ICOT Technical Report: TR-307

TR-307

並列推論マシン：
その研究の方向づけ

内田俊一

October, 1987

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1 Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

並列推論マシン：その研究の方向づけ

Parallel Inference Machine: Its Research Direction

内 田 俊 一

Shunichi Uchida

新世代コンピュータ技術開発機構

ＩＣＯＴ

1. はじめに

コンピュータの新しい応用として知識情報処理の技術を応用したいろいろな知的なシステムが注目を集め，それとともに，このような知的システムの中核的なメカニズムとして「推論」と呼ばれる処理が多用されるようになってきた。この背景には，第五世代コンピュータプロジェクトにおいて，その研究開発目標とするコンピュータシステムを，「知識ベースを用いた論理的推論を行うもの」と定義づけ，これが世界的に広まったという経緯がある。 [1] [2]

第5世代コンピュータプロジェクトでは，論理型言語をそこで研究開発されるソフトウェアやハードウェアの核となる言語（核言語）として，論理型言語を並列に実行するコンピュータシステムを並列推論マシンとよんだ。その後，論理型言語，特に，Prologを高速に実行するマシンの研究が盛んになり，この言語が並列実行可能なモデルをもつことから，並列論理型プログラミング言語マシン，すなわち，並列推論マシンの研究も逐次型推論マシンとともに，世界の各地で盛んにおこなわれるようになった。

並列推論マシンをここでは，並列論理型プログラミング言語マシンと考えて，その実現にあたって研究開発すべき重要な問題は何かという点に重点をおいて，その研究の現状を解説する。従って，ここでいう推論とは，論理型言語でいうところの，三段論法のような機械的な推論操作，もしくはこれをもう少し細かく分解した処理をさすこととする。

2．研究開発の流れ

並列推論マシンの研究開発の活発化した背景は，Prologに代表される論理型言語が，関数型言語などと比べ，より直接的に知識情処理の応用システムの推論の機構と結びつき，応用システムとマシン間の隔たりを小さくした（ように見せた）ことや，言語モデル自身が並列処理を考えやすいものであった ことがあると思われる。これは，多くの研究の始まりが，ゲームにおける探索を念頭において，ＯＲ並列型の処理の並列化の検討に置かれることから推測できる。

また，知識情報処理の応用の多くが，極めて多量の計算処理を必要とし，できる限り強力なマシンを求めていること，知識表現やその基となる理論モデルにおいて，逐次型モデルより並列型モデルのほうが素直に記述対象を表現できることなども，研究を開始する大きな動機となっている。

しかし，並列推論マシンの研究以前にも，関数型言語のための並列マシンの研究は，有名なデータフローマシンの研究を始めとして多くの試みがなされており [3]，対象とする言語を関数型から論理型に変えたからといって，並列処理に本質的な問題は解決されたわけではなく，ほとんどどがそのまま研究対象として残っている。ノイマン型言語との比較で見ると，論理型と関数型の言語は極めて近い関係にあり，並列推論マシンの研究は，関数型言語を対象とする並列マシン研究の上に積み上げていくこととなる。

並列推論マシンの研究が従来の並列マシン研究と異なることといえば，より強力なマシンを求める知識情報処理の応用システムが数多く存在するとともに，それらの応用システムがルールやフレーム等のいろいろな形の知識ベースをもち，これを用いた三段論法的な（推論的）操作をソフトウェア的に実現した形で構成されていることであろう。そのため，このような推論的操作をマシンがより直接的に，能率よく実行できることが強く望まれる状況となっている。

このようなマシン研究を取り巻く環境は，マシンの研究開発を進める上で極めて大きい影響を持つ。 従来の並列マシン研究の主体は，どちらかといえばアーキテクチャやハードウェアの研究者にあり，応用システムやソフトウェアの研究者ではなかったといえる。これは，並列マシンの研究にとっては望ましいことではない。 この結果として，並列マシンは，ソフトウェアが比較的簡単ですむ応用，すなわち，計算の構造が動的に変りにくい数値計算やパターン処理の一部においてのみ実用化されることとなった。

