計算機アーキテクチャ 79－4
（１９８９．１１．２０）

並列推論マシン PIE64 の推論ユニット間通信

清水 剛，小池 汎平，田中 英彦
東京大学 工学部

概要

我々が 開発を 進めている並列推論マシン PIE64 のネットワーク ・ インタフェス ・ プロセッサ（ NIP ）は、 PIE64 の各推論ユニット（ IU ）内部と 相互結合網とのインタフェスを行なう プロセッサである。

NIP は、 データ転送やプロセス間同期、 および、 分散メモリ環境における ガベージ・コレクション支援のコマンドを ネットワークを介して実行し、 PIE64 での並列論理型言語 FLENG の実行処理における 並列処理機能を支援する。

本稿では、 NIP の内部ハードウエア構成、 及び NIP 各部での処理動作をふまえ、 PIE64 の 相互結合網を介して行なわれる 2 個の NIP 間での 通信のプロトコル、 性能などについて述べる。

INTER-PE COMMUNICATION OF THE PARALLEL INFERENCE MACHINE PIE64

Takeshi Shimizu, Hanpei Koike, Hidehiko Tanaka

Department of Electrical Engineering, Faculty of Engineering,
University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo 113, Japan

Abstract

The Network Interface Processor (NIP) of PIE64 is a processor used for the interface between each Inference Unit (IU) and the interconnection network of PIE64. NIP executes three types of commands between 2 IUs through the interconection network. One is to transfer data, another is to exchange process syncronization messages and to manage goal suspension information directly by hardware, and the other is to support global garbage collection under the distributed memory systems of PIE64.

This paper shows the internal hardware organization of NIP, reports how it works, and describes the commnuication protocol on the interconection network of PIE64.

1 はじめに

現在、我々の研究室では、並列推論マシン PIE64 [1] [2] の開発をすすめている。PIE64 は 64 台の推論ユニット (Inference Unit: IU) が 2 系統の自動負荷分散機構付き多段ネットワーク [3] [4] によって結合された構造を持ち、並列論理型言語 Fleng [5] および、その上位言語である Fleng++ [6] を実行する。

各 IU には、IU 内部と相互結合網とのインタフェスを行なうネットワーク・インタフェス・プロセッサ (Network Interface Processor: NIP) が用意される。NIP は PIE64 で行なわれる並列推論処理において、2 つの IU 間でのデータ転送、Fleng の論理変数を用いたプロセス間同期、PIE64 の一括ガベージ・コレクション [7] といった並列処理機能をハードウエア・レベルで直接支援する [8] 。

本稿では、NIP の内部ハードウエア構成、及び NIP 各部での処理動作をふまえ、PIE64 の相互結合網を介して行なわれる 2 個の NIP 間での通信のプロトコル、性能などについて述べる。

2 並列推論マシン PIE64

PIE64 は、64 台の推論ユニットを、2 系統の自動負荷分散機構付きの多段ネットワークにより結合した構成を持ち、基本言語として並列論理型言語 Fleng および その上位言語である Fleng++ を 実行する。

PIE64 の 2 つの相互結合網は、2 系統のネットワークは、プロセス分散網 (Process Allocation Network: PAN)、および、データ配置網 (Data Allocation/Accessing Network: DAN) と呼ばれ、どちらも 自動負荷分散機能を備えた 64×64 の回線交換式の多段網である。この 2 つネットワークにはハードウエア上の差異はない。

各 IU には Fleng の実行処理を受け持つ推論プロセッサ UNIREDII[9]、IU 間通信 ・ Fleng のプロセス間同期を支援する NIP、IU の管理や入出力処理を担当する管理プロセッサとしての SPARC がある。各ネットワークには、ネットワークへの 接続要求を 出す、いわば、ネットワークへの出口にあたる マスタ NIP と、ネットワークからの要求に従って動作する、いわば、ネットワークからの入口にあたる スレーブ NIP が接続されるので、各 IU には計 4 個の NIP が実装されることになる。

