計算機アーキテクチャ 130－19
（1998．8．6）

追跡型多重系による高信頼コンピュータ・システム

加藤肇彦†

コンピュータ・システムの高信頼化の方式を提案する。本方式は従来から適用され実証されてきた空間ダイバーシティーと時間ダイバーシティーを組み合わせ、さらにこれらの方式を止揚することによって達成される。本発表ではまず人工衛星「ひてん」に試験的に搭載されたコンピュータの障害解析より得た障害分布を紹介し、費用効果を維持しながら高信頼化を実現する方法を考察する。次にその結果を踏まえて「追跡型多重系」という新しい方式を紹介し、システムの各レベルにおける適用と、信頼性向上の効果を評価する。さらに性能と費用への影響を考察し、最後に拡張形態と縮退形態を紹介する。

A High Dependability Computer Attainable by a Tracking Redundancy Scheme

Hatsuhiko Kato†

A high dependability computer configured with a new concept will be proposed. The concept was created by combining and sublating spatial diversity and time diversity which were already worked out and proven. This paper starts with the fault analysis of the on-board computer tested on the satellite "Hiten" and scrutinizes the fault distribution over components constituting a typical computer system to optimize the reliability to cost ratio. Based upon these study, the paper proposes a scheme, which we named "Tracking Redundancy" or "Skewed Parallelism". The paper first introduces its generic configuration and thereafter develops its details introducing examples to be applied at different levels of implementation and granularity. The paper next evaluates the improvement in the system reliability expected by the implementation of the scheme. The paper also studies the various influence of the scheme on the system performance and cost together with their partial solutions. The paper shows some suggestions for the extension and degeneracy of the scheme.

1．従来の高信頼化技法の問題点

1．1　空間ダイバーシティー

空間ダイバーシティーは従来から耐障害性の典型的な実現手段として利用されてきた。具体的には同一構造の処理機構を複数単位用意して冗長系を構成し、それらの処理結果を比較して多数決をとるという方式である。[1] この方式の問題点は、多数決をとる回路が一重であるため弱点となることである。これを補うために多数決回路を多重化すると、今度はこれらの多数決回路の診断をするための多数決回路が必要となり、議論が再帰的になってしまう。

1．2　時間ダイバーシティー

このような再帰的議論を回避するために、我々は多数決回路は1段のみとし、その出力を各処理機構にフィードバックして自己診断させる方式を考案した。これは時間軸におけるダイバーシティーであり、「ひてん」の搭載実験によりその有効性を実証した。しかしこの方式では

† 湘南工科大学 工学部 情報工学科
Department of Information Science,
Shonan Institute of Technology

自己診断に要する時間オーバヘッドが処理能力と応答性を低下させるという問題点がある。

1．3　共通の問題点と多重化の着眼点

上記2種類のダイバーシティーに共通の問題点は、実際のコンピュータ・システムの障害分布を反映していないことである。通常コンピュータの障害はCPUやランダムロジックに局在しており、規則的構造を持ちしかも量産されているメモリ素子の障害は起こりにくく、かつ発生後も誤り訂正符号(以下 ECC)により除去できる可能性が高い。このことは我々の「ひてん」搭載コンピュータの実測データが裏付けている。[2),3)]

最近のコンピュータ・システムのコスト構成を分析すると、メモリ素子のコスト比率が増大する傾向が看取される。障害が起こりやすい部分のみを多重化し、障害が起こりにくく、かつたとえ起こっても修正が容易なメモリ素子は一重にすることによって、限られた費用を障害対策に重点的に配分することができる。さらに、重点的多重化は消費電力、発熱、重量、サイズの低減を可能にする。

2．追跡型多重系

2．1　発展過程

我々は1970年代よりコンピュータ・システムの耐障害性向上に関心を持ち、その過程で「追跡型多重系」の構想を発展させてきた。[4)] しかしながらこの構想は当時の技術基盤に照らして実現上の問題が多く、社会の耐障害性コンピュータへの要求も少なかったため、顕著な評価を得られなかった。その後アーキテクチャ並びに半導体プロセスの進歩により構想の実現性が向上し、同時にコンピュータの普及による耐障害性の要求も増大してきた。最近の類似構想の発表によっても、この傾向は裏付けられる。[5),6)]

