計算機アーキテクチャ 87－5
（１９９１. 3. １１）

PIE64 のネットワーク・インタフェス・プロセッサ LSI の詳細

清水 剛, 小池 汎平, 田中 英彦
東京大学 工学部

要旨

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、この NIP の最終仕様と実際に設計したハードウエア構成、 及び詳細設計に基づく予測性能を示す。

DETAILS OF THE NETWORK INTERFACE PROCESSOR FOR 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 interconnection network. One is to transfer data, another is to exchange process synchronization 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 final specification of NIP, details of the internal hardware organization, and the result of the estimated performance.

1 はじめに

並列推論マシン PIE64 [6] 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 [1] [2] によって 結合された構造を 持ち、 並列論理型言語 Fleng [3] および、 オブジェクト指向を取り入れたその上位言語である Fleng++ [4] 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 [5] 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 [7] 。

本稿では、 NIP の最終仕様と実際に設計したハードウエア、詳細設計に基づく性能予測を示す。

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
- ベクタ転送時におけるデレファレンスを含む転送
- 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、ベクタ型のポインタのチェーンを転送時に解消するものであり、後述する。

第 4 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 / コピーの継続を支援する mark コマンド
- GC の最終フェーズでの リモート・ポインタの書き戻しをする restore コマンド

この二種のコマンドを利用することにより、コンパクション方式だけでなく、コピー方式のガベージ・コレクタを実装することができる。

4 仕様拡張とメンテナンス機構

4.1 ベクタ転送命令の拡張

PIE64 上での Fleng++ の効率の良い実装を考慮した場合、アクセス・コストの低減やメモリ資源を有効的に利用するために、ベクタのような構造データが一意性を保ちながら移動する事を許す実装方法が考えられる。このとき、PIE64 の分散共有メモリ中にベクタ型のポインタによるチェインが生じる。このリモートに張られたリンクをソフトウエアで解消すると、オーバーヘッドを生じてしまう。

そこで、PIE64 において、ある構造データをローカルメモリのスクラッチ領域へ読みだすときに、ベクタ型のポインタをデレファレンスし、構造データのマイグレーションによるアクセスコストの増大を抑えるように仕様を変更した。

ここでは、このベクタ型ポインタのデレファレンスに対応したより一般的なベクタ転送の機構について述べる。

複数のプロセスによって共有される構造データを、各プロセスのそのデータへのアクセス頻度や、各推論ユニットのメモリ使用量に応じて、より適した位置へ一意性を保ちつつ移動をする場合、元の位置から新しい位置を参照できるようなリファレンスを設ける必要がある。 このとき、このリンクは通常リモートに張られ、デレファレンスのコストが高く、何らかのアクセス機構が必要となる。

ベクタを移動するには、その先頭ワードに移動先を指すベクタ型のポインタを上書きしてしまうのが、もっとも単純でよく (図 1)、これは、PIE64 のデータ構造において、ベクタの先頭を指すベクタ型のポインタのチェインを許すようにすればよい。

NIP には変数用の deref コマンドが用意されてお

り、変数へのポインタのチェインをたぐることができるので、このシーケンスを利用して、ベクタ型ポインタをインビジブル・ポインタとして拡張し、転送系にベクタポインタのチェインの処理を行えるようにした。

通常のベクタ読み出しのシーケンスは、

1. 読み出す対象のベクタを指すリモートアドレス (ベクタ型のポインタ) と、ローカルなスクラッチ領域のアドレスを NIP に渡す。
2. リモート・アドレスの指す先をベクタの先頭と解釈し、ベクタ長ワードとして取り込む。
3. 転送が終了すると、読み出したデータの格納されたスクラッチ領域のアドレスが転送終了同期のためのリプライ・ワードとして転送を要求したプロセッサに返される。

であったが、これに、ベクタ型ポインタのチェインをデレファレンスするステップを組み込む。

チェインをたぐる処理は基本的に変数用の **deref** コマンドのメカニズムと同じであり、リモートポインタが与えられたならばネットワークからデータをフェッチし、ローカルポインタが与えられたならばメモリからフェッチする。 この時にフェッチしたデータのタグを調べ、ベクタの先頭ワードについたオブジェクト・タグが現われるまでこの操作を繰り返す。ベクタの先頭ワードをあらわすタグを持ったデータをフェッチしたら、それを転送の対象であるベクタのベクタ長と見なし、転送カウンタへのロード、チェックをおこなう。 この手続きは、**deref** コマンドを実行する NIP 内部のシーケンサに若干の変更を加えることで解決できる。 また、通常のベクタ転送処理における転送カウンタの操作とタグ・チェックを並行して行なえるため、オーバーヘッドにはならない。

最終的に発見したベクタ本体がリモートにあれば、本来の構造データの読み出し操作を行ない、スクラッチ領域に転送するが、デレファレンスの結果として、本体がローカルに存在するケースもありうる。この場合には、