数値計算やパターン処理においても，問題が高度化すれば当然計算の構造も複雑となり動的に変るとともに，プログラムもなかば作り付けの状態から，そのユーザが自由にプログラムを作成し得る必要が生じる。　知識情報処理の応用では，対象とする応用範囲が極めて多様であり，効率的なプログランミング環境が必須であるとともに，計算処理の構造もいろいろと変化に富んでいる

応用面からの並列推論マシン実現の要請は，従来からあった研究分野に加え，ソフトウェアやアルゴリズム，基礎理論の研究も含めた新しい研究の展開を可能にし，ソフトウェア主導の並列マシンの研究開発が始まることとなった。従って現在の並列推論マシンの研究においては，並列言語処理系やプログラミング環境，並列ＯＳ，並列アルゴリズムなどの研究とアーキテクチャやハードウェアの研究が密接に組み合わさった形で展開しつつある。

このような展開は，日本の第５世代コンピュータプロジェクトのほか，英国のアルベイ計画におけるFlagshipプロジェクトなどでおこなわれている。[4]　また，最近では，汎用のマイクロプロセッサを多数接続したマシンが，英国のInmos社や米国のシークェント社等から市販されており，並列処理のソフトウェアやプログラミング環境に関する実験もやりやすくなり，世界の各地でおこなわれるようになってきた。

知識情報処理をめざす並列推論マシンの研究においては，　知識表現やシステム記述のための言語，プログラミングやデバッグのための手法，アルゴリズムなど幅広い研究開発が必要であり，これと結び付いた形で，マシンのアーキテクチャやＶＬＳＩを用いたハードウェアの研究開発が行われねばならない。 今後は，このような分野の研究が幅広く展開していくものと思われる。

３．並列推論マシンの構成と研究のポイント

ここでは，並列推論マシンの研究開発にあたって重要な，核となる言語の性質と実行モデルについて説明し，それをもとに構成される並列推論マシンのソフトウェアとハードウェアの持つべき機能と，それらの研究開発において重要な研究上のポイントについて，例を挙げて説明する。

並列推論マシンの構成としては，その最上位は知識情報処理の応用ソフトウェアであり，最下位は，ＶＬＳＩを含むハードウェアであるが，ここでは，システム記述言語のレベルから，マシンのアーキテクチャまでの範囲に注目することとする。並列推論マシンは，この構成の上位では，知識プログラミングを支援する（超）高級言語マシンとして見え，下位にいくほど並列マシンとしての性格が顕著になる。

３．１　並列推論マシンの中核となる言語と計算モデル

３．１．１　中核言語のもつべき性質

並列推論マシンを設計するにあたってまず考えるのは，中核となる言語である。その言語のもつべき性質には次のようなものが求められる。

１）形式的に整った並列計算モデルをもつ。（従って，プログラムの意味がその表現から，一意的に定まる。）

２）計算の過程で副作用を積極的に用いておらず，メモリが関数型や論理型の言語における変数のような形に抽象化されている。（ノイマン型マシンのように書き換え可能なメモリセルとして見えない）

３）計算モデルを実現する能率のよいアルゴリズムと実装手法が存在する。（これはハードウェア的な実現を考えるとできるかぎり簡素なものがよい。）

本格的な応用ソフトウェアを対象として，並列処理を行おうとするときには計算の制御構造が複雑となり，言語の性質として明確かつ簡潔な規則を持たない限り，その計算結果を保証することができず，また，そのマシンを作ることも，そのソフトウェアをつくることも困難となる。

論理型や関数型の言語は，このような性質を満たしている。これらの言語における変数は，およそ数値であれ文字列であれ，なんでもいれられる容器として考えればよく，ノイマン型言語にあるようなメモリセルの名札ではない。また，通常はその容器は使い捨てであり，容器はいくらでもあると考える。