2．2　包括的定義

追跡型多重系は包括的に「**複数単位の処理機構(ユニット)に、時間差をおいて同一処理を実行させ、先行ユニットの処理結果を後続ユニットにチェックさせる方式**」と定義される。ここでいうユニットとそれによって実行される処理の粒度としては、表1に示す各レベルが想定できる。次章以降でこれらの各レベルにおける追跡型多重系の適用例を紹介する。

表1．追跡型多重系構成の粒度

ユニットの粒度	処理の粒度
算術論理演算ユニット	算術論理演算
マイクロコード実行	マイクロコード操作ユニット
CPU	機械語命令
CPU＋メモリ＋結線論理	タスク/プログラムモジュール
データ処理システム	ジョブ/ユーザトランザクション
機能実行システム	システム機能

3．追跡型多重系構成の適用例

例1：算術論理演算ユニットレベル

処理装置は複数個の算術論理演算ユニット(以下ALU) ALU1からALUnを持ち、これらは以下のように単一のデータフローに対して時間差をおいて同一の処理を加える。データバス経由CPUに取り込まれたオペランドはデータレジスタに転送され、ALU1はこれに対し算術論理演算を実行する。中間にあるALUk-1のオペランドと処理結果は一時的に先着順バッファ(以下FIFO)に保存され、ALUkによる処理とチェックを待つ。ALUk-1によるFIFOへの書き込みとALUkによるFIFOからの読み出しは互いに非同期に実行される。以下処理結果はALUからFIFO経由次のALUへとチェックを受けながら順次転送され、ALUnより演算結果レジスタへ出力される。ALUkによる処理結果とFIFOの内容の不一致が検出されると、ALUkはまずその処理を再実行し、なお不一致が起こる場合はALUk-1に不一致が伝達され、ALUk-1が再実行を行なう。

例2：マイクロコード実行ユニットレベル

メモリよりフェッチされた命令は命令レジスタに置かれ、命令解読器は命令の操作部を解読して対応するマイクロプログラムを起動する。マイクロプログラムを構成する各マイクロコードは制御メモリより順次取出され、マイクロプログラム実行ユニットにより実行される。実行ユニットは複数組用意し、マイクロコード、オペランド、実行結果の組はFIFOに保存され、後続の実行ユニットによる実行・比較を待つ。マイクロプログラムのさらに下位にナノプログラムやピコプログラムの層がある場合には、これらの層に対しても追跡型多重構成を再帰的に適用することが可能である。

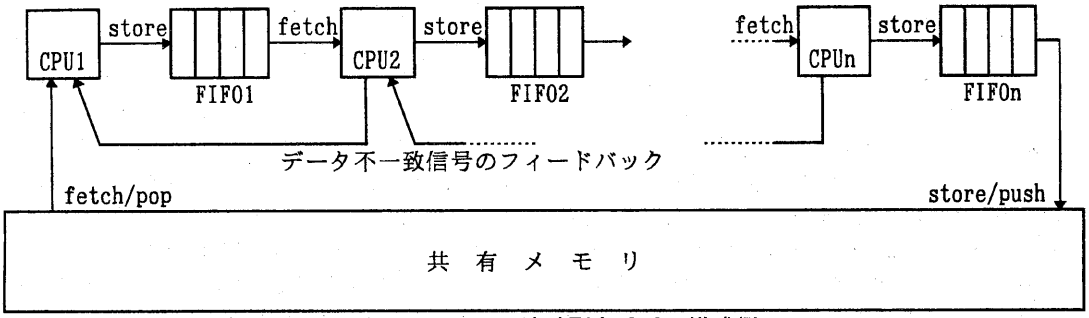

図１．CPU レベルの追跡型多重系の構成例

スーパスケーラ構成の CPU に対しては、各パイプラインストリング(ウェイ)毎に独立の FIFO を用意する必要がある。水平マイクロプログラム方式では、上記レベル１の冗長構成が適用できる。ハーヴァードアーキテクチャのように命令とデータが独立の情報パスを流れる構成においては、それぞれの情報パスに対して独立の FIFO を用意し、これらの内容をタグにより同期させる必要がある。このレベルの耐障害性機能は CPU アーキテクチャに集積化しておき、ソフトウェアや応用に対しては隠蔽することも可能である。

