計算機アーキテクチャ 93－11
(1992. 3. 23)

# 実時間用並列処理計算機 CODA の ハードウェアアーキテクチャ

西田健次*、戸田賢二、島田俊夫
電子技術総合研究所
〒305 茨城県つくば市梅園 1-1-4

CODA はセンサフュージョンシステムに用いる実時間用並列処理計算機である。CODA は同型のプロセッシングエレメントをパケット通信ネットワークで結合した形態をとり、高い処理効率とスケーラビリティを両立することを目指している。CODA のプロセッシングエレメントは、32bit の優先度フィールドを持つことでデッドライン時刻を直接優先度として扱うことが可能である。そして、割り込み処理や通信パケットなどもこの優先度にしたがって処理される。これにより全ての処理が優先度にしたがって行なわれることなり、効率的な実時間処理を行なうことができる。

# The Hardware Architecture of a Real-Time Paralell Processor CODA

Kenji NISHIDA, Kenji TODA, Toshio SHIMADA
Electrotechnical Laboratory
1-1-4 Umezono Tsukuba-shi, IBARAKI 305, JAPAN

This paper presents CODA, a parallel processor architecture for real-time applications. CODA consists of multiple identical processors connected via a packet communication network. A processor element for CODA integrates computation and communication to a processor architecture, while prioritizing every essential operation such as process execution, interruption handling and communication. CODA provides a 32-bit priority field to handle deadlines as a priority to reduce the overheads for priority management. Thereby, CODA provides effective hardware support for building real-time systems.

*Email: nishida@etl.go.jp

## 1 はじめに

将来の高度情報処理システムにおいては、外界の認識及び判断を含んだ高度な自律性が要求されることが予想される。このような要求に対して、膨大な数の異種センサからの情報を統合し、従来にない新たなセンシング機能を付与し、認識 / 判断等の高度な処理に対する支援を行なおうとするものとしてセンサフュージョンシステムが提案されている [2]。

センサフュージョンシステムを支える計算機アーキテクチャは、外界の情報を得るため多数のセンサからの情報を処理するための高い I/O 能力と実時間性、そして認識 / 判断などを高速に行なうための高い計算能力の両立が必要となる [4]。従来より、高い I/O 能力と計算能力を両立させるためには並列処理が有望であるが、並列処理に特有のプロセッサ間通信やプロセススケジューリングのオーバヘッドにより処理時間の予測可能性が損なわれるため、実時間処理への適用は困難であると考えられてきた。

しかし、プロセッサアーキテクチャレベルから実行時間の予測可能性を考慮した設計を行なえば、並列計算機を実時間処理に適用することは可能であり、充分に高い応答性を保ちながら、より高い I/O 能力と計算能力を持ったシステムを構築することができる。本稿では、プロセッサアーキテクチャレベルで実時間処理を支援することを目的とした並列計算機 CODA のアーキテクチャについて述べる。

CODA は高速な同期機構を有し、通信機能をプロセッサのパイプラインに統合している。実時間処理を支援する機能としては、プロセッサ間通信により命令実行が影響されないように通信専用のパイプラインを有すること、プロセッサ内部に優先度キューをハードウェアで持つこと、通信ネットワーク上で優先度にしたがったルーティングが行なわれる [6] ことなどがあげられる。

## 2 センサフュージョンシステム用計算機システムの要件

センサフュージョンシステム用計算機の満たすべき条件は、(1) 実時間処理に対する支援、(2) 膨大な数のセンサ / アクチュエータの制御を可能にする I/O バンド幅、(3) 判断 / 認識などの高度な処理を行なうための高速な計算処理の 3 点であると考えられる。素子技術、実装技術による高速化が限界に達しつつあると考えられるため、高速な計算能力を得るためには並列化が必須である。本節では、並列化に際し問題となる実時間処理の支援について考察する。

