用するトランジスタ数が増加し，引いては，チップ当りのソフトエラー発生確率が高くなる．

第2.2節で説明したように，マイクロプロセッサの耐故障性を向上するには様々な冗長化が必要となる．実際には，複数の冗長化技術を組合せることにより，命令実行の順序関係と結果，ならびに，記憶データを保護し，故障検出と実行回復を可能にする．しかしながら，上述したように，冗長化を施した場合には様々なオーバヘッドが顕在化する．したがって，マイクロプロセッサにおいてMITFを改善するためには，「**如何に性能/面積オーバヘッドを抑制しつつ，AVFを改善できるか？**」が極めて重要となる．

3．情報冗長化による高信頼化

データ通信における信頼性向上技術として，古くから誤り検出/訂正符号が用いられてきた．マイクロプロセッサにおいては，レジスタ・ファイルやキャッシュ・メモリといった記憶素子の信頼性向上にこのような符号化技術が使用される．また，最近では，マイクロプロセッサのデー

	Power6	SPARC64V
Registers	Parity	Parity
L1 Instruction Cache	Parity	Parity
L1 Data Cache	Parity	SECDED
L2 Cache	SECDED	SECDED

図 2：商用プロセッサでの情報冗長化

タパスにおけるデータ通信（例えば，命令パイプライン中を流れる各データの転送）での故障検出にもパリティ符号が利用されるようになった．Power6ならびにSPARC64マイクロプロセッサで採用されている情報冗長化を図2に示す[2), 7), 8)]．Power6ではライト・スルー方式を採用しているため, L1キャッシュには常にクリーンなデータが滞在する．よって，L1キャッシュにおいてパリティ・エラーが検出された際にはL2キャッシュより当該データを再ロードすることで回復できる．これに対し，SPARC64ではL1データキャッシュにてSECDED（Single Error Correction Double Error Detection）コードを採用しており，L1データキャッシュ中のダーティ・データに1ビットの故障が生じた場合でも対応できる．一方，命令パイプライン中でパリティ・エラーが検出された場合には，プログラム実行を回復するため，故障を発生した命令を含むそれ以降の全命令の無効化（いわゆるパイプライン・フラッシュ），ならびに，当該命令からのプログラム再実行を実施する．

4. 空間冗長化による高信頼化

4.1. 命令パイプラインの多重化

空間冗長化に基づく代表的な高信頼プロセッサとしてDIVA（Dynamic Implementation Verification Architecture）が提案されている[3)]．その主な特徴は，同一機能で異なる回路構成の命令実行ユニットを2つ搭載し，それぞれの実行結果を比較することで故障を動的に検出する点にある．