これによりプログラマはメモリの管理から解放されプログラムは簡単化する。しかし棄てられた容器の回収（ガーベージコレクション）はマシン側の役目となり，その負担は増す。また，容器は使い捨てであるから，いったん一つの変数に与えられた値は変ることがなく（書き換えられないから），並列に動作するいくつものプロセスがいろいろな順序関係で読みだしても結果は変わらない。このような性質を副作用が無いといい，並列処理における重要な性質となる。

ここでは，並列推論マシンに限って説明するので，関数型言語についてはふれないが，論理型と関数型の言語は，兄と弟のようによく似ており，関数型言語を対象として研究された並列マシンの研究成果の上に並列推論マシンの研究は築かれている。

3．1．2　並列論理型言語と計算モデル

並列推論マシンの基本となる言語は，Pure Prolog（純Prolog）とよばれ形式的（数学的に）に整った性質をもっている。Pure Prologは並列実行も可能で，いくつかのOR関係にある節（クローズ）があるとき，これらを並列実行するＯＲ並列処理ができる。そして，このような実行方式にもとずく並列推論マシンの研究が，各地でおこなわれている。[5]

しかし，Pure Prologは言語の機能としては抽象度が高く，プロセス間の同期や通信の記述が難しくＯＳの記述等に用いるには問題がある。そこで，できるかぎり形式的にととのった性質を保ちつつ，同期の機能などをいれた並列論理型言語が研究された。

このようなものとして，Concurrent Prolog（CP）[6]，PARLOG [7]，Guarded HornClauses（GHC）[8] と呼ばれる言語が設計された。これらの言語の性質はこれらが同一のルーツから発展したものであるため，よく似ている。ここでは，これらのうちもっとも簡潔でハードウェア的実現に適したＧＨＣについて説明する。

ＧＨＣの構造は，図－１に示すように，Prologの節のゴール（節の呼出し）の間にコミットバーと呼ばれる縦棒を加えたものである。コミットバーの前の部分をガードもしくはパッシーブパートとよび，後ろをアクティブパートと呼ぶ。

いくつかOR関係にある節は，呼び出され，ユニフィケーション（単一化）に成功すると並列に実行が開始される。しかし，どれか一つの節のガード（パッシィブパート）の実行が成功裏に終了しコミットバーを越えると，残りの節の実行は失敗したものとして捨てられる。すなわち，OR関係にある節の一つのみが生き残り，アクティブパートの実行に進むことができる。

単一化もガードとアクティブパートでは，異なっている。ガードにおける単一化では，ある変数に対し何か値を結合（バインド）しようとしたとき，その変数になにも値が入っていない（アンバウンド，未定義である）と，その実行は一旦停止（サスペンド）する。サスペンドされたゴール（節全体）は，一つのプロセスとして扱われ，サスペンド・プロセスをためておく待行列（キュー）にいれられる。

アクティブパートにおいては，未定義変数に対する単一化により値がバインドされる。もし，この変数に値がバインドされるのを待っているサスペンド・プロセスがあると，そのプロセスは実行再開（レジューム）が可能となり，ほかの実行可能なプロセス（レディ・プロセス）と一緒にレディ・キューにいれられ，実行される。このようすを，図－２に示す。

ＧＨＣでは，ANDの関係にあるゴールも並列に実行してよく，このゴール間においても，上でのべたように，変数のバインドのタイミングでもって同期がとられながら並列処理が行われていくことになる。通常，これらのゴール間では，変数を共用して処理結果をパイプライン的に引渡しつつ処理すすめるため，ストリーム処理に基づく並列処理を行う。

このように，論理型言語の実行は，多数のプロセスが相互に同期をとりながら並列実行されるというモデルに帰着する。この種の言語は，その実行を並列にやってもよいし，逐次にやってもよく，結果は同じになる。このため，まず，逐次型のマシンの上でいろいろな研究や実験を行ないつつ，並列マシンの上に移していく方針をとる。