### 2.1 実時間処理の支援

実時間処理においては、実行時間の予測可能性を確保することが重要な課題である。プロセス (あるいはタスク) レベルでは、プロセスの終了しなくてはならない時刻 (デッドライン) にしたがってプロセス毎に優先度を与え、優先度の順に実行していくことにより、優先度高いプロセスの実行時間の上限を定めることができると考えられている。しかし、計算機のハードウェアレベルで考えると、例えば、一つの命令の実行に必要な時間でさえも、割り込みの有無やキャッシュのミスヒットなどにより、予測することが困難な場合がある。そのため、実時間性に対する要求の厳しいシステムでは、割り込みを使わずに入出力処理が記述され、また、キャッシュなどの確率的に実行時間が変化するようなハードウェア構成は避けられてきた。

センサフュージョンシステムでは、接続されるセンサ / アクチュエータの数が膨大なものとなるため、一つのプロセスが全センサ / アクチュエータの制御を行なうような記述をすることは、プログラミングの点からも困難であり、実行効率の点からも望ましくない。センサ / アクチュエータが計算機からの制御を必要とする場合に、割り込みをかけ入出力処理を行なう形での記述を行なう必要がある。そして、割り込み要求が発生した際に、実行中のプロセスを中断して割り込み処理を行なうか、あるいは、割り込み処理を遅延すべきかは、各々の処理のデッドラインによって決定されるべきである。しかし、従来、ハードウェアで用意されていた程度の割り込みレベル数では、割り込み処理に対してプロセスの持つ優先度と同等の解像度を与えることが不可能であるため、割り込み要求はプロセスの実行に優先して処理されることが多い。そのため、割り込み処理によって優先度の高いプロセスの実行が妨げられ、プロセス実行時間の予測性が損なわれてしまうと考えられる。割り込み処理に対してもデッドラインを設定し、それに応じた優先度を付与することにより、割り込み処理とプロセススケジューリングを一元的に管理することが可能になる。そのためには、割り込み要求に対して動的にデッドライン (優先度) を与える機能が必要となる。

並列計算機においては、プロセッサ間の通信 / 同期などの基本操作が、プログラムの実行時間に与え

る影響が大きいと考えられる。そこで、通信や同期操作によって命令実行時間が影響を受けないようにする必要がある。そして、通信に伴うパケットの処理もプロセスと同様の解像度を持った優先度を与えることにより、処理時間の予測可能性を確保する。

## 2.2 スケーラビリティ

センサフュージョンシステムでは、接続されるセンサ / アクチュエータの数が膨大なものとなるため、単一のプロセッサでは必要なI/Oバンド幅を確保することは困難である。そこで、並列プロセッサを用いることでI/Oバンド幅を拡張できるようにする必要がある。しかし、実時間処理に適用する場合には、単にバンド幅を向上するのみでなく、実行時間の予測性も確保されなくてはならない。すなわち、並列化による実行速度、及び、入出力バンド幅の向上とともに、最小デッドライン間隔を確保する事とプロセッサに許される最大ロードファクターの低下を招かないようにすることが重要である。我々は、実時間用並列計算機におけるスケーラビリティを次のように考える。

**拡張性:** 同型のプロセッサを接続し台数を増やすことにより、システム規模を容易に拡張できること。それにより、制御可能なセンサ / アクチュエータの数を増やすことができると同時に、処理可能なプロセス数も増やすことができる。

**応答性:** プロセッサ台数を増やすことにより通信や同期のオーバヘッドが生じるが、それによって最小デッドライン間隔が影響されないこと。

CODA は上記二点、特に応答性の確保を目指して設計されている。

# 3 CODA の概要

## 3.1 プログラミングスタイル

多数のセンサ / アクチュエータを効率良く制御するためには、複数のプロセスが互いに通信しながら処理を進めていく形でプログラムを記述することになる。このような記述法を実時間処理に用いる場合には、プロセス間の通信や同期操作の処理時間をどのように見込むかということが問題となる。CODA では、各プロセスにデッドラインに応じた優先度を持たせるのみでなく、一つ一つの命令実行やパケット送受などの基本操作に対してもプロセスのデッドラインに応じた優先度を与え、一命令実行サイクル毎に優先度をチェックすることで基本操作のレベルから実行時間の予測性を確保する。