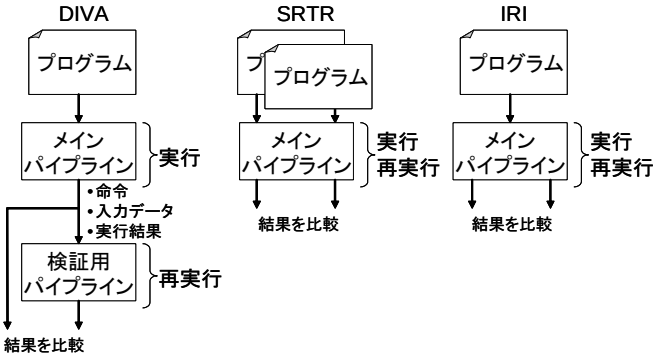

図 3：プログラム実行の多重化

DIVAでは, 図3に示すように, 通常のメイン・パイプラインに加え，DIVAチェッカと呼ばれる検証用パイプラインが直列に接続される．このDIVAチェッカは構成が極めて単純であり，設計マージンを大きくとる等の対策により故障が発生しないという前提に立っている．従来型のプロセッサでは，パイプラインの最終ステージにおいて，命令の実行結果がプロセッサ状態に反映される（命令の実行が完了する）．これに対し，DIVAでは，一度メイン・パイプラインで実行された命令がDIVAチェッカによって再実行される. そして, これらの結果を比較し，一致していれば当該命令の実行を完了する．一方，比較結果が不一致の場合，これはメイン・パイプラインにて何らかの故障が発生したことを意味する．したがって，故障が発生した命令の正しい実行結果を求め，故障発生命令の後続命令を再実行する必要がある．DIVAでは，故障が検出された場合，メイン・パイプラインに滞在する全ての命令を無効化する. そして, DIVAチェッカより得た正しい実行結果を用いて，故障発生命令からプログラムの実行を再開する．このように，故障の検出から回復までをハードウェアのみでサポートしている．

なお，DIVAチェッカにおいて故障が発生しないという前提は，メイン・パイプラインにおける設計誤りや永久故障からの回復も可能にするためであり，過渡故障のみを対象とする場合には必ずしも必要ではない．つまり，第5節で

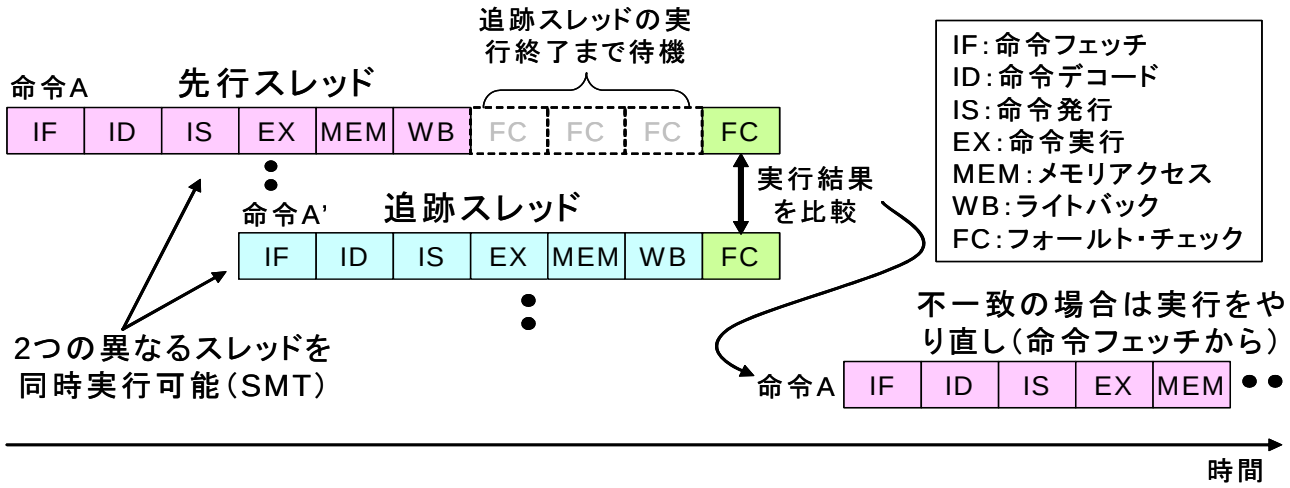

図 4：SRTR でのスレッド実行と故障検出

説明するSRTRと同様に，メイン・パイプラインとDIVAチェッカでの実行結果を比較し，不一致であれば当該命令以降を再実行すればよい．この場合，メイン・パイプラインまたはDIVAチェッカのどちらで過渡故障が発生したとしても，それを検出して回復することができる（ただし，メイン・パイプラインとDIVAチェッカの両方で故障が発生し，同じ実行結果を出力した場合には検出できない）．