これらの計 6 個のプロセッサ (UNIREDII、NIP ×4、管理プロセッサ) が協調動作し、Fleng の実行をすすめるが、この IU 内部のプロセッサ間通信は、高速・双方向で 32 ビット幅の専用のコマンド・バスを介して、コマンド / リプライのやり取りによって行なわれる。

また、これらのプロセッサは、3 系統の同期メモリ・バスを介して、4 個のローカル・メモリ・バンクを共有する。

3 ネットワーク ・ インタフェス ・ プロセッサ

NIP は 以下のような並列処理支援のための機能を IU 内部に提供する。IU 内部のプロセッサは、マスタ NIP に対してコマンドを発行することにより、これらの機能を利用できる。

- データ転送処理
- サスペンション・レコード・リストの管理によるプロセス間同期処理
- 一括ガベージ・コレクション支援

このうち、データ転送と 一括ガベージ・コレクション支援に関しては、マスタ NIP と ネットワークを介して 接続された相手側の スレーブ NIP とが協調動作して処理を実行する。

また、プロセス間同期処理のうちの リスト処理にあたる部分は、接続先のスレーブ NIP で行なわれる。IU 内部で ローカルにサスペンション・レコード管理機能を 利用する 場合には、スレーブ NIP にコマンドを送ることもできる。

データ転送処理においては、以下のような特徴を持つ。

- Fleng のデータタイプを考慮した転送命令形態
- 自動負荷分散機構を利用したデータ転送
- 構造体領域内の未束縛論理変数領域の一意性を保証

第 3 項の機能は、リストやベクタ中に UNDEF 型のデータ・タグを持つ 未束縛論理変数の実体が 埋め込まれていた場合、データ転送時に

その要素を、もとの未束縛論理変数の実体を指す変数型のポインタとして すり替えて転送するものである。

プロセス間同期処理については、以下のコマンドが用意されている。

- サスペンドを 起こした コンテクストを、その原因となった 未束縛論理変数の サスペンション・レコード・リストに 登録するための suspend コマンド
- 未束縛論理変数に値の束縛を試み、その変数のサスペンション・レコード・リストに登録されていたコンテクストをアクティベイトするための bind コマンド
- リモートに実体のあるコンテクストがアクティベイトされたことを、そのリモートの IU に通達するための activate コマンド
- サスペンション・レコード・リストを受け取り、それに登録されている全てのコンテクストに対する activate コマンドを発行するための activates コマンド

また、一括 ガベージ・コレクション支援に 関しては、以下の機能が 用意されている。

- リモートのマーキングを支援する mark コマンド
- コンパクション・フェーズでの リモート・ポインタの書き戻しをする restore コマンド

4 NIP のハードウエア

NIP の 内部ハードウエア構成、および、各部の処理動作について述べる。

4.1 NIP の内部ブロック構成

NIP の内部ブロックは、大きく分けて以下の 4 つのブロックから成る。

- コマンド・バス・インタフェス部
- メモリ・バス・インタフェス部
- データ転送処理部
- プロセス間同期処理部

コマンド・バス・インタフェス部は IU 内の他のプロセッサ (管理プロセッサ SPARC, 推論プロセッサ UNIREDII, 他の NIP) との間で、コマンド / リプライの送受信を行なう。また、性能測定用のカウンタ、および、プロセッサの動作モード設定用のレジスタを持つ。

メモリ・バス・インタフェス部は、IU 内部の 3 本のメモリ・バスと接続され、そのうちの 1 本に対しては IU 内ローカル・メモリに対する読み書きを行ない、他の 2 本に関しては、ロック・アドレスの監視をしている。また、負荷データやメッセージ受信に使用される ヒープ・メモリ管理用のレジスタを持つ。このヒープ・メモリ管理用のレジスタは 1 組の予備ヒープ・メモリ用のレジスタを含み、2 組が用意されており、スレーブ NIP で使用される。