プロセススケジューリングには、デッドラインスケジューリング、最小ラクシティファーストスケジューリングなどいくつかの手法があるが、プロセスを実行する (あるいは実行が終了する) 時刻を元にスケジューリングを行なうことは共通している。本稿ではデッドラインスケジューリングを例に取り CODA でのプロセススケジューリング法を述べることにする。

プロセスは動的に生成されるものとするが、プログラム実行に必要な資源はシステムの持つ資源を越えないものとする。プロセス生成時に、現在時刻との相対的なデッドライン (あらかじめ宣言されている) から、そのプロセスの実行が終了しなくてはならないデッドラインが絶対時刻として計算される。プロセスキューが全体で一つであれば、デッドラインが最も近いものから順に実行していくことにより、実時間性は保証される。しかし、並列プロセッサ上に単一のプロセスキューを実装した場合、プロセスキューに対するアクセス競合により、実行効率が低下すると考えられる。そこで、プロセスキューは各プロセッサ毎に持ち、生成されたプロセスを各プロセッサに分配する。これにより最適な実行順序が守られなくなる場合、実時間性を保証できるロードファクターの低下を起こす。しかし、デッドラインに充分な解像度があれば、その影響は充分に低く抑えることができると考えられる。

## 3.2 プロセス間の同期

CODA では、プロセスは並列呼び出しによって生成され、プロセスの終了によって消滅する。ユーザ定義による関数やループ本体等がプロセスとして用いられる。並列呼び出しは、呼ばれた側からの返値の有無により呼び出し側が同期を取ることを特徴とする。呼び出し側で返値を使用するよりも前に返値が定義されていれば、呼び出し側のプロセスは実行を中断することなく処理を続ける。呼び出し側で使用する際に返値の値が定義されていない場合、呼び出し側のプロセスは処理を中断し同期待ちの状態となる。その時プロセッサは実行コンテクストを切替えることにより、プロセッサの利用効率を高く保つようにする (図 1(a))。

プロセスは、その生成時にプロセスの持つデッドラインにしたがった優先度を与えられる。プロセス

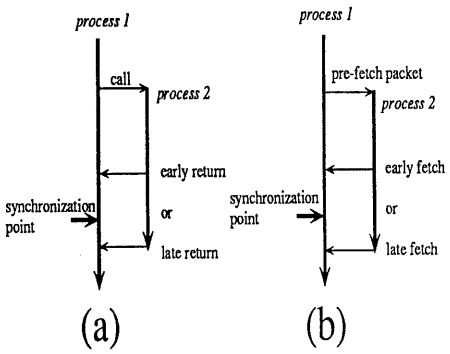


図 1: CODA でのプロセス間同期

は優先度キューに保持され、実行中のプロセスの優先度と比較される。優先度キューに保持されたプロセスの優先度が実行中のプロセスの優先度よりも高くなった場合には、実行中のプロセスを中断し、より高い優先度のプロセスを実行する。これにより、常に最高優先度のプロセスを実行する。

プロセスは共有変数を介して互いに通信を行なうこともできる。共有変数は各プロセッサのローカルメモリ上に割り付けられ、Iストラクチャと同様の同期操作を行なう。共有メモリ上のデータが定義されないうちに読み出しを行なおうとする場合、読み出し要求はメモリ上に保存され、データの書き込みが起こるまで待たされる。共有変数に対する同期操作は、単一書き込み単一読み出しをハードウェアでサポートし、複数の読み出し要求が生じた場合にはトラップを発生する。

同期待ちの頻度を減らすため、コンパイラはデータのアドレスが確定した後すぐにプリフェッチ要求を出す様にすることができる。同期ポイントに達する前にプリフェッチ要求に対する値が返ってきていれば、実行中のコンテクストは中断することなく処理を継続できる (図 1(b))。