4.2. キャッシュ・メモリの多重化

多くの商用マイクロプロセッサでは16～32KBのL1キャッシュ，また，数MBのL2キャッシュを搭載する場合がる．第3節で説明した情報冗長化を適用する場合，基本的にはこれら全てのメモリ領域に対して誤り検出/訂正符号を適用しなければならない．一方，キャッシュ・メモリはチップ面積の多くを占めるため，単純な空間冗長化では面積オーバヘッドが極めて大きくなる．そこで，キャッシュ・メモリが潜在的に有する空間冗長性を活用する方法が提案されている．

多くのプログラム実行において，常にキャッシュ・メモリの全領域が使用される訳ではない（多くの場合で10～30%程度）[6]．そこで，このような特徴を利用し，キャッシュ・メモリの未使用領域に記憶データのコピーを保存する[15]．キャッシュ・メモリからデータを読出す際，対応するコピーも同時に読出し，値を比較することで故障を検出できる．このように，「プログラム実行において使用されないハードウェア」を有効活用することで空間冗長化を実現し，信頼性向上に伴うオーバヘッドを抑制する．

5. 時間冗長化による高信頼化

5.1. スレッド・レベル多重実行

時間冗長化による高信頼プロセッサの特徴は，同一実行ユニットにて各命令を複数回実行し，それぞれの結果を比較することで故障を動的に検出する点にある（多くの場合は2回実行）．代表的な時間冗長化プロセッサとして，SRTR（Simultaneously and Redundantly Threaded processors with Recovery）がある[12]．SRTRでは，図3ならびに図4に示すように，プログラムから先行スレッド（leading thread）とその複製である追跡スレッド（trailing thread）を生成し，これらをある程度の時間差を持って実行する．基

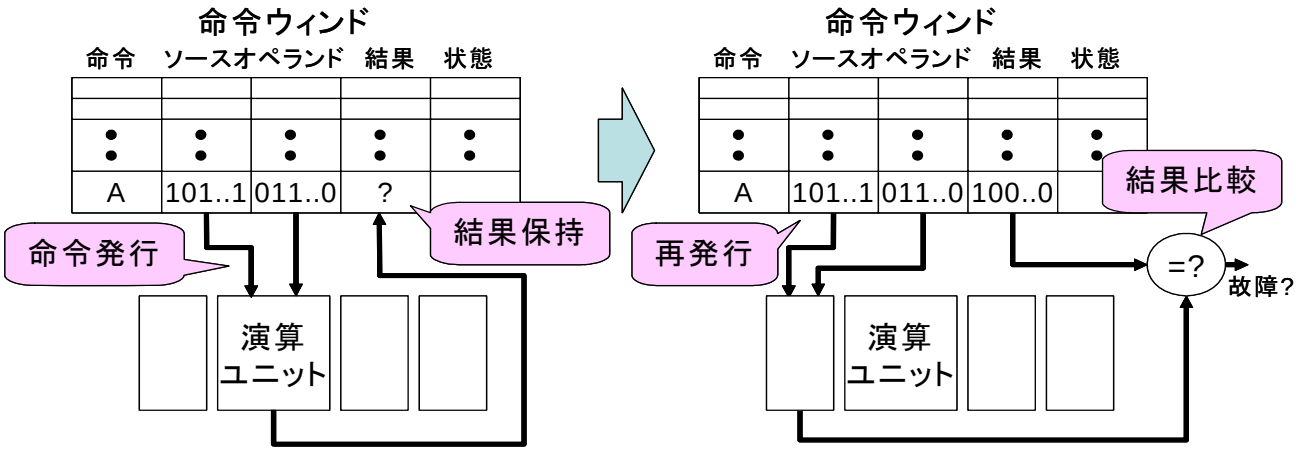

図 5：IRI での命令再発行と故障検出

本的には先行スレッドの命令がその実行を先に終える．ただし，実行結果が直ちにプロセッサ状態へ反映される訳ではない．先行スレッドの命令は，フォールトチェック・ステージにおいて，追跡スレッド内の対応する命令実行が終了するのを待つ.そして,これらの結果を比較し，一致していれば実行結果をプロセッサ状態へと反映する．一方，比較結果が不一致の場合，これは先行スレッドもしくは追跡スレッド何れか一方の実行において故障が発生したことを意味する．そこで，故障が検出された命令とその後続命令を無効化し，再度実行をやり直す．現在の高性能プロセッサは様々な投機実行をサポートしており，誤った命令実行パスから回復するためのハードウェア機構を備えている(例えば，分岐予測ミスからの復帰）．これを利用することで，特別な回路の追加なしに故障からの回復を実現できる．