３．２　並列推論マシンのソフトウェア

並列推論マシンのソフトウェアの役割は，応用ソフトウェアを作成するためのシステム記述言語とそのプログラミング環境を提供すること，および，作成されたソフトウェアをハードウェア上で，効率よく実行するための実行環境を提供することである。これらは，まさにプログラミングシステム（ＰＳ）とオペレーティングシステム（ＯＳ）を備えることにほかならない。

３．２．１　システム記述言語とプログラミング環境

１）要求される機能

システム記述言語は，各種の応用ソフトウェアの記述に共通的に持ちいられるとともに，プログラミングシステムやＯＳの記述に用いるものである。研究としてみると，その設計上の目標は，まず 並列推論マシンのＯＳを十分に記述できることである。この言語は，中核となる並列論理型言語をもとに作られたものであるとともに，モジュール化機能などＯＳを構成するソフトウェアモジュール群を能率よく作成するために必要な機能をもたねばならない。

このレベルにおいては，システム・プログラマは，一つのジョブを並列処理可能ないくもの部分ジョブ（タスク）に分割する際の分割方法や，それぞれの部分ジョブにどの程度の資源（ＰＥやメモリ）を割れ当てるかを指定したい場合が生じる。このような記述能力もシステム記述言語の言語機能として検討する必要がある。

そのプログラミング環境も編集やデバッグなどを能率よくすすめるための各種のツールが必要となる。並列ソフトウェアのデバッグは，逐次型のものに比べむずかしいことが予想され，重要な研究テーマである。これまでも，アルゴリズミック・デバッギング[9]とよばれる手法が提案されたり，プログラミング環境については，PPARLOGをベースとするＯＳ[10]などの研究が開始されている。

プログラミング環境のなかには，システム記述言語から中間言語，または機械語に変換するコンパイラやライブラリを含む言語処理系も含まれる。コンパイラは，書きやすさ重視のシステム記述言語から，実行性能重視の機械語に変換する役割を負っており，そこでの最適化の手法の良否がシステム全体としての性能に大きく影響する。逐次型のPrologでは，D.H.D. Warrenにより提案されたWAMとよばれる抽象命令セット[11]とコンパイラの最適化手法が，性能向上に大きく貢献しているが，これと同様の手法を並列論理型言語にも適用しようという試みがなされている。

25

2） 並列推論マシンの言語系とその例

中核となる言語の仕様や実行モデルをもとに，システム記述言語や機械語を設計していくと，一つの言語の体系ができあがる。一例として，ICOTの並列推論マシンの研究開発における言語の体系を示すと，図－3のようになる。 英国のFlagshipプロジェクトにおいては，また別の体系があり，そこでは関数型言語を中核においている。

図－3に従って説明すると，まず，前に説明したＧＨＣがKL1-Cとして，この体系図の中核におかれている。 ジョブの分割方法や資源割当ての指定などを行う機能は，KL1-Cとは独立した性質のものである，ここでは，KL1-Pと呼ぶ別の言語（記述方法）を定義しようとしている。

KL1-Uはシステム記述言語であり，これは，KL1-Cの上につくられ，KL1-Pの機能を含みさらに，プログラムをモジュール化する機能やマクロ機能などを付加しようとしている。KL1-Uは，ＯＳの記述に重点をおいており，オブジェクト指向の概念に基づくモデルにもとずいて設計しつつあり，モジュール化についても，クラスと多重継承に基づくものを採用している。[12]

オブジェクト指向に基づくモジュール化機構は，逐次型推論マシン PSIのOS記述に用いられた拡張Prolog，ＥＳＰでも用いられ，成功をおさめている。並列論理型言語の上にオブジェクト指向の概念を持ち込む試みは，CPについても行われており，その言語は，Vulcan[13]と呼ばれている。