データ転送処理部は、ネットワークを 介したデータ転送処理、及び、一括 ガベージ・コレクション 支援の コマンドを 実行する。また、ネットワークの自動負荷分散機能に対応して、ネットワークを通じて流れてくる 8 ビットの負荷分散情報を 監視するための コンパレータと、ネットワークを介したデータ転送時のデータ列の垂直パリティを生成するパリティ・ジェネレータを持つ。データ転送時に使用される アドレス生成用のカウンタと 転送長を計量するカウンタは メモリ・バス・インタフェス部にある。

プロセス間同期処理部は、スレーブ NIP において、サスペンション ・ レコードの 管理のためのリスト処理を行なう。

これらのブロックはそれぞれ独自にシーケンサを持ち、協調動作をするとともに、並行処理を可能としている (図1)。

4.2 データ転送処理ブロック

データ転送処理ブロックのデータ・パスを図 2 に示す。データ転送処理においては、

1. 最大効率時に、1 クロック 1 ワードのデータ転送が可能となること
2. IU 内 ローカル ・ メモリには、アービトレーション＋ アドレス出力、データ転送の 2 フェーズからなる パイプライン的なアクセスがなされること
3. ネットワークに約 50 ns の遅延が存在すること
4. 構造データ内に埋め込まれた未定義変数領域の一意性を保証するため、この変数領域

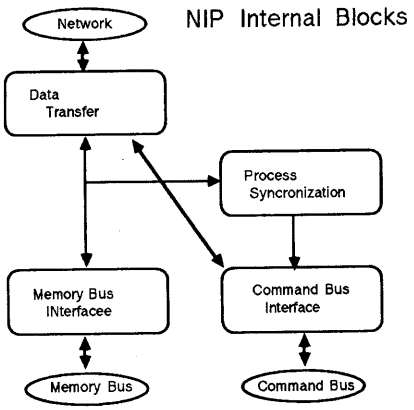

図 1: NIP の内部ブロック構成

をポインタに変換して転送できること

等の要求及び制約を満たすために、データ読みだし側の NIP で 3 段、ネットワークを間にはさんでデータ書き込み側の NIP で 3 段のパイプラインを構成する形になっている。

読みだし側での 3 段は、図 2 中の $BUF0 \rightarrow BUF1 \rightarrow NOB$ のパスで、前 2 段は、それぞれ、アドレス出力 と データ・フェッチのステージに相当し、その時点でのリクエスト・アドレスが前 2 段で保持され、前記 4 の変数型ポインタへの変換処理に利用される。このポインタとデータの置き換えは、3 段目のネットワークへの出力段に移行する $BUF1 \rightarrow NOB$ の段階で行なわれる。

書き込み側での 3 段は、図 2 中の $NIB \rightarrow BUF2 \rightarrow NOB$ のパスで、それぞれ、ネットワーク からの データ受信、メモリ・リクエスト、データ書き込みの ステージに 相当する。このデータの流れの様子を 図 3 に示す。

パイプライン的に 読み書きされる メモリを、ネットワークを挟んで接続し、両側の制御回路の状態を 1 クロックの遅延[1] ある信号線を用いて伝えあって分散制御しているため、書き込み側でメモリバスのアクセス権を確保できなかった場合にパイプラインの乱れが生じ、読みだし側がオーバー・ランをしてしまう。

[1]前述のとおり、ネットワークの信号遅延自体は 半クロック程度であるが、同期回路の設計上、ネットワークを 1 クロックの遅延とみなしている

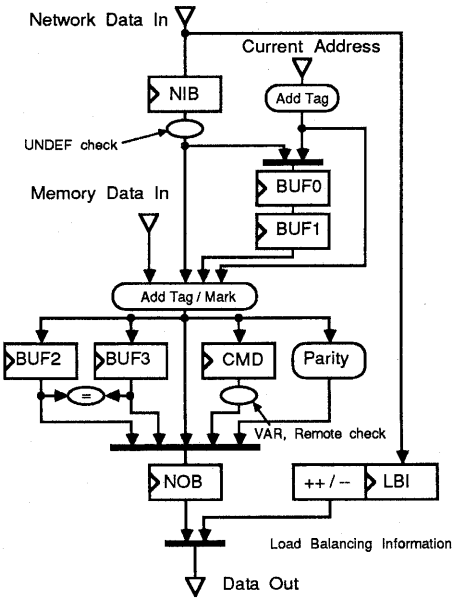

図 2: データ転送処理ブロックのデータパス

図 3に示した以外のデータ転送処理部のレジスタは全て、パイプラインの乱れを解消するためのバッファとして利用される。このバッファリングは、送信側で 2 ワード、受信側で 3 ワード保持される形にした。この構成にすれば、もっともゲート数が少なく、かつ、他の機能の実装と併せて考慮した場合に自然に実装できるからである。