例３：ＣＰＵレベル

複数個の CPU(CPU1、CPU2、...CPUn)と、これらが共有する一組のメモリと論理により処理装置を構成する。[7] CPU1 は最も先行してプログラムストリングを実行し、CPU2 はこれに次ぐ。同様に CPUk は CPUK-1 に後続し、CPUk+1 に先行する。直接先行する CPU と後続する CPU の間には FIFO が介在し、先行 CPU は命令、タイムスタンプ、処理前データ、処理結果の組を FIFO に送出し、後続 CPU はこれらを先着順に FIFO から読み出して、動作を追跡しながら確認する。メモリからの読み出し動作は最先行の CPU1 のみが実行し、メモリへの書き込み動作は最後続の CPUn のみが実行する。同様にメモリのスタック領域への push/pop の動作も FIFO をメモリと見なして実行される。後続 CPU により処理結果を比較し、不一致が検出された場合は直先行 CPU に停止信号をフィードバックした後 FIFO に蓄積されたコンテクストによりリトライを実行する。 リトライの結果不一致が再度発生した場合の回復法としては、数種類の候補が考えられる。一例として第三の CPU が先行する２個の CPU をその処理結果により診断し、自己の処理結果と一致する方を正しいと判断して出力するという方法が考えられる。図１は CPU レベルの追跡型多重系の構成例を示す。

パイプライン処理で命令が実行される場合、命令がある期間インストラクションバッファに滞留しても処理結果との対応をタグにより維持できるように、CPU の設計時に追跡機能を組み込んでおく必要がある。アウトオブオーダ制御により命令実行順序の緩和を行なう場合、リオーダバッファと連動する命令のタイムスタンプの順序変更機能を組み込む必要がある。

キャッシュを使用する場合は、整合性維持と遅延の防止に留意する必要がある。一次キャッシュに対しては、二つのレベルで対処が可能である。一つは追跡を実行する全 CPU を１個のチップに集積して一次キャッシュを共有させ、チップ間遅延を防止する方法である。今一つは、全ての CPU＋一次キャッシュの組のリプレースメント動作を一致させ、追跡は外部端子レベルで実行する方法である。これらの場合はハードウェアの専用設計が必要である。二次キャッシュに関しても同様に、共有キャッシュと個別キャッシュの二通りの方式が可能である。現在の CPU の大半は分散キャッシュをもつが、これらに対してそれぞれの FIFO が必要である。

複数個の CPU が単一のメモリを共有するため、メモリと FIFO の内容の整合性に注意する必要がある。最先行 CPU がデータを変更して FIFO に store/push した場合は、その後最後続 CPU がメモリに対してそのデータの store/push 動作を完了するまでは最先行 CPU はメモリからの fetch/pop 動作を保留しなければならない。

このレベルでの追跡型多重系は１．３節で述べた障害分布を反映し、従来の冗長方式のもつ問題点の欠点を補っている。

VLIW 方式の CPU に関しては上記レベル１の多重系を構成し、マイクロプログラム層を含まない縮少命令セットコンピュータでは、レベル２に相当する多重系を構成すればよい。

例４：ＣＰＵ+メモリ＋結線論理レベル

このレベルでは追跡型多重系を CPU、メモリ、および結線論理回路をユニットとして構成する。また、メモリの内 RAM のみを各ユニットに持ち、障害の起こりにくい ROM は一重として共有する方法も考えられる。先行ユニットと後続ユニットの処理結果はタスクまたはプログラムモジュールレベルで比較される。比較は CPU レベルに比べて長い時間間隔で行なわれるが、ハードウェアの専用設計は少なくて済み、タイミング要求も緩やかである。同一入力に対する処理結果の同一性が保証される限り各ユニットは必ずしも同一構成である必要はないため、設計多様化による信頼性向上が期待できる。