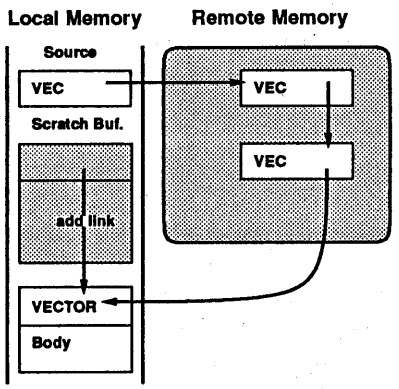


図 1: ベクタ型ポインタのチェインと転送の例

- 実際にデータのあるローカル・アドレスを同期用リプライワードとして返し、余計なデータ転送は行なわない。
- スクラッチ領域の先頭に実データの先頭を指すチェインのためのローカルポインタを上書きする。
- ネットワークを介して自分自身と同じ IU に接続し、ネットワークを経由したスクラッチ領域への転送をする。
- NIP が独自にローカル・メモリだけをアクセスして、データをスクラッチ領域へ転送する。

などの操作が考えられるが、 初めの 2 つの方法で実装を行なった (図 1)。第 1 番目の手続きによりデータ本体のアドレスをリプライとし、このリプライ・ワードはベクタ転送を要求したプロセッサによってデータ本体のアクセスに利用される。また、第 2 番目の手続きによりスクラッチ領域に妥当な最小限のデータを書き込んでいる。

ベクタ型ポインタのチェイン中に他のデータタイプのタグを持ったポインタが存在した場合には、転送エラーを生じる。

4.2 ポインタの書き換えに対する考慮

これまでの部分では、移動したベクタを読み出す場合について述べたが、実際にデータを移動する際には、ベクタの先頭部分にリンクポインタを書き込む破壊的代入が行なわれるため、何らかの手段で競合を解消する必要がある。 PIE64 のハードウエアでは、束縛済みのデータに関して、排他制御を直接おこなう事ができないので、移動しうる構造データに関して競合が起きないように、

1. 破壊的代入の起きる部分をインビジブルポインタを用いて構造体から除外する。
2. メッセージなどを用いたオーナシップ管理

といった書き込み条件を制限するようなソフトウエアによる管理を必要とする。

4.3 メンテナンス機構

基板実装時のデバッグ、メンテナンス作業を考慮し、全レジスタをコマンドバスからアクセスできる機構や、性能測定用の 24 ビットカウンタ 2 個の他に、

- 外部信号ピンへの各部状態遷移回路の状態信号をマルチプレクスして出力する
- 状態遷移回路において、各シーケンスを構成する、より基本的な動作のそれぞれを外部から起動可能とする

機能を設けた。これらにより、設計時のデバッグ作業も効率化することができた。

5 NIP のハードウエア

NIP の ハードウエア構成について述べる。まず、諸元を表 1 にしめす。

5.1 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。

表 1: NIP の諸元

使用ゲートアレイ	富士通 AU シリーズ
パッケージ	PGA, 256 ピン
信号線数	219 本
実ゲート数	約 19,000 ゲート
内部 FF ビット数	780 ビット

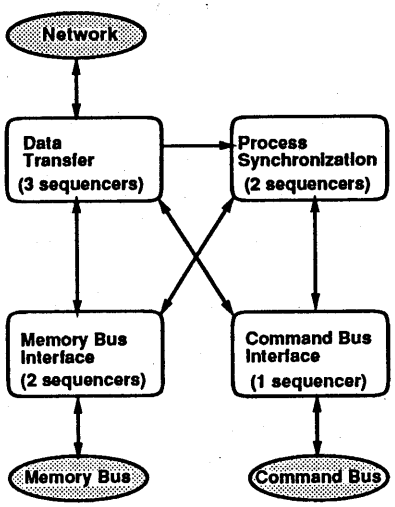


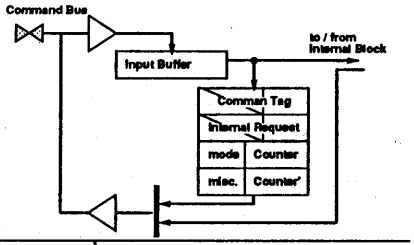
図 2: NIP の内部ブロック構成

- 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

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 2)。以下に、それぞれのブロックについて述べる。

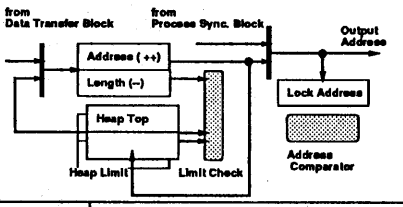
5.2 コマンドバス・インタフェス部

コマンドバス・インタフェス部は IU 内の他のプロセッサとの間でコマンド / リプライの送受信を行な



ゲート数	約 2800
状態数	14 状態 (バス・プロトコル用)

図 3: コマンドバス・インタフェス・ブロック



ゲート数	約 3600 ゲート
状態数	8 状態 (バス・プロトコル用)
状態数	8 状態 (ヒープ・レジスタ管理)

図 4: メモリバス・インタフェス・ブロック

う部分であり、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。

5.3 メモリバス・インタフェス・ブロック

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、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、および、負荷データ / メッセージの受信に使用される 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。