また、一括 ガベージ・コレクション 支援時のリモート・ポインタの マークと、コンパクション後のポインタ書き戻し処理の場合には、一部のレジスタがハッシング及びキャッシングのために利用される。

これらの他に、プロセス間同期処理のうち、ネットワーク側から要求された bind 処理に対して、未束縛論理変数からのサスペンション・レコード・リストの取り出しと値の束縛のアトミックなオペレーションについては このブロックで行なわれ、サスペンション・レコード・リストのリスト処理に 必要な パラメータのみが コマンドと 同一の形で プロセス間 同期処理部に 送られる。

4.3 プロセス間同期処理ブロック

プロセス間 同期処理ブロックの データ ・ パスを 図 4 に示す。

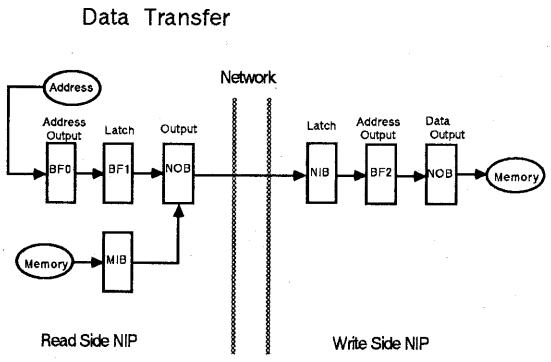

図 3: データ転送処理時のデータの流れ

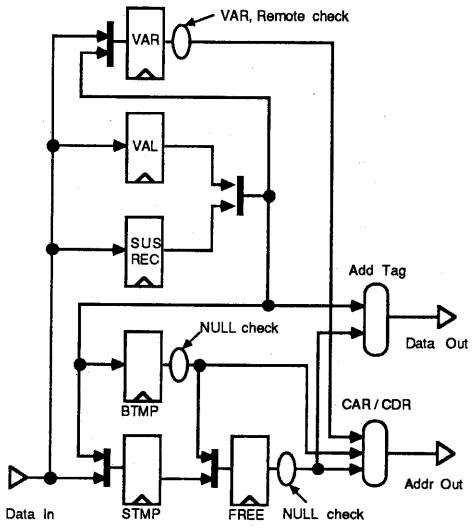

図 4: プロセス間同期処理ブロックのデータパス

このブロックでは、 プロセス間同期処理のうちの、 suspend 処理における サスペンション・レコード ・リストへの コンテクストの 追加登録、 および、 activates 処理における サスペンション・レコード・リストの 回収を 主として行なう。

サスペンション・レコード・リスト用のメモリは、 フリーリストの形でスレーブ NIP によって管理されている。

このブロックは、 これらの コマンドのための引数レジスタ (VAR, VAL)、サスペンション・レコード・リストの リスト処理 に 使用される テンポラリ・レジスタ ($SUSREC, BTMP, STMP$)、 フリーリストを 保持する レジスタ ($FREE$) を持つ。 また、 リスト処理時にポインタに対して必要に応じてタグの書き換えを行なう。

PIE64 のローカル・メモリ・バスは パイプライン的にアクセスが行なわれるため、 あるメモリ・アクセスの結果を見てから次のメモリ・アクセスを行なうような、 シーケンシャルな、 メモリ上の データに 対する 処理を する場合には スループットがあがらない。

そこで、 スレーブ NIP では、 リスト処理時の 1 つのセルの処理に必要な 4 回から 5 回のメモリ・アクセスに対し、 アクセス順序の制約を崩さない範囲で 動的にアクセス順序を変え、 PIE64 のメモリ・バス・プロトコルに適合した高速なリスト処理を実現している。