KL1-Uの言語の雰囲気を感じてもらうために，簡単なプログラム例を，図－4に示す。その構文もマクロ機能などをいれたことで，もともとのGHCよりは，かなり見易いものとなっている。

KL1-Bは，機械語であり，KL1-Cの実行モデルとＰＥのアーキテクチャや結合ネットワークの仕様を考慮して設計した機械語命令セットよりなる。[14] このレベルでは，単一化時の変数のバインディング等によりおこるプロセス間の通信に対しても，ＰＥの中の通信とＰＥの外との通信は別種の命令が準備されたり，単一化の操作も，より細かなレジスタやスタック，ヒープの操作命令に分解されたものとなっている。

システム記述言語と機械語では，プログラマから見て，機能的にも文法的にもかなり違ったものものであり，この間は，コンパイラにより結びつける。このとき，コンパイラはいろいろな最適化を行なう。 逐次型のPrologでおこなったクローズインデクシングや再帰ループの最適化と同様なことが可能であり，これにより，実行に要する機械語のステップ数を減らし高速化が可能となる。

ここでも，KL1-Bの雰囲気を感じてもらうために，GHCプログラムからKL1-Bへのコンパイル例を図－5に示しておく。

3．2．2 オペレーティングシステム

並列推論マシンのＯＳの重要な役割は，ユーザがプログラムを作成するにあたり，マシンを構成する要素プロセッサ(PE)の数やメモリ容量，PE間の接続の構造，入出力機器の構成などのハードウェアの細部の物理的構成条件にわずらわされないような，論理的に簡潔な実行環境を作りだすとともに，並列に実行されるプログラムの実行管理やプログラムの実行に必要なＰＥやメモリなどのハードウェア資源管理を行うことである。

この実行管理や資源管理は，従来より並列処理の重要テーマとして研究が行われてきた。通常の応用システムでは，並列に処理可能なタスク，またはプロセスの数は，ハードウェアとして存在するＰＥ数に比べはるかに多い。また，単に多いのみならずその数は時間的に増減を繰り返す。 また，そのプロセス間では，相互に通信しあったりデータを共有しあったりする。 その通信の頻度もジョブの構造などによりプロセスごとに片寄りがあったりして一様ではない。

このような並列ジョブの静的，および動的な計算処理の構造をどうしたら効率よくハードウェアの持つ構造にマッピングし得るかという問題は，マシンの性能や構成に大きく影響する。このマッピングにおいて，システムプログラマによるジョブの分割方法や資源割当て方法の指定をどう生かすかは，興味深い研究テーマである。

ＯＳの重要な研究目標は，以上のような機能をどう実現するかという点にあり，ＰＥや結合ネットワークのアーキテクチャやハードウェアもＯＳの機能が実現しやすいような性質のものでなければならない。

また，ＯＳが作り出し，プログラマに提示する論理的実行環境として，ＰＥの機能や結合ネットワークの構造（トポロジー）をどのように抽象化して見せるかも従来あまり研究されなかったテーマである。その提示の仕方により，プログラマが応用ソフトウェアを構築するさいの構造化手法やアルゴリズム，また，ジョブの分割方法や資源の割当て方法が変ってくる。

ICOTにおける研究では，論理的実行環境としてProcessing Power Plane (PPP)モデルを検討している。[15]これは，図-6に示すように，２次元平面に多数のプロセッサが均質に並んでおり，分割されたジョブ（を実行するプロセス）は，この平面上の一区画に割り当てられる。プロセス間の通信のコストは，この平面上の距離に比例する。また，あるジョブにどの程度の計算資源を割り当てるかは，この平面上でどのくらい広い区画を割り振るかで指定する。

