2. 新しいデザインバランス

2

信頼性・安全性とプロセッサ

井上 弘士
九州大学 大学院システム情報科学研究院／科学技術振興機構さきがけ
inoue@i.kyushu-u.ac.jp

高性能化／低消費電力化から信頼性／安全性の向上へ

1970年代初頭に開発されて以来，プロセッサは現在までに目覚しい発展を遂げてきた．微細加工技術の順調な進歩を背景に，並列処理や投機実行，オンチップ・キャッシュの大容量化など，豊富なトランジスタ資源を有効利用することで劇的な性能向上を達成した．また，1990年代中頃からPDAや携帯電話といったモバイル機器が広く普及するようになり，バッテリ寿命の延長に対する要求が高まった．その結果，プロセッサの低消費電力化／低消費エネルギー化に関する研究・開発が加速し，現在では電源電圧の動的最適化などさまざまな技術が実用化されている．

このような技術革新により，プロセッサの応用も広がった．PCや携帯電話，家電製品はもちろんのこと，自動車や航空機に搭載された制御システム，電子マネーを対象としたアプリケーションなど，人命や財産に直接関係する場合もある．これほどまでに現代社会に深く浸透したプロセッサであるが，はたして我々は安心して利用できるのであろうか？　多くのユーザは，「正しく操作すれば，コンピュータは自分が指示した仕事だけを正確に処理してくれる」と信じている．しかしながら，近年，この大前提が成り立たない状況が発生している．

第1の理由は，プロセッサの信頼性の低下である．たとえシステムに不具合がなくとも，さまざまな外的要因により「正しくプログラムを実行する」ことができない場合がある．その主な原因として，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．

第2の理由は，第三者による悪意ある攻撃の存在である．現在我々は，整備されたネットワーク環境を活用して遠隔地の計算資源やデータベースへ容易にアクセスできる．しかしながら，これは，自分のコンピュータ・システムが攻撃対象になり得ることを意味する．たとえば，コンピュータ・ウィルスはネットワークを介して世界中に広がり，きわめて深刻な社会問題を引き起こしている．また，利用者が気づかないうちに攻撃を受け，コンピュータが不当に利用される場合や，秘密データが改竄／盗聴されるなどの被害もある．

このような背景をもとに，近年，「信頼性」や「安全性」を考慮したプロセッサの研究・開発が活発化している．高性能化や低消費電力化のためだけでなく，「安心して利用できる道具」としてのきわめて本質的な要件を満足すべく，プロセッサ・アーキテクチャを考え直そうという流れである．本稿では「信頼性」ならびに「安全性」を考慮したプロセッサの研究事例について紹介する．

フォールト・トレラント・プロセッサ

■ 基本はプログラムの繰り返し実行

LSI内部での故障は，主に，長期間にわたって存在する永久故障と，システム稼働サイクルと比較して非常に短い時間にだけ存在する過渡故障に大別される．永久故障の発生要因としては，半導体接合の破壊や回路短絡，断線などが挙げられる．一方，過渡故障は，温度の変化や振動，α粒子や宇宙線などの外部的要因だけでなく，電源電圧の変動といった内部的要因も主な原因となる．永久故障は何らかの診断を行うことで検出可能である．これに対し，基本的に過渡故障は再現性がなく，プログラム実行中に発生した場合には検出が難しい．その

ため，過渡故障の検出を目的とした高信頼プロセッサの研究が注目されている．

ここで，「信頼できる」とは，LSI 内部で何らかの故障が発生した場合でも，プロセッサが本来有する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，(1) プログラム実行中に故障を検出し，かつ，(2) 正しく誤りを訂正した実行状態へと回復できなければならない．これらをサポートする一般的な方法として，誤り訂正符号による情報冗長性の利用がある．実際に，現在のプロセッサでもキャッシュ・メモリにおいて ECC（Error Correcting Code）を採用している場合が多い．しかしながら，演算回路や制御回路など，プロセッサ全体に対して情報冗長性を追加することは難しい．