5 通信プロトコル

PIE64 において、 相互結合網を介して NIP が行なう推論ユニット間通信のプロトコルについて述べる。

PIE64 の ネットワーク自体は マルチ・キャストを サポートするよう設計・製作がされているが、 NIP は マルチ・キャストの機能を 使用せず、 通信は すべて 1 対 1 通信で行なわれる。

通信は以下の手順によって進められる。

1. マスタ NIP の接続要求
2. スレーブ NIP の接続受理
3. マスタ NIP からコマンドの引数の送信
4. スレーブ NIP からのパラメータ送信
5. データ転送
6. 回線切断

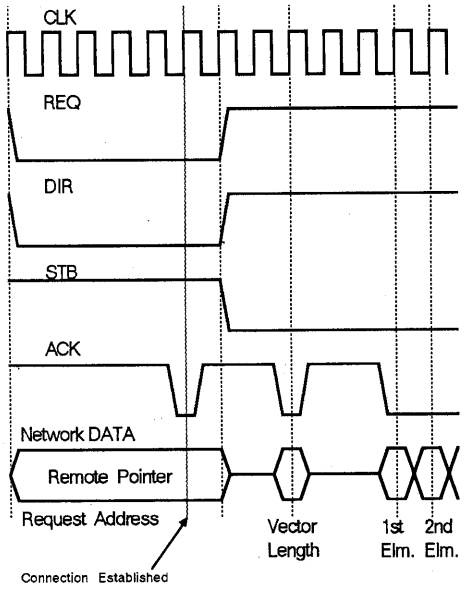

図 5: **readn** コマンドによるベクタ転送時の通信プロトコル

実行されるコマンドの種類によっては、 省略されるフェーズもある。

また、 リモートな IU へのデータ書き込みを行なう場合のパラメータ転送終了後 (データ転送前) と 全てのコマンドでのデータ転送終了後 (回線切断前) に、 それ以前までに ネットワーク上でやり取りしたデータ列の垂直パリティを、 データ書き込み側に送信し、 チェックを行なう。ネットワーク上での 通信プロトコルの 例として、 **readn** コマンドによるベクタ読みだし転送時のタイミングを図 5 に示す。

ネットワークを介した 2 つの NIP は、 **STB** と **ACK** の 2 本の信号線により お互いの状態を伝えあい、データの転送を行なうが、 1 クロック 1 ワードの転送をするために、 1 ワードごとのハンドシェークをするのではなく、 これら 2 本の信号は、 そのサイクルで、データを送信していることと、 データの受信が可能な状態であることを表した制御回路の状態信号となっている。

図 5 の例では、 マスタ NIP 側からの接続要求がネットワークを通り、 スレーブ側で受け取られたのち、 ベクタの先頭に格納されているベクタ長が読み出され、 それがスレーブ側から 転送パラメータとして送信された後に、 パイプライン的な転送状態に入る様子を表している。

6 デッドロックの回避

プロセス間同期処理においては、 スレーブ NIP から マスタ NIP への **activate** コマンドあるいは **suspend** コマンドが発行される状況があるので、 ストアアンドフォワード・デッドロックが発生する危険がある。

すなわち、 いま、 簡単に説明するためにネットワークは 1 系統しか使っていないものとしたうえで、

1. **bind** コマンドが IU 0 と IU 1 で立て続けに、ほぼ同時にそれぞれの IU で 複数回生成されたとする (この場合、 両側で計 6 回以上) 。
2. 実体が IU 0 にある未定義論理変数でサスペンドしていたコンテクストの実体は IU 1 にあり、 IU 1 の未定義論理変数でサスペンドしていたコンテクストの実体は IU0 にあるものとする。

とすると、 以下のような状態で デッドロックに おちいる。

- Master NIP は、 **bind** コマンドのために ネットワークに接続要求を出しており、 相手側 Slave NIP の接続応答待ちで Busy.
- Slave NIP のデータ転送処理部は その 1 つ前に 受理した **bind** コマンドから 派生した **activates** コマンドをプロセス間同期処理部へ 送ろうとして Busy.
- Slave NIP のプロセス間同期処理部は さらにその 1 つ前に受理した **bind** で生じた **activates** コマンドによる リスト処理を行なっているが、 その過程で生じる **activate** コマンド発行のために、 Master NIP の空きを待って Busy.