一般に，プログラムの実行においてマイクロプロセッサのハードウェア資源を全て同時に活用することは難しい．ハイエンドのマイクロプロセッサは多数命令の同時実行を想定して設計されているのに対し，実際のプログラム実行において活用できる命令レベル並列性はそれほど高くない(多くの場合で2～3程度)ためである．

そこで，スレッド・レベルでの多重実行によりハードウェア資源の利用効率を改善し，信頼性向上に伴う性能/面積オーバヘッドを抑制できる．このようなスレッド・レベルでの多重実行の発展として，部分的なプログラム冗長実行を実現することで性能オーバヘッドを大幅に削減する方法などが提案されている（この場合は信頼性がある程度犠牲になる）[5)].

5.2. 命令レベル多重実行

第5.1節で説明したSRTRでは先行スレッドと追跡スレッドを生成するための仕組み（OSのサポートなど）が必要となる．これに対し，全くソフトウェアの介在なしに時間冗長化を実現する方法としてIRI（Instruction Re-Issue）が提案されている[10)]．IRIでは，データ投機実行で利用される命令再発行機構を応用して，プロセッサ内部で自ら時間冗長性を作り出す．アウト・オブ・オーダ実行をサポートする現在のスーパスカラ・プロセッサでは，その内部にフェッチした複数の命令を保持する命令ウィンドを搭載している．通常，命令ウィンドの各エントリは，対応する命令が実行を開始した（発行された）時点で開放される．これに対し，図3ならびに図5で示すように，IRIでは命令ウィンドより各

命令を2回発行する．具体的には，1回目の命令発行を実施し，その実行結果を対応する命令ウィンド・エントリに保存しておく．そして，当該命令を再度発行して2回目の実行を開始する．これら2回の実行による結果を比較することで故障を検出できる．また，故障からの回復においても命令再発行のメカニズムを利用することが可能である．ただし，このような命令レベルでの多重実行においては，実行すべき命令の取得において誤りがないことが前提となる．すなわち，命令フェッチを司るハードウェア部分において故障が発生した場合はその誤りを検出できない．

6. 新しいアプローチ

6.1. 実行振舞いに基づく故障検出

第4節と第5節で説明した空間冗長化ならびに時間冗長化は，アーキテクチャ・レベルでのプログラム多重実行をサポートすることで信頼性を向上する方式であった．しかしながら，このような手段では同一命令を複数回実行するため，性能や面積オーバヘッドが依然として存在する．これに対し，近年，より小さなオーバヘッドを目指し，プログラム実行振舞いの監視に基づく故障検出法が幾つか提案されている．つまり，プログラム実行において期待される振舞いを事前に定義する．そして，プログラム実行中に期待から外れる状態(異常と推測される状態)が検出された場合には何らかの故障が発生したと判断して命令実行のやり直しを行う．このような「Anomaly 検出」はネットワークの IDS（Intrusion Detection System）にて多く用いられている．この考え方をマイクロプロセッサでのプログラム実行に応用する．例えば，以下のような方式が提案されている．