例５：データ処理システムレベル

データ処理システムとして自己完結したシステムに追跡型多重系を適用する場合は、仮想記憶との関連を考慮する必要がある。さらに仮想記憶は小さな実メモリを大きなアドレス空間全体と同じ容量を持つかのように見せる古典的な「real<virtual」と、大きな実メモリをそれより小さなアドレス空間に折りたたんで収容する現在の「real>virtual」の両方がある。以下にそれぞれの考察結果を示す。

(a)real<virtual の場合

実メモリと補助メモリの間の整合性(コヒレンシ)の維持が必要である。整合性維持は全ユニットの実メモリの間でも必要である。スワッピング実行の前にはメモリからの新しいデータのフェッチを停止して、全ての FIFO を空にしなければならない。そしてスワッピング完了後は TLB と DAT の内容を無効化しなければならない。実メモリと補助メモリのデータ転送については、次章の入出力動作の節で考察する。

(b)real>virtual の場合

実メモリは CPU から直接アクセス可能な仮想メモリ空間に対応するバンクに区切られ、バンクの切り替えによって大容量の実メモリ全体をアクセスする。全 CPU からアクセスされるバンクの同一性と同時性を保証するために、バンク切り替え時には全てのストア動作が完了するまで切り替えとフェッチ動作を保留する。

例６：機能実行システムレベル

機能実行システムレベルでの追跡型多重系は、ソフトウェア全体の処理結果に対して適用される。このレベルでは専用ハードウェアはほとんど必要ないが、耐障害機能はオペレーティングシステムまたはアプリケーションに包含しておく必要がある。データはネットワーク経由緩いタイミング要求で非同期的に交換される。プロトコルの整合性が保たれている限り処理機能はバージョンの異なるソフトウェアで、しかも仕様の異なるハードウェア上で実行されてもよい。

４．詳細事項の検討

４．１　詳細事項検討の必要性

前章で追跡型多重系の構想と数種の構成例を説明した。本章では実施上必要になる事項を取り上げ、対策例により詳細に検討する。

４．２　ＦＩＦＯバッファの管理

追跡を実行するユニットの処理速度に不整合があると、FIFO に滞留する情報の待ち行列が増大し、あふれによる情報の喪失が起こることがある。これは追跡型多重系の全てのレベルで起こり得る共通の問題である。ここでは以下の二つの対策を紹介する。一つは FIFO の容量に下流ほど大きくなるように傾斜をつけ、上流へのあふれの伝播を防ぐという方法である。しかしこの対策は部分的であり、FIFO の容量に拘らずあふれの可能性は皆無にはならない。このためには FIFO が満状態になれば上流側 CPU の処理を停止するという方法が考えられる。

４．３　割り込み処理

追跡を実行する全てのユニットに対して、プログラムコンテクスト切り替えの同時性を保証する必要がある。最先行ユニットは割り込み処理要求を受け付け、認識信号を要求元へ返した後、ベクタアドレスより割り込み処理プログラムを開始する。各ユニットが独立のメモリを持つ場合は、タイムスタンプによって割り込み受付クロックの同時性を維持する。

４．４　入出力処理

入出力処理は DMA 方式とメモリマップド入出力の二種類に大別され、DMA 方式はさらにサイクルスチールとバーストの２モードに分類される。サイクルスチールを CPU レベルで実行する場合は、FIFO と共有メモリの内容の整合性を維持するために CPU と DMA デバイスが同一バン

クをアクセスすることを禁止するか、FIFOとメモリの内容の整合性を維持するために FIFO 中のデータが入出力の対象になったかどうかスヌーピングによって確認しなければならない。CPU+メモリ+結線論理レベルでは全ユニットのメモリ内容の整合性を維持するために、DMA 転送のタイミングをタイムスタンプによってそろえる。CPU レベルのバーストモード転送の場合はまず CPU1 はメモリアクセスを中止し、FIFO に保持されていた中間結果の全 CPU による処理が完了した後、CPU1 は DMA デバイスを起動してデータバスを空け渡す。メモリマップド入出力ポートでは入出力データは同時性を維持するために処理中は専用レジスタにラッチする。