故障検出可能範囲をプロセッサ全体に広げる手段として，プログラムを複数回実行する時間冗長性や，複数プロセッサで同一プログラムを実行する空間冗長性の利用が考えられる．しかしながら，時間冗長性では性能オーバーヘッドが，また，空間冗長性では面積コストの増大が問題となる．したがって，プロセッサの信頼性向上においては，「冗長性を用いてプログラム実行を多重化しつつ，いかに性能／面積オーバーヘッドを削減するか？」が重要となる．次節より，**図 -1** に示す 3 つの代表的な高信頼プロセッサを紹介する．これらは，ハードウェア・レベルでの過渡故障検出ならびに実行回復機能をサポートしている．なお，実際には，プログラム実行の多重化だけでなく，レジスタ・ファイルや命令ウィンドなど一部の記憶素子において ECC を用いている．また，以降，「故障」とは過渡故障を意味する．

■ 空間冗長性を利用する

空間冗長性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 [1]．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．

DIVA では，図 -1 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 チェッカは構成がきわめて単純であり，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すべ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，故障発生命令の直後からプログラムの実行が再開される．これにより，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する．

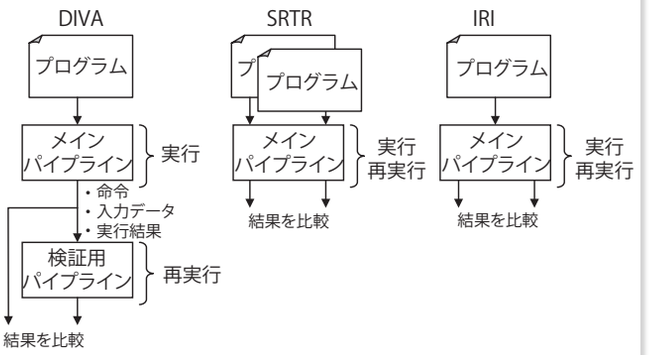

図-1　プログラム実行の多重化

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，次節で説明する 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．

■時間冗長性を利用する

時間冗長性を活用した（特別な命令実行ユニットの追加を必要としない）高信頼プロセッサもいくつか提案されている．その特徴は，同一実行ユニットにて各命令を 2 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．

代表的なプロセッサとして，SRTR（Simultaneously and Redundantly Threaded processors with Recovery）がある [9]．SRTR では，**図 -2** 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わけではない．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