### 3.3 優先度の管理

CODA では、プロセスのみでなく、割り込み処理や通信パケットに対しても、デッドラインに応じた優先度を付与し、これらを一元的に管理しようとしている。そのためには優先度には充分な解像度が必要であり、また、優先度の判定を行なう頻度も高

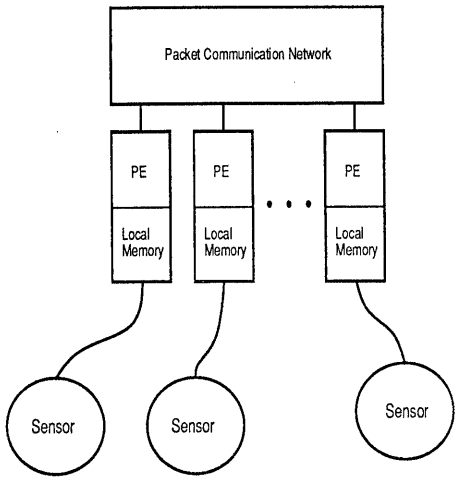


図 2: CODA の全体構成

くなるため動的な管理を行なうことは大きなオーバヘッドを伴うことになる。

優先度フィールドに、時刻をそのまま表現できるだけの幅を持たせることができれば、デッドラインをそのまま優先度として用いることができる。デッドラインは、一旦決定されればその処理 (プロセス実行や割り込み処理、通信パケット処理など) が終了するまで変化しないと考えられるため、優先度の動的な管理が不要となり、優先度の高い処理から順に実行していけば良いことになる。

## 4 CODA プロセッサのアーキテクチャ

CODA は、128 台のプロセッシングエレメント (PE) とセンサがパケット通信ネットワークを介して結合された並列処理計算機である (図 2)。各 PE はローカルメモリとセンサ / アクチュエータ用のデータ入出力ポートを有する。CODA は物理的な共有メモリは持たないが、論理的な共有変数は各 PE のローカルメモリ上に保持される。本節では CODA の PE のハードウェア構成について述べる。

## 4.1 CODA プロセッシングエレメントの概要

CODA プロセッサでは、デッドラインを直接表現できる 32bit の幅を優先度に持たせる。これにより、優先度の動的な管理を省き、優先度管理をハードウェア化する。そして、プリエンプションに伴うコンテクスト切替えを高速化することにより、デッドラインスケジューリングを効率良く実現する。また、並列プロセッサにおいて実行時間の予測性を損なう要因と考えられる通信と同期が、命令実行を妨げないように設計されている [3]。

CODA は、命令実行パイプライン (IP) とパケット処理パイプライン (PP) の 2 本のパイプラインを有する (図 3)。PP は、パケット受信を行なう受信部 (Recieve pipeline) とパケット出力を行なう送信部 (Transimit pipeline) からなり、IP の命令実行に影響することなくパケットの入出力を行なう。入力パケットのデータは 6 ポートレジスタ (2 書き込みポート、4 読み出しポート) ないしメモリに直接書き込まれる。レジスタファイルは 2 セット用意されており、IP が「表」のレジスタセットを使用してプログラムを実行している間に、「裏」のレジスタセットに次に実行される (プロセス優先度により推定される) プロセスをあらかじめロードしておくことによりコンテクスト切替えの高速化を図っている。

同期操作のオーバヘッドは、同期の伴うコンテクスト切替えの他に同期操作そのものの効率も考慮されなくてはならない。CODA ではレジスタ読み出しの際に同期判定を行なうことにより、余分なメモリアクセスを行なうことなく、また、パイプラインブレークを発生することなく同期操作を行なうことが可能である [7]。