4．5　リード・モディファイ・ライト動作

テストアンドセット命令使用する場合、CPU レベルでは、 CPU1 がセマフォビットを読み取り、テスト後 FIFO に書き込む。 メモリへの書き込みは CPUn が実行する。CPU+メモリ+結線論理レベルでは全 CPU によるセマフォビットのテストが完了後に次の命令の実行を開始する。

4．6　タイマ管理

多重系では CPU 間でハードウェアタイマへのアクセス時間に差が生じるため、時間管理の一元性への考慮が必要である。時間管理の精度への要求が厳しくない場合は、タイマは他の入出力ポートと同様に見なしてよい。タイマの設定は CPUn によって実行し、読み取りは CPU1 によって実行する。時間のずれは無視できる。高精度の時間管理が必要な場合は、時間ダイバーシティーは適用できない。タイミング信号は本質的に時間軸上で一意であるからである。

5．方式の評価

5．1　信頼性の向上

本方式によるシステムの信頼性は、誤りを訂正する耐障害性と、安全側動作を取らせるための障害発生の告知能力によって評価する必要がある。本方式は本質的に空間/時間冗長系であるため、これらの評価基準については、従来の空間/時間冗長方式と同程度である。

5．2 稼働率

稼働率の向上は一過性で被覆可能な障害による停止の低減と、さらに停止時の回復時間短縮により達成できる。本方式は処理の流れの上で即時に誤りを訂正するため、従来の空間/時間冗長方式より停止頻度が低い。また、停止した場合には FIFO に保持されたコンテクストを利用することにより回復時間の短縮が期待できる。

5．3　平均処理能力

CPU レベル以下の追跡型多重系では、中間データの転送と比較はマシンサイクルの中で達成でき、非冗長系に比べて処理能力は低下しない。これより高いレベルでは転送と比較のためにマシンサイクルが必要となり、処理能力の低下が起こるが、処理とフィードバックデータの比較を逐次的に行なう従来の空間/時間冗長方式よりは処理能力の犠牲は少ない。稼働率を考慮に入れた広義の平均処理能力では、本方式が高い。

5．4　応答性

割り込み処理要求の発生より受理までの時間は、中間データを FIFO に保持する本方式では非冗長系より長くなる。しかし従来の空間/時間冗長方式に比較した場合は処理能力が高い分だけ応答性の犠牲は少ないと言える。

5．5　コスト

CPU 以下のレベルで本方式を実現するためには、他のオンチップ多重系と同様に集積化コストが必要であるが、これが標準アーキテクチャとして量産化されればコストは低下する。しかも全機能を多重化する従来方式とは異なり、集積度の制約の影響は少ない。また、ソフトウェアや応用の面からは多重化されていることを意識する必要はなく、特にコストは必要ない。逆に高位レベルでは本方式はハードウェアコストは不要になるが、ソフトウェアコストが必要になる。

6．拡張と縮退

6．1　動的再構成

システムの長時間運用中の一部要素の劣化に対して、再構成法と、それに必要な劣化要素同定のための診断アルゴリズムを考察する。

後続ユニットは不一致を検出するとまず再実行し、不一致が解消しなければ不一致は先行ユニットに伝達され、先行ユニットにより不一致を起こした処理が再実行される。複数の先行ユニットの全てが既に一致している場合は、後続ユニットによる診断結果を無視して先行ユニットの出力をそのまま正しいものとすることも考えられる。この場合異なる結果を出した後続ユニットには決められた回数だけ再実行