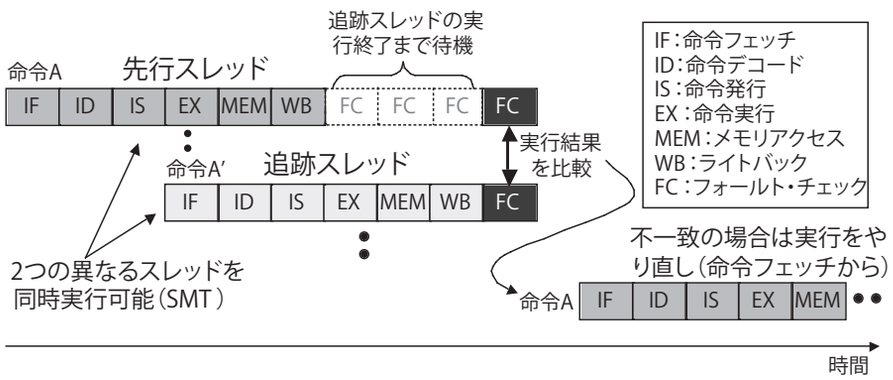

図-2　SRTRでのスレッド実行と故障検出

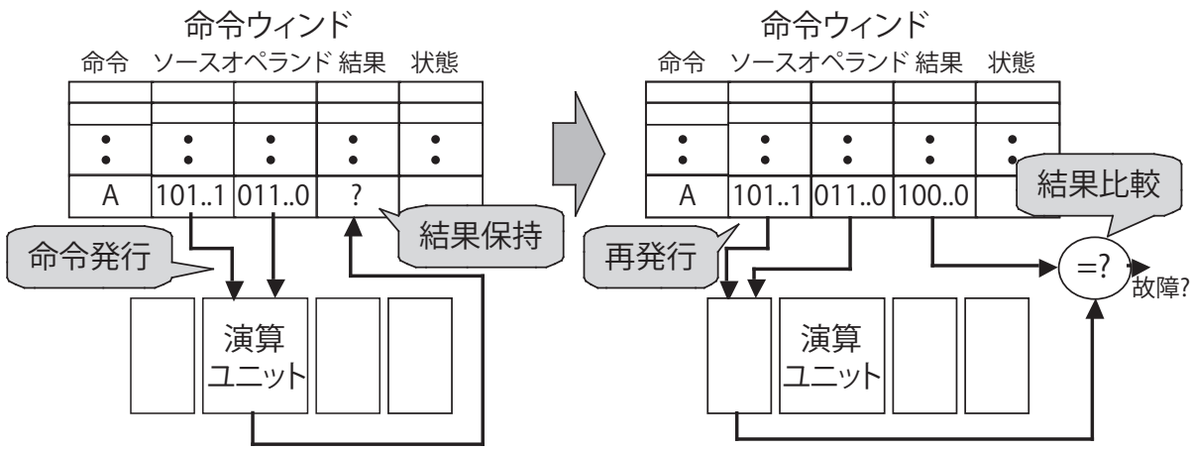

図-3　IRIでの命令再発行と故障検出

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いず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さまざま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．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．

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，まったくソフトウェアの介在なしに時間冗長性を実現する方法として IRI（Instruction Re-Issue）が提案されている[6]．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ースカラ・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** 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．

■ゼロ性能オーバーヘッドを目指して

信頼性の向上を達成するためには性能を犠牲にしなければならない．したがって，いかにして性能オーバーヘッドを低減するかがきわめて重要となる．DIVA，SRTR，IRI に共通するのは，各命令をある程度の時間差を持って 2 度実行する点である．一般に，同一命令を繰り返し実行する場合，1 度目の実行よりも，2 度目の実行の方が短時間で完了できる．これは，最初の実行によって判明／取得した多くの情報（分岐結果やソースオペランドの値など）を再実行時に利用できるためである．また，SRTR のようにプロセッサ内部の演算資源を有効活用する方法や，IRI の後続研究で提案された演算再利用，サブワード演算の活用など，ゼロ性能オーバーヘッドの実現に向けた今後の研究成果が期待される．

セキュア・プロセッサ

■何を守るのか?

安全とは，「①安らかで危険のないこと．②物事が損傷したり，危害を受けたりする恐れのないこと（広辞苑より）」である．それでは，「安全なコンピュータ」とはどのような物であろうか？　ここでは，想定された攻撃に対して，それに伴う傷害を回避できるコンピュータを「安全である」とする．したがって，想定される攻撃とその対象（つまり，何から何を守るのか？）を明確にする必要がある．

現在のコンピュータ・システムには多くの脅威が存在する．たとえば，破壊といった物理的な攻撃から，ネットワークを介した不正侵入，データの改竄や盗聴などさまざまである．古くから，通信データや記憶データの盗聴に対する防御策としてさまざまな暗号方式が用いられてきた．また，今日のコンピュータでは，パスワードによるユーザ認証が当然のように行われている．ここでは，「プログラムの実行」と「プログラム・コードならびにデータ」を悪意ある攻撃から守ることを目的としたセキュア・プロセッサの研究事例を紹介する．ここで，**図 -4** に示すように，プロセッサ・チップの内部は安全であり，悪意ある攻撃はチップ外部に対して発生すると仮定する．つまり，チップそのものに対する破壊的攻撃や利用者自身による誤った操作等は考えない．

■実行制御の乗っ取りを防止する

コンピュータ・ウィルスなどの不正プログラムは，利用者が気づかないうちに突然暴走し，その被害を拡大する．しかしながら，どんなに巧妙に作成された不正プログラムであっても，プロセッサで実行されなければその猛威は振るえない．それではいったい，不正プログラムはどのようにして実行を開始するのであろうか？　利用者自身が誤って不正プログラムを直接起動する場合もあるが，より深刻なのは実行制御の乗っ取りである．不正プログラムは，攻撃対象となるコンピュータが正規アプリケーションを実行している最中，セキュリティ・ホール（バッファ・オーバーフローなど）を利用して実行制御を乗っ取る．

セキュリティ・ホールが既知の場合はそれに応じた対策を施せばよい．しかしながら，システム内部のすべての脆弱性を事前に把握することはきわめて難しい．この問題を解決する手段として，安全であると認証されたプログラムのみを実行可能とする方法がある．システム内部にいかなるセキュリティ・ホールが存在するとしても，実行対象となるプログラムが安全であれば問題は発生しないという考えに基づいている．これを実現するためには，プログラム実行中，常にその認証を行う仕組みが必要となる．