この平面上での指定は，ＯＳによって物理的なＰＥ（実際のハードウェア）にマッピングされる。各ＰＥの稼働状況，すなわち，どのくらい忙しいかは，この平面上の対応する区画の歪みで示される。すなわち，忙しいＰＥに対応する区画は縮むと考えると，マッピング時に，縮んだ区画へ割り当てられるジョブは少なくなり，相対的に広くなった区画へのジョブ割当てのチャンスは多くなるから，ジョブ割当ての均等化が計れると考えられる。

このモデルの有効性は今後の研究を待たねばならないが，このような研究を行ないつつ，ＯＳの研究開発をすすめていく必要がある。[16]

３．３　並列推論マシンのハードウェア

３．３．１　機械語とマシンのアーキテクチャ

１）　機能的な要求と検討のすすめ方

機械語は，並列論理型言語を基本として，ＯＳ等が必要とする機能や，ＰＥのハードウェア・アーキテクチャや実装方式，ＰＥ間の結合ネットワークの構成などを考慮して設計される。また，コンパイラによる最適化の導入がしやすいことも考慮すべき条件となる。

機械語は，ハードウェアとソフトウェアの境界であり，この辺でもっとも支配的な条件となるのは，ハードウェアの回路や実装条件である。必要な機能を盛り込んだ上で目標とする性能や容量が実現できるか，所定の物理的サイズにおさまるかといったことが検討される。

求められる機能のどこまでを，ハードウェアでやり，どこまでをＯＳや言語処理系でやるかの境界の設定は最も重要な技術的判断である。原則としては，利用する頻度が高く簡素な機能はハードウェアでサポートし，たまにしか利用しない複雑な機能はソフトウェアでサポートするのであるが，その中間的なものをどうするかが問題となる。

また，ＰＥ間の通信に関する機能は複雑でもハードウェアでサポートする必要があり，ガーベージコレクションなども同様である。

この辺の設計上の判断を正確に行うためには，各機能が利用される頻度を前以て測定できればよいわけだが，通常これはほとんど不可能である。しかしそうもいっていられないから，いろいろなサンプルプログラムを作り，シミュレータ上で走らせて評価データを収集し分析して，実際に応用プログラムが走る時の状況を推定しつつ判断を行うこととなる。

システム記述言語やＯＳなど並列ソフトウェアの研究をすすめておくと，OSを含めた応用ソフトウェアの動作特性の推定が可能となり，機械語の設計をより適切なものとできる。並列ソフトウェアとハードウェアの研究を一体化して進めることの重要性の一つがここにある。

ＰＥ間の結合ネットワークについては，接続するＰＥ数，応用システムに広く適用可能な構造（トポロジー），必要な通信性能，ハードウェアの実装量などを考慮して設計する。通信機能のどこまでをＰＥに含め，どこからを結合ネットワークのハードウェアにやらせるかは重要な研究課題である。

この研究をすすめるためには，さらに規模の大きな評価用プログラムを作成しシミュレーションを行う必要があるが，実際の応用ソフトウェアを反映した評価用プログラムを作ろうと思っても，応用ソフトウェアの動作特性の推定はなかなか難しく，また，シミュレーションをやろうにもその実行環境の準備も手間のかかるものとなる。結局，小規模ながらも各種の評価を行ないつつ設計を進めることとなる。

2） アーキテクチャの設計の例

並列推論マシンの設計は，上で述べた機械語仕様とその実行モデルをもとに，ＰＥや結合ネットワークのアーキテクチャやハードウェア構造，さらに回路条件を決定していく作業である。

ＰＥの設計はマシンの性能を決定するから最も重要である。知識情報処理の応用システムでは，

一般的に見て，並列処理の単位となるプロセスに含まれる計算量がかなり小さい（グラニュアリティが小さい）ことが推測される。また，計算の構造が動的に変化するため，並列に動作するプロセスの生成，消滅の度合や，プロセス間の通信路の変化の度合も大きいと考えられる。