させて、なお不一致が解消しない場合はこの後続ユニットを異常と見なし、一時的に切り離して診断に参加させない。切り離されたユニットは修理を受けるか、自然回復を待って待機状態のプールに登録される。修理が完了して待機状態に入るか否かは、自己診断プログラムによるか、正常状態のユニットと同じ処理を実行し、結果を自分で比較することにより可能である。追跡型多重系の縮退形として、先行のkユニットだけを処理に参加させて残りは待機させておく方法もある。

6．2　単独動作

診断の結果ただ一つのユニットを除いて全てが異常と診断され、なお処理の信頼性を維持する必要がある場合は、正常常態にあると見なされたユニット単独で縮退した追跡チェックを続行して異常ユニットの回復を待つ。この場合は決められた回数だけ同じ処理を反復実行し、結果が一致した場合のみ出力する。これは空間ダイバーシティーを含まず時間ダイバーシティーのみに依っているため、処理能力は低下する。信頼性より処理能力を重視する場合は反復実行をやめて非冗長動作を実行する。

6．3　マルチスレッド追跡

追跡型多重系は複数のデータフローに対しても適用可能である。処理の半順序関係を解析し、全体処理を複数のスレッドに分解して各ユニットに分配する。信頼性上クリティカルな意味を持つ処理は複数のスレッドに多重に割り付け、プールにある待機状態ユニットを処理状態に移行させて実行させる。各スレッドの合流点であらかじめ決めた個数以上の処理結果が一致すればその結果は正しいものとして下流スレッドに送る。マルチスレッドの追跡型多重系は、従来の空間/時間ダイバーシティを内包させることにより、処理能力を維持しながら自律性を持たせて柔軟に高信頼性を追求することができる。

7．おわりに

以上に従来の多重系の実績に基づく問題点を指摘し、追跡型多重系という新しい方式を提示した。次に各レベルでの本方式の実現法を例により示した。そして各従来方式との定性的な比較を信頼性、性能、コストの各観点から行ない、本方式が同等以上であるという見通しを得た。さらに、本方式の拡張と縮退形態にも言及し、異なる要求条件に対する柔軟な対処法の糸口を示した。

上記の各試みにも拘らず、今回の発表は本方式の検討としては完結したものとは言えない。まず、実現法の例はいずれも机上検討の段階を出ていない。実用化を踏まえた詳細の問題点摘出と具体化が残されている。また、従来方式との比較は定性的な範囲に留まっている。現実的なモデルを想定した定量的な比較はこれからの課題である。新方式の提示段階においては課題提起自体が課題であると考え、今回はまず概念を提示し、その要検討事項の範囲を見極めることに重点を置いた。各位の御叱正を頂きたい。

最後に本報告に御助言頂いた、国際医療福祉大学の井原廣一教授、イリノイ大学のRavishankar Iyer教授、およびIFIP WG 10.4の各メンバに深甚の謝意を表します。

参考文献

1) Avizienis,A.et al.:Featured Issue on Fault Tolerance, *Computer,* Vol.30, No.4(1997).

2) Takano,T.,Yamada,T.,Kato,H.,Tanaka,T., Ihara,H.,Kanekawa,N.,Maejima,H.:Long-Life Dependable Computers for Spacecrafts, *Proc.15th International Federation for Information Processing.* pp.153-173(1989).

3) 加藤，高野，上杉，周東，山田，金川，井原，田中：衛星搭載用コンピュータの高信頼化とその実証，情報処理学会論文誌，Vol.35, No.9, 1936-1947(1994).

4) 加藤：追跡型擬似二重系計算機，特許出願書82098(1975).

5)Smith,J.E.,Vajapeyam,S.:Trace Processors: Moving to Fourth-Generation Microarchitectures, *Computer,* Vol.30,No.9, pp.68-74 (1997).

6)Chandra,A.,Bossen,D.C.,: Time-Lag Duplexing － A Fault Tolerance Technique for Online Transaction Processing Systems, *Proc. 1997 Pacific Rim International Symposium on Fault Tolerant Systems,* pp.202-207(1997).

7) Stenström,P.,Brorsson,M.,Dahlgren,.F., Grahn,H.: Boosting the Performance of Shared Memory Processors, *Computer,* Vol.30, No.7, pp.63-70(1997).