この状況は **bind** コマンドと、 すでに 束縛されて しまっている 変数に 行き違いで発行された **suspend** が含まれていても 起こり得る。 これを図 6 にしめす (実線部分)。

この回避のための方法として、スレーブ NIP において、 プロセス間同期処理部が BUSY で、データ転送処理部が 送ろうとしている **suspend** コマンドか **activates** コマンドを受け付けることができず、 かつ、 データ転送処理部にネットワークからの接続要求が到着した場合に、 この

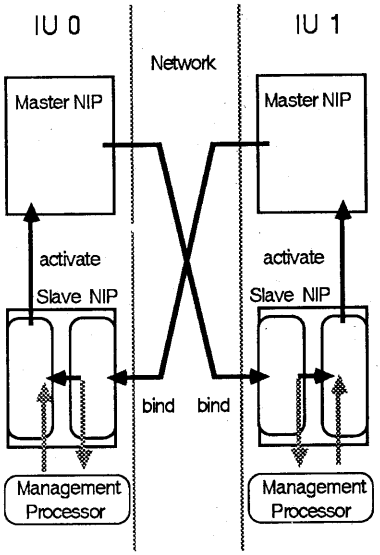

図 6: デッドロックの回避 (1)

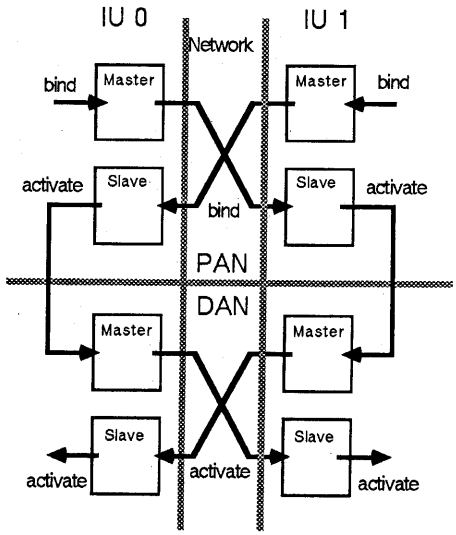

図 7: デッドロックの回避 (2)

プロセス間同期処理部に送るべきコマンドを コマンド・バスを通じて管理プロセッサ側に廻し、あとで、 管理プロセッサから スレーブ NIP にたいして 再発行するようにしている。スレーブ NIP 内部の調停は、 ネットワークと コマンド・バスとでは、 コマンド・バス側の優先順位が高くなるように実装してある。

もう 1 つの方法としては、 PIE64 に 2 系統のネットワークがあることを ファームウエア上で 積極的に利用する方法が挙げられる。

すなわち、

1. リモート変数に対する bind コマンド 及び suspend コマンドを 発行する マスタ NIP を、2 系統あるネットワークのうちの一方のもの (例えば PAN) に限るよう、UNIREDⅡと管理プロセッサに指示する。
2. bind / suspend コマンドを実行するスレーブ NIP が activate コマンドを発行する場合は、もう一方のネットワーク (例えば DAN) の マスタ NIP を用いるように スレーブ NIP をモード設定する。
3. bind コマンドでバインドする値が 変数であった場合、 コンテクストの登録し直しのために、 通常は suspend コマンドを発行するが、そこで代わりに activate コマンドを発行するように スレーブ NIP をモード設定する。また、 suspend コマンドにおいて、変数が 行き違いで別の変数に バインドされてしまった時も suspend コマンドではなく activate コマンドを発行するように設定する。 activate コマンドで送られたコンテクストは、 UNIREDⅡで処理されて 結局サスペンドするのだから、 若干のオーバーヘッドを生じるが、 論理的に誤っている訳ではない。 この条件により、NIP のコマンドから NIP のコマンドが起動される連鎖は、 高々 2 段までに抑えられ、 デッドロックが回避される。