## 4.2 パイプライン構成

IP(図 4) は、優先度判定段 (P)、命令フェッチ段 (F)、命令デコード段 (D)、実行段 (E)、レジスタ書き込み段 (W) の 5 段のパイプライン構成をとる。P 段では実行中のプロセスの優先度とプロセス優先度キュー中の最高優先度を比較し、プロセス優先度キューの優先度の方が実行中のものより高い場合にはプロセスを切り換える (プリエンプション)。

レジスタは一語毎に presence bit を持ち、レジスタ上のデータが存在する (定義されている) かどうかを示す。presence bit は D 段でのレジスタ読

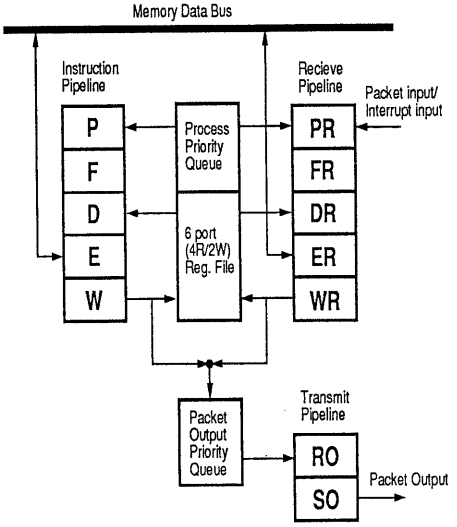


図 3: CODA PE の構成

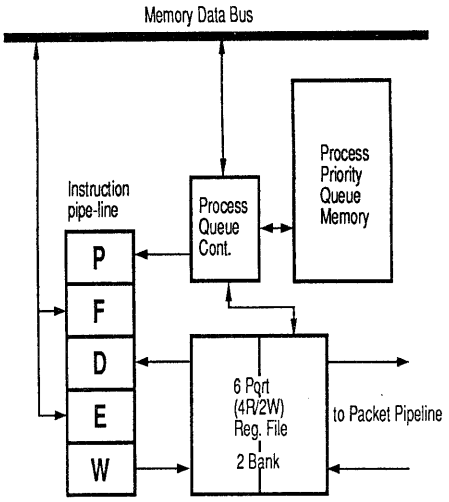


図 4: 命令パイプラインの構成

み出しの際に同時に検査され、presence bit が *off*の場合にはトラップが発生し続く 2 サイクルはコンテクスト切替えに使われる [7]。コンテクスト切替えに使用される 2 サイクルは、プロセス優先度キューとプログラムカウンタの間で実行アドレスの転送を行なうために最低限必要な時間である、同期操作そのものはパイプラインにバブルを発生することなく実行される。W 段は E 段の演算結果をレジスタかパケット出力キューに書き込む。パケット出力キューは IP からはレジスタとして扱われ、書き込みによって PP の送信部を起動しパケット出力を行なう。

PP(図 5) は 5 段の受信パイプライン部と 2 段の送信パイプライン部から構成される。パケット優先度判定段 (PR)、パケットフェッチ段 (FR)、パケットデコード段 (DR)、パケット処理段 (ER)、パケット書き込み段 (WR) の 5 段のパイプラインは、IP と同様に命令実行を行なうことができ二番目に高い優先度のプロセスを実行する。そして、命令挿入機構を用いることによりパケットの受信および割り込みを処理する。パケット出力は、出力優先度判定段 (RO) と出力段 (SO) の 2 段で処理される。

パケット入出力及び割り込み処理を PP が実行することにより、IP が最高優先度のプロセス実行に専念できるようにする。プリエンプション時のコンテクスト切替えの高速化とともに、最高優先度のプロセスの実行時間の予測性を改善しようとするものである。

### 4.3 優先度キュー

CODA は、プロセスキューとパケット出力キューの二つの優先度キューを持つ。CODA の優先度キューは上位 8 優先度の挿入と最上位優先度の取り出しを一定時間で実行できるように設計されている。プロセスキューの最上位の優先度は、常に実行中のプロセスの優先度と比較され、プリエンプションが必要な場合は P ステージにリクエストを送る。