これはハードウェアとしては，非常に厳しい条件である。（従って最も汎用性の高い並列マシンを狙うこととなる。）

ＰＥのアーキテクチャの選択としては，逐次実行ベースのタグアーキテクチャやデータフローアーキテクチャ，さらには，連想処理をベースとするアーキテクチャなどが候補にあがる。

それぞれ長所，短所があるが，コンパイラによる最適化手法の蓄積，ハードウェアの実装技術の蓄積，現在のVLSI技術のレベル，実装すべきＰＥ数などの諸条件と，ソフトウェア重視の立場を考えると，逐次実行ベースのタグアーキテクチャをもちいるのが現時点では得策と思われるので，以下，その場合の例について説明する。

もちろんハードウェアのアーキテクチャの新しさを追及する場合には，データフローや連想処理をベースとするマシンを選択するとよく，実際に，コネクションマシンのような連想処理をベースとするマシンも開発されている。コネクションマシンでは，連想処理を用いた情報検索ではすぐれた性能を示すが，その他の応用に対してはプログラミングが極めて難しいといわれている。

タグアーキテクチャをベースとする場合でも，そのハードウェアの構造は，対象とする機械語の実行について最適になるように設計しなければならない。並列推論マシンの応用においては，計算の構造が動的に変るため，実行時の条件判定が複雑で，かつ，頻繁に発生する。（単一化というのは，もともとそのような特性を持つ。）

またプロセスのグラニュアリティが小さいから，プロセス切替えの高速化のサポートが必須である。 ＧＨＣの場合は，言語の性質としてプロセス切替えの負担が小さくなっているがハードウェア的サポートなしでは十分な高速化は難しい。また，メモリの利用は使い捨てをベースとするから，ガーベージコレクションのサポートも必須であり，そのほかＰＥ間の通信のサポートも行う必要がある。このようにして，設計したＰＥのアーキテクチャの概念構成を図－７に示す。[14]

ＰＥ間の結合方式については，極めて多種の方式が提案されている。 しかし，ＰＥ間の通信の特性として，個個のデータを連続性がないバラバラのタイミングで送り，かつその送信に対する応答が早く欲しい場合，すなわち応答時間を重視する場合の方法は限られている。 その方法の一つは，並列キャッシュと共有メモリを用いる方法である。その欠点は，メモリ および メモリとPE間を接続するバスを共有化するために，接続できるＰＥ数を多くできないことである。

そこで，結合ネットワークを階層化し，共有メモリにより密に結合した１０台程度のPEをクラスタと呼ぶ一単位とし，クラスタ間はそれより疎な結合とすることとなる。こうような方式をとった例を，図－８に示す。[17]

この場合は，プロセス間の通信の頻度に局所性を仮定していることになる。即ち，たくさんのプロセスのなかには関係の深いものと，そうでないものがあり，関係に深いもの程近くにおけばよいと考えるわけである。

このような結合方式のほかにも，クロスバースイッチやNxNのルータを使った多段スイッチなどがあり，それぞれ長所短所がある。 このような結合ネットワークの構成は，応用ソフトウェアの持つ論理的な計算の構造をハードウェアの構造にマッピングする際の，やりやすさに影響する。 まだ，並列処理をベースとした応用ソフトウェアの構造としてどのようなものが一般的なのかは明らかではない。 今後の研究により，それが分かってくることで，より効率のよいネットワークを設計することが可能となる。

３．３．２　ハードウェアの実装

ＰＥや結合ネットワークの実装は，目標性能の達成に支配的に影響する。 用い得るＬＳＩのゲート数の多少により，必要な回路規模が何個のチップに収まるかは，ＣＰＵのサイクル時間にひびいてくる。 また，メモリーを含めた一つのＰＥが一枚のプリント基板にのるか否か等が，多数のＰＥの接続するときＰＥ間の通信時間やシステム全体の性能や信頼性に影響する。