これを 図7 にしめす。 この方式では bind / suspend コマンドの実行のスループットが 半分に落ちてしまうことで問題がある。

7 性能

基本的なコマンドの実行クロック数については、表 1 のようになる。 NIP の動作モード設定により若干変化するが、 最小時間で動作する場合をしめす。 ここでは、ネットワーク、及び、メモリ・バスのアクセス時に衝突がおこらない場合を考える。 また、クロックは、 NIP の基本クロックである 10MHz を想定する。

この表中の “Master” は、コマンド・バス上にコマンドのリクエストが出されてから、そのリプライが返るまでのターン・アラウンド・タイムをあらわす。この時点で、すでにネットワークの解

表 1: NIP の基本コマンド実行時の処理時間

コマンド	Master	Network	Slave
read1	14	11	7
read2	16	12	8
readn	17+n	13+n	9+n
write1	16	14	9
write2	20	18	14
writen	22+n	20+n	16+n
suspend	15	13	10
bind	19	16	6n+12

単位 クロック

放が行われており、マスタ NIP は新しいコマンドを受け付けることができる。"Network" は、このオペレーションによってネットワークが占有される期間であり、また、"Slave" は、マスタ NIP にコマンドが送られてから、スレーブ NIP での処理がすべて終了するまでのクロック数を表す。プロセス間同期コマンドでは、サスペンション・リストのリスト処理のため、ネットワーク接続解放後も処理が継続する。bind コマンド欄の n は、サスペンション・リストのリスト長をあらわす。

8 終りに

PIE64 の NIP のハードウエア構成、内部データ・バス及び処理動作、並びに、ネットワークを介した通信プロトコルについて述べた。

現在、シミュレーションによる最終チェックを行なっている。

今後の課題としては、サンプルの性能評価、および、PIE64 の IU の設計・実装に伴って、実際の相互結合網を使用した接続試験があげられる。

なお、本研究は文部省の特別推進研究 No. 62065002 の一環である。

参考文献

[1] Koike, H. and Tanaka, H.: *"Multi-Context Processing and Data Balancing Mechanism of the Parallel Inference Machine PIE64"*, Proc. of Fifth Generation Computer Systems, ICOT, Nov. 1988.

[2] 小池, 田中: "並列推論マシン PIE64 の概要", 情報処理学会第 37 回 全国大会, 5N-4, Sep. 1988.

[3] 高橋, 田中, "並列推論マシン PIE64 におけるインターコネクションネットワークの作成と評価", 情報処理学会 計算機アーキテクチャ 研究会, 76-1, May, 1989.

[4] Koike H., Takahashi E., Yamauchi T. and Tanaka H.: *The High Performance Interconnection Network of Parallel Inference Machine PIE64*, Computer Architecture Symposium IPS Japan, 1988.

[5] Nilsson, M. and Tanaka, H. : *"FLENG Prolog - the Laguage which turns Spuercomputers into Prolog Machines"*, Proc. of Japanese Logic Programing Conference '86, ICOT, June, 1986.

[6] 中村, 小中, 田中: "並列論理型言語 FLENG に基づいた並列オブジェクト指向言語 FLENG++", 日本ソフトウェア科学会, オブジェクト指向計算に関するワークショップ WOOC '89, 1989.

[7] 小池, 許, 田中, "分散メモリにおけるガベージコレクション", 日本ソフトウエア科学会第 6 会大会, C5-1, Oct. 1989.

[8] 清水, 小池, 島田, 田中, "並列推論マシン PIE64 のネットワーク・インタフェース・プロセッサ", 並列処理シンポジウム '89 A2-2, 情報処理学会, Feb. 1989.

[9] 島田, 下山, 清水, 小池, 田中: "推論プロセッサ UNIREDIIのアーキテクチャ", 情報処理学会 計算機アーキテクチャ研究会 77-2, July, 1989.