- **間違えるはずのない分岐予測での予測ミス**：多くの高性能マイクロプロセッサでは，分岐命令の実行に伴うパイプライン・ストールを回避すべく分岐予測機構を搭載している．多くの場合で 95%以上の正しさで分岐結果（Taken か Not-Taken）を予測できる．そこで，「分岐予測の信頼度」を計測し（ハードウェア・カウンタで容易に実装可能），予測信頼度が高いにも関わらず分岐予測が誤りであった場合には何らかの故障が発生していると判断する [13]．この場合には，チェック・ポイント地点まで実行をロールバックし再実行する．
- **予想レンジを超えたデータ値の更新**：プログラム実行において各命令が生成する演算結果には，ある主の局所性や規則性があることが知られている．そこで，命令ごとにデータの局所性や規則性を検出し，プログラム実行時に生成されるデータ値がこれらから逸脱していないか判定する．もし逸脱していれば，何らかの故障が発生したと判断してプログラム実行をやり直す [9]．

このように，全ての命令実行の詳細に関して逐一故障検出を行うのではなく，命令実行の振る舞いと言ったより抽象度の高いレベルで故障検出を行うことで性能や面積オーバヘッドを削減する．

6.2. マルチコアでの故障検出

近年，複数のマイクロプロセッサ・コアを 1 個の LSI チップに搭載したマルチコア(または，チップ・マルチプロセッサ）が主流となっている．例えば，インテル社の Xeon や IBM の Power6 では 2 個，サン・マイクロシステムズの SunUltraSPARC-T2 では 8 個，ソニー/東芝/IBM が開発した Cell B.E.では 9 個のコアを 1 チップに搭載している．このようなマルチコアにおいては，あるコアで発生した故障が他コアへと伝搬する可能性があるため，信頼性向上に対する要求はより高くなる．文献 [11]では単一プログラム実行を 2 つのストリームとして別々のコアで

実行する方式を提案している．また，近年ではコア間を接続するオンチップ・ネットワークに再構成可能性を実装し，オンチップでの故障コアの完全分離を可能にする方式 [1)]や，高信頼コヒーレンス・プロトコル [4)]などが提案されている．

7. 今後の展望

本稿では，特に過渡故障に着目したマイクロプロセッサの高信頼化技術を紹介した．より高い耐故障性を実現するには，永久故障の事前検出技術も非常に重要である．これに加え，システムレベルで信頼性を向上するには,ハードウェアだけでなく，不具合のないソフトウェア開発の実現が必要不可欠である．一方，近年の高度情報化社会においては,信頼性をより広く捉え，コンピュータ・ウィルスや情報漏洩といった安全性の向上も欠くことができない [16)].

現在のマイクロプロセッサ設計においては，従来の高性能化や低コスト化，低消費電力化に加え，高信頼化は極めて重要な設計制約の１つとなっている. 今後, デバイス・レベルからアーキテクチャ・レベルだけでなく，システムソフトウェア・レベルやアプリケーション・レベルといったシステム上位階層を含めた総合的な高信頼化技術の確立が望まれる．

謝辞

本稿執筆の機会を与えて頂いた日本大学の中村英夫先生をはじめ，日本信頼性学会編集委員会の皆様に心より感謝いたします．また，日頃からご討論頂く九州大学の村上和彰教授，安浦寛人教授, 佐藤寿倫教授をはじめ,九州大学システム LSI 研究センターの諸氏に感謝します.