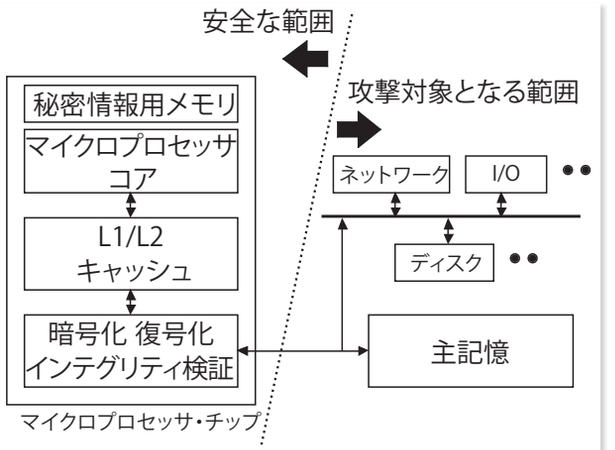

図-4　セキュア・プロセッサ

このような動的認証の実現手段として，プログラム実行の振舞いを鍵情報とする方法が提案されている[10]．事前にプログラムの静的解析を行い，たとえばシステム・コール呼出しシーケンスなどの情報を抽出する（ここでは実行ルールと呼ぶ）．そして，プログラム実行中に実行ルールに基づく振舞いが観測されるかチェックする．もし実行ルールに定義されていない動作が検出された場合には不正プログラム（つまり，未認証プログラム）によって実行の制御が乗っ取られたと判断できる．ただし，実行ルールを満たす不正プログラムは検出できないといった問題が残る．

その他の代表的な動的プログラム認証方式として MAC（Message Authentication Code）の利用がある．MAC とはメッセージ認証コードのことであり，いわば，メッセージ・データの要約である．メッセージの改竄やなりすましが行われていないことを保障するために利用される．ここでは，認証すべきメッセージをプログラム・コードと考える．MAC の計算にはハッシュ関数（入力に対し必ず固定長の異なる値を出力する不可逆関数）を用いる方法と，暗号アルゴリズムを活用する方法がある[2), 5)]．たとえば，プログラムのコンパイル時に命令ブロック単位で MAC を求め，それをオブジェクト・コードとともに保存する．そして，プログラム実行時，オンチップ・キャッシュにロードした命令ブロックの MAC を計算し，コンパイル時に求めた値と比較する．MAC を求める際にはプロセッサ・チップの内部に保存された秘密鍵を用いる．よって，MAC の値が異なる場合は不正なプログラム・コードであると判定できる（攻撃者には秘密鍵が公開されていないため）．当然，不正プログラムが混入しないよう，安全なコンパイル環境を提供する必要がある[2)]．

その他，動的認証を実現する単純な方法としてプログラム・コードの暗号化がある[4)]．プロセッサは秘密鍵に

よって復号化したコードのみを実行する．言い換えると，正規に暗号化されたプログラム・コードのみが正しく実行される．

■データの改竄を検出する

プログラムを正しく実行するためには，その制御フローだけでなく，実行途中に生成される中間結果（データ）も守らなければならない．通常，プログラム実行において必要となるすべてのデータをプロセッサ・チップ内部に保存することは難しい．そのため，外部に接続されたメモリ（たとえば主記憶）とのデータ通信が発生する．このような状況において，主に以下の攻撃が考えられる[4]．

- **Spoofing アタック**：攻撃者が偽のデータを生成し，あたかもそれがプログラム実行において正しいデータであるかのように見せかける．メモリ・データの改竄や，プロセッサ―主記憶間のデータバス値の変更などがある．
- **Splicing アタック**：攻撃者が主記憶データの配置を勝手に変更する．データの並び替えや，他メモリ空間へのデータコピーなどがある．
- **Replay アタック**：攻撃者が最新のメモリ・データを古い値に戻す（古いバージョンに改竄する）．過去に正しく記憶されたデータを保存しておき，後にそれを用いて同一アドレスの主記憶データを改竄する方法などがある．たとえば，ループ・カウンタを古い値に戻して不当にループ実行回数を増やすといった攻撃が可能となる．

Spoofing アタックは MAC を用いることで検出できる．たとえば，L2 キャッシュからのデータ追い出しが発生した際，当該データとチップ内部に保存している秘密鍵よりハッシュ値を求める．そして，追い出し対象データとともに主記憶に記憶しておく．L2 キャッシュへデータをフィルする時には，秘密鍵と主記憶からロードしてきたデータよりハッシュ値を計算し，データ書き込み時に生成したハッシュ値と比較する．もし比較結果が不一致の場合，これはロード・データに対して何らかの攻撃があったことを意味する．これと同様に，Splicing アタックに関しても，MAC を求める際にアドレス情報を含めることで解決できる．