5.4 データ転送処理ブロック

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、外部に置かれた 8 ビットの負荷分散情報レジスタを 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。

データ転送処理ブロックのデータ・パスを図 5 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 60 ns の遅延が存在すること

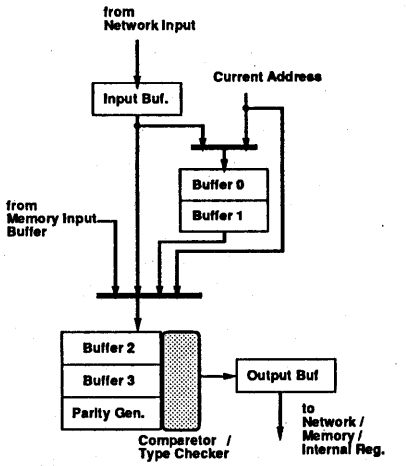
等の要求及び制約を満たすために、 データ読みだし側の NIP で 3 段、 ネットワークを 間に はさんで データ書き込み側の NIP で 3 段のパイプラインを構成する形になっている。

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 クロックの遅延ある信号線を用いて伝えあって分散制御している。

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。

5.5 プロセス間同期処理ブロック

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。



ゲート数	約 6000 ゲート
状態数	54 状態 (プロトコル制御用)
状態数	32 状態 (基本シーケンス用)
状態数	4 状態 (バッファリング用)

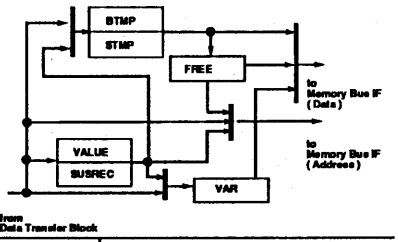
図 5: データ転送処理ブロック

プロセス間 同期処理ブロックの データ ・ パスを図 6 に示す。

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。

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、 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。



ゲート数	5000 ゲート
状態数	55 状態 (基本シーケンス用)
状態数	8 状態 (レジスタ管理)

図 6: プロセス間同期処理ブロック

6 性能

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。

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。 readn コマンドには必要に応じて、 deref コマンドと同じデレファレンス処理時間が加わる。

7 まとめ

ベクタ転送処理の一部に変更を加え、構造データのマイグレーションに対応した転送を行なえるようにした。 この変更は、従来 NIP が備えていた変数型ポインタのデレファレンスのシーケンスに、若干の修正を加えることによって実現できたため、ハードウエアのコストにはほとんど変化がなかった。

また、ベクタ転送命令においては読み出したベクタの第 1 ワードをベクタ長と見なし、転送プロトコルの初期に内部カウンタへのロードやチェックといった固有の操作を行なっていたが、 これと並行して、デレファレンスのためのタグチェックを行なっている。 従って、デレファレンスの必要のない転送をする場合でも、デレファレンスの機構のために処理時

表 2: NIP の基本コマンド実行時の処理時間 (単位: クロック数)

コマンド	Master NIP 処理時間	Network 占有時間	Slave NIP 処理時間
read1	14	11	7
read2	16	12	8
readn	17+n	13+n	9+n
readx	18+n	14+n	10+n
write1	16	14	9
write2	20	18	14
writen	22+n	20+n	16+n
writex	19+n	17+n	13+n
writel1	19	16	11
writel2	27	24	20
writeln	29+n	26+n	22+n
writelx	27+n	24+n	20+n
writem1	16	14	9
writem2	25	23	19
writemn	27+n	25+n	21+n
writemx	25+n	23+n	19+n
deref	11×r+4×l+4	11×r	7 × r
suspend	15	13	10
bind	19	16	6n+12
activate	15	13	10
mark	16	12	5
restore	15	11	4

間が増大することが避けられた。

今後は、 応用プログラムを用いたシミュレーションによる評価、および、 実際に PIE64 を用いた性能測定等を順次行なっていく予定である。

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] 高橋, 小池, 田中, “並列推論マシン PIE64 の相互結合網の作成および評価”, 並列処理シンポジウム JSPP'90, May 1990.

[2] 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.

[3] 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.

[4] 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.

[5] Koike, H. and Tanaka, H.: *Generation Scavenging GC on Distributed-Memory Parallel Computers*, Proc. of High Performance and Parallel Lisp Workshop, London, Nov., 1990.

[6] 小池 汎平, 田中 英彦: “並列推論エンジン PIE64”, 並列コンピュータアーキテクチャ, bit 臨時増刊, Vol.21, No.4, 1989, pp. 488-497.

[7] 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.

[8] 清水, 小池, 田中, “ 並列推論マシン PIE64 の推論ユニット間通信 ”, 情報処理学会計算機アーキテクチャ研究会 79-5, Nov. 1989.

[9] 島田, 下山, 清水, 小池, 田中, “ 推論プロセッサ UNIRED II 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.

[10] 日高, 小池, 田中, “ 並列推論エンジン PIE64 の推論ユニットのアーキテクチャ”, 電子情報通信学会コンピュータシステム (CPSY) 研究会, SWoPP 琉球 '90, July 1990.