参考文献

1) N. Aggarwal, P. Ranganathan, J. P. Jouppi, and J. E. Smith, “Configurable Isolation: Building High Availability Systems with Commodity Multi-Core Processors,” Proc. Of the International Symposium on Computer Architecture, pp.470-481, June 2007.
2) H. Ando, K. Seki, S. Sakashita, M. Aihara, R. Kan, K. Imada, M. Itoh, M. Nagai, Y. Tosaka, K. Takahisa, and K. Hatanaka, “Accelerated Testing of a 90nm SPARC64V Microprocessor for Neutoron SER,” IEEE Workshop on Silicon Errors In Logic, Apr. 2007.
3) T. M. Austin, “DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design,” Proc. Of the International Symposium on Microarchitecture, pp.196-207, Dec. 1999.
4) R. Fernandez-Pascual, J. M. Garcia, M. E. Acacio, and J. Duato, “A Low Overhead Fault Tolerant Coherence Protocol for CMP Architectures,” Proc. Of the International Symposium on High-Performance Computer Architecture, pp.
5) M. Gomma and T. N. Vijaykumar, “Opportunistic transient-fault detection,” Proc. Of the International Symposium on Computer Architecture,” pp.172-183, June 2005.
6) S. Kaxiras, Z. Hu, and M. Martonosi, “Cache decay: exploiting generational behavior to reduce cache leakage power,” Proc. Of the International Symposium on Computer Architecture, pp.240-251, June 2001.
7) J. W. Kellington, R. McBeth, P. Sanda, and R. N. Kalla, “*IBM POWER6 Processor Soft Error Tolerance Analysis Using Proton Irradiation* ,” IEEE Workshop on Silicon Errors In Logic, Apr.

2007.
8) M. J. Mack, W. M. Sauer, S. B. Swaney, and B. G. Mealey, “IBM POWER6 reliability,” IBM Journal of Research and Development, Vol. 51, Num. 6, Nov. 2007.
9) P. Racunas, K. Constantinides, S. Manne, and S. S. Mukherjee, “Perturbation-based Fault Screening,” Proc. Of the International Symposium on High-Performance Computer Architecture, pp.169-180, Feb. 2007.
10) T. Sato, “A Transparent Transient Faults Tolerance Mechanism for Superscalar Processors,” IEICE Transaction on Information and Systems, Vol. E86-D, No. 12, Dec. 2003.
11) K. Sundaramoorthy, Z. Purser, and E. Rotenberg, “Slipstream Processors: Improving both Performance and Fault Tolerance,” Proc. Of the International Conference on Architectural Support for Programming Languages and Operating Systems, Nov. 2000.
12) T. N. Vijaykumar, I. Pomeranz, and K. Cheng, “Transient-Fault Recovery Using Simultanious Multithreading,” Proc. Of the International Symposium on Computer Architecture, pp.87-98, May 2002.
13) N. J. Wang and S. J. Patel, “ReStore: Symptom Based Soft Error Detection in Microprocessors,” Proc. Of the International Conference on Dependable Systems and Networks, pp.30-39, June 2005.
14) C. Weaver, J. Emer, S. S. Mukherjee, and S. K. Reinhardt, “Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor,” Proc. Of the International Symposium on Computer Architecture, pp.264-275, June 2004.
15) W. Zhang, S. Gurumurhi, M. Kandemir, and A. Sivasubramaniam,” ICR: In-Cache Replication for Enhancing Data Cache Reliability,” Proc. Of the International Conference on Dependable Systems and Networks, pp.291-304, June 2003.
16) 井上弘士, “信頼性･安全性とプロセッサ,” 情報処理学会誌, Vol.46, No.11, 2005 年 11 月.
17) “Tera-Flops Research Chip,” http://techresearch.intel.com/articles/Tera-Scale/1449.htm
18) “Microprocessor Though The Ages,” http://www-vlsi.stanford.edu/group/chips_micropro.html

（いのうえ　こうじ／九州大学）

井上　弘士

昭和46年生．平成8年九州工業大学大学院情報工学研究科修士課程修了．同年横河電機(株)入社．平成9年より(財)九州システム情報技術研究所研究助手．平成11年の1年間Halo LSI Design & Device Technology, Inc. にてフラッシュ・メモリの開発に従事．平成13年九州大学にて工学博士を取得．同年，福岡大学工学部電子情報工学科助手．平成16年より九州大学大学院システム情報科学研究院助教授．平成19年4月より，同大学准教授．現在に至る．高性能/低消費電力キャッシュメモリ・アーキテクチャ，セキュアプロセッサ・アーキテクチャ，性能評価に関する研究に従事．電子情報通信学会，情報処理学会，ACM，IEEE各会員．