しかしながら，過去に生成されたデータを用いて同一アドレスの値を変更する Replay アタックに関しては別の方法が必要となる．この解決策としては**図 -5** に示すハッシュ木の利用がある[7]．保護すべきメモリ空間をあるデータ・ブロックに分割（たとえば 64 バイト）し，それをリーフとするハッシュ木を構成する．各ノードはデータ・ブロックまたは子ノードのハッシュ値である．主記憶へのライトバック発生時に毎回ハッシュ木を更新するため，ルートは常に保護対象メモリ空間の最新状態を表現することになる．通常，ルートは安全なプロセッサ・チップ内部に保存される．そして，主記憶からキャッシュへデータをロードする際にはルートの値を計算し，チップ内部に保存している値と比較する．もし不一致であれば，以前のメモリ更新から現在までの間に何らかのデータ改竄が発生したことになる．

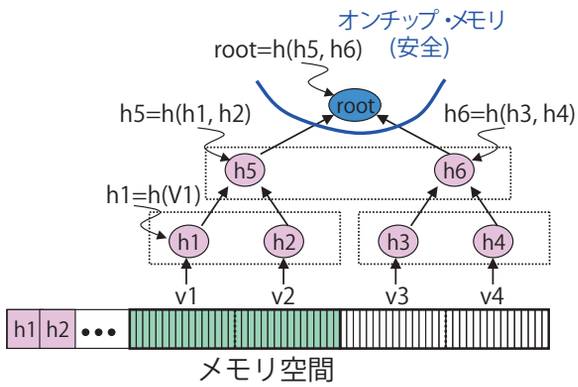

図-5　ハッシュ木によるメモリ・インテグリティ検証

■プログラム・コードやデータの盗聴を防止する

これまでに紹介した技術では，「プログラム実行の妨害」の動的検出を可能にし，安全な実行を保証することが主な目的であった．これに加え，コンピュータの安全性を考えた場合には，「プログラム・コードやデータのプライバシを確保する」ことも重要となる．XOM（eXecute-Only Memory）ではオフチップ・メモリのデータはすべて暗号化されていることを前提としており，プロセス間でのプライバシ確保を実現するマシン・モデルを提案している[4]．また，その他多くのセキュア・プロセッサでも同様の前提に立っており，L2 キャッシュと主記憶の間に暗号／復号回路を搭載する．なお，多くの場合は高速な処理を実現するために秘密鍵暗号方式である DES（Data Encryption Standard）や AES（Advanced Encryption Standard）が採用されている．

■性能オーバーヘッドの隠蔽

多くの研究では，安全性を確保しつつ，いかにして性能低下を抑制するか？という課題に挑戦している．たとえば，メモリ・インテグリティ検証では，ハッシュ木の更新やルート値の計算において多くの主記憶アクセスが発生し，メモリバンド幅を浪費する．そこで，ハッシュ木のノードをオンチップ・キャッシュに格納する方法や，メモリアクセス・シーケンスをログとして扱う方法などが提案されている[7]．また，データの暗号化においては，復号処理時間がそのままメモリ・アクセス・レイテンシとして見えるため，プロセッサ性能に大きな悪影響を及ぼす．近年のプロセッサではメモリ性能ボトルネックが

高性能化の阻害要因となっており，きわめて深刻な問題である．この対策として，復号化とメモリアクセスの並列処理を可能にするOTP（One-Time-Pad）の利用がある[7]．その他，データ・プリフェッチ技術を活用する方法や，復号化に必要なデータを予測する方法なども提案されている．

より安全で信頼できるプロセッサを目指して

現在，信頼性や安全性の向上はコンピュータ・アーキテクチャの研究分野で大きな注目を集めている．実際，ISCAやMICROといった著名な国際会議においても，「Reliability」や「Security」に特化したセッションが構成されるようになった．