### 4.4 ネットワーク

CODA ではプロセッサ間を多段ルータネットワークで結合する。ネットワーク上でも優先度制御を行なうが、ネットワーク上での優先度逆転が問題となる。そこで、優先度逆転をを解消する優先度先送り方式 (BPF) の評価を行い [5]、LSI 化における得失を検討している。

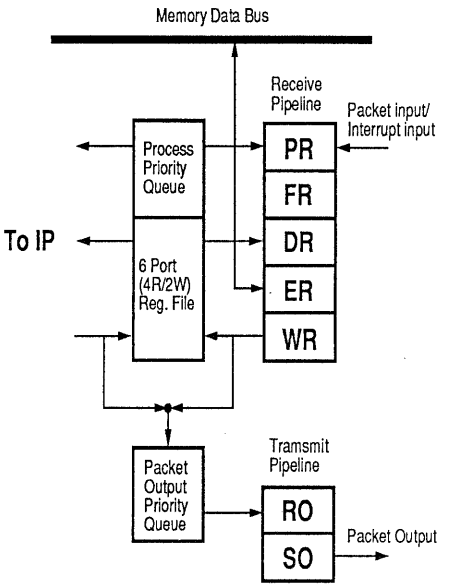


図 5: パケット処理パイプラインの構成

## 5 おわりに

CODA は 32bit の優先度フィールドを持つことにより優先度の動的管理を不要とするとともに、全ての処理をデッドラインに応じた優先度にしたがって行なうようにした。また、優先度制御されたネットワークを用いることにより、通信時間の予測可能性も向上させている。しかし、プロセッサ間で負荷の不均衡が生じた場合には、並列化による優位性が損なわれることになる。しかも、不均衡を解消するためにプロセスマイグレーションを行なうことは、マイグレーションに必要な時間を見積もることができない限り、処理時間の予測可能性を損なうことになる。そこで、最適な負荷分散をあらかじめ得ることが、実時間処理では特に重要な課題となる。自動負荷分散機構 [1] に優先度を考慮した機能を付加し、低オーバヘッドで負荷の均一化を図ることを検討している。また、プログラマによるスタティックな負荷分散も大きな助けとなると考えられる。

## 謝辞

本研究を遂行するにあたり御指導、御討論いただいた弓場敏嗣情報アーキテクチャ部長ならびに計算機方式研究室の同僚諸氏に感謝致します。

## 参考文献

[1] K. Hiraki, S. Sekiguchi, and T. Shimada. "Load Scheduling Schemes Using Inter-PE Network". *Systems and Computers in Japan*, 18(1):55–65, 1987.

[2] 石川正俊. "センサフュージョンの現状と課題" *SICE '90* 学術講演学会予稿集, Vol.1, July 1990.

[3] 西田健次, 戸田賢二, 坂井修一, 平木敬, 島田俊夫. "実時間用並列処理計算機 CODA-r のアーキテクチャ" *SWOPP '91 (IPSJ ARC 89-21)*, pages 151–157, July 1991.

[4] T. Shimada, K. Nishida, and K. Toda. "Real-Time Parallel Architecture forSensor Fusion". *To be appear in Journal of Prallel and Distributed Computing*, June 1991.

[5] 戸田賢二, 西田健次, 坂井修一, 平木敬, 島田俊夫. "優先度先送り方式を用いたオメガネットワークの性能評価" 電子情報通信学会研究会資料 *CPSY 91-53*, pages 9–14, December 1991.

[6] K. Toda, K. Nishida, Y. Uchibori, S. Sakai, and T. Shimada. CODA: A Multiprocessor Architecture for Sensor Fusion. In *Proceedings of The Fifth International Symposium on Intelligent Control*, pages 261–266, September 1990.

[7] K. Toda, K. Nishida, Y. Uchibori, and T. Sakai, S. Shimada. Parallel Multi-Context Architecture with High-Speed Synchronization Mechanism. In *Proceedings of the Fifth International Parallel Processing Symposium*, pages 336–343, April 1991.