フォールト・トレラント・プロセッサに関しては，ゼロ性能オーバーヘッドに向けた努力が今後も必要である．たとえば，出力結果に影響を及ぼす故障のみを検出対象とするなど，より効率的な検出／回復メカニズムの開発が望まれる．また，故障発生確率そのものの低下を目的とした命令実行方式の検討も重要である[11]．一方，高信頼化技術を応用して，プロセッサの性能向上や低消費電力化を実現しようという流れもある．たとえばRazorでは，最悪ケースに基づき決定した場合よりも低い電源電圧でプログラムを実行する[3]．もし誤動作が発生した場合には，高信頼化技術を用いてプロセッサ状態を回復し正しい実行を保証する．このように，高信頼化技術の確立は，最悪ケースを制約とする従来の最適化方式の限界を打破する新しいアプローチとして期待できる．

一方，セキュア・プロセッサに関する研究もいまだ発展途上にあり，解決すべき多くの課題がある．たとえば，セキュア・プロセッサでは，少なくとも，データの暗号化などを前提とした安全モードと，従来プロセッサと同様に動作する通常モードをサポートしている．これらのモード切り替えや相互通信においていかに安全性を確保するかといった議論が必要である[4]．また，OSとのインタラクション，秘密情報の管理方法の確立[5]，ユニークなチップIDの生成技術[8]なども重要な課題である．さらに，今後はCMP（Chip Multi Processing）のようにプロセッサ・チップ内で複数のプロセスが同時実行される環境が主流になる．それに加え，ハードウェアの複雑さが増し，設計時に何らかの誤りが混入する可能性も高くなる．このように，「プロセッサ・チップ内部は安全である」という前提が必ずしも成立しない状況での議論も今後は重要である．

プロセッサの信頼性と安全性を向上することは，安全かつ安定した情報化社会を実現する上で必要不可欠である．現在では「信頼性／安全性と性能のトレードオフ」に関して多く議論されている．これに加え，今後は消費エネルギーを考慮することもきわめて重要になるであろう．また，デバイス，回路，アーキテクチャ，システム・ソフトウェア，アプリケーション・プログラムを跨いだシステム・レベルでの信頼性／安全性向上技術の確立が期待される．

謝辞　本稿執筆の機会を与えていただいた東京大学の坂井教授に深く感謝いたします．また，高信頼プロセッサに関して貴重なコメントをいただいた九州工業大学の佐藤寿倫助教授に感謝いたします．さらに，日頃から安全性向上技術やプロセッサ技術に関して議論いただいている九州大学の安浦寛人教授，村上和彰教授，ならびに，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）Austin, T. M. : DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design, International Symposium on Microarchitecture, pp.196-207 (Dec. 1999).
2）Drinic, M. and Kirovski, D. : A Hardware-Software Platform for Intrusion Prevention, International Symposium on Mecroarchitecture, pp.233-242 (Dec. 2004).
3）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 (Dec. 2003).
4）Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J. and Horowitz, M. : Architectural Support for Copy and Tamper Resistant Software, International Conference on Architectural Support for Programming Languages and Operating Systems, pp.168-177 (Dec. 2000).
5）Lee, R. B., Kwan, P. C. S., McGregor, J. P., Dwoskin, J. and Wang, Z. : Architecture for Protecting Critical Secrets in Microprocessors, International Symposium on Computer Architecture, pp.2-13 (June 2005).
6）Sato, T. :A Transparent Transient Faults Tolerance Mechanism for Superscalar Processors, IEICE Transaction on Information and Systems, Vol. E86-D, No. 12 (Dec. 2003).
7）Suh, G. E., Clarke, D., Gassend, B., Dijk, M. and Devadas, S. : Efficient Memory Integrity Verification and Encryption for Secure Processors, International Symposium on Microarchitecture, pp.339-350 (Dec. 2003).
8）Suh, G. E., O' Donnell, C. W., Sachdev, I. and Devadas, S. : Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Functions, International Symposium on Computer Architecture, pp.25-36 (June 2005).
9）Vijaykumar, T. N., Pomeranz, I. and Cheng, K. : Transient-Fault Recovery Using Simultaneous Multithreading, International Symposium on Computer Architecture, pp.87-98 (May 2002).
10）Wagner, D. and Dean, D. : Intrusion Detection via Static Analysis, IEEE Symposium on Security and Privacy (S&P01), pp.156-168 (May 2001).
11）Weaver, C., Emer, J., Mukherjee, S. S. and Reinhardt, S. K. : Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor, International Symposium on Computer Architecture, pp.264-275 (June 2005).

（平成17年9月8日受付）