また，多数のＰＥを結合するのであるから，そのうちの一部のＰＥが故障した時の診断方法や切り放し方法も検討しそのために必要なハードウェアも付加しておかねばならない。

ソフトウェアと一体化し，評価用に十分な規模をもつ並列ソフトウェアを走らせることを考えただけでも，ハードウェアとしては，かなりの信頼性，および 保守性を備えなければならず，それを考慮したハードウェア試作が必要となる。

ＬＳＩチップの試作においては，設計から製造，テストの完了までのターンランド時間をできるだで短くすることが大切であり，ＣＡＤを含めた試作環境にも配慮が必要となる。

4. おわりに

並列推論マシンの実現のために必要となる研究の範囲と，そこにおける研究上の問題点，そこへのアプローチの例についてのべた。例については，現在，ICOTですすめている並列推論マシンの研究成果を主に用いた。しかし，ここでのべた技術上の見解も，現時点におけるものであり，今後の研究の進歩によりいろいろ変り得るし，新たなものが付け加えられるべきものである。

ソフトウェアの研究を特に重視した内容になっているが，これは極めて自然なものである（と筆者は思っている）。

これまでに例として述べたような並列推論マシンを実装した場合，PEあたり 数100KLIPS を望むとすれば，必要なゲート数は，数万ゲート以上となり，メモリもかなり多量に必要となろう。

これは，現在のLSI技術では，１チップに収めることは少々困難であり，もう少し待つ必要があろう。しかし，数チップにはおさまるから，１ＰＥを１枚のプリント基板にのせることはできよう。また，システム規模として数百ＰＥからなるマシンに対しては，ＯＳのオーバヘッドを除き，殆どのPEが忙しく稼働できるような理想的状況なら，100MLIPS以上の性能は達成できそうである。

ＯＳのオーバヘッドがどうなるか，応用システムの動作特性がどうなるかにより，実質的な性能はかわるが，それらがどうなるかは，今後の楽しみな話題である。

末筆ながら，いろいろ教えて戴く，ICOT，大学，電総研および メーカ各社の研究員諸氏に感謝する。

参考文献

[1] S. Uchida: Inference Machines in FGCS Project, ICOT TR-278, and Proc. of VLSI'87, p211-220, Aug. 1987.

[2] A. Goto and S. Uchida: Toward a High Performance Parallel Inference Machine — The Intermediate Stage Plan of PIM —, ICOT TR-201, Sept. 1986.

[3] 雨宮：関数型言語とリスト処理向きデータフロー・マシン，情報処理 Vol. 26 No. 7, p765-779, Jul. 1985.

[4] ICL Technical Journal —The ICL Fifth Generation Programme—, Vol. 5, Issue 3, Oxford Univ. Press, May 1987.

[5] K. Kumon, et al.: Kabu-wake: A New Parallel Inference Method and itsEvaluation, Proc. of COMPCON Spring 86, p168-172, Mar. 1986.

[6] E. Shapiro: A Subset of Concurrent Prolog and Its Interpreter, ICOT TR-003, 1983.

[7] K. L. Clark and S. Gregory: PARLOG: Parallel Programming in Logic, Res. Report DOC 84/4, Dept. of Computing, Imperial College of Science and Technology, London.

[8] 古川・溝口 共編: 並列論理型言語GHCとその応用, 共立出版, 1987.

[9] E.Shapiro: Algorithmic Program Debugging, The MIT Press, 1983
または [8]の本の第10章を参照.

[10] I.Foster: Logic Operating Systems: Design Issues, Proc. of 4th Int'l Conf. on Logic Programming, p910-926, May 1987.

[11] D.H.D Warren: An Abstract Prolog Instruction Set, Tech. Note 309, AI Center, CS and Tech. Division, SRI, Oct. 1983.

[12] K.Yoshida and T.Chikayama: KL1-U -Parallel Object-Oriented Language upon KL1-, ICOT TR (To apper)

[13] K.Kahn: Vulcan: Logical Concurrent Objects, Xerox PARC TR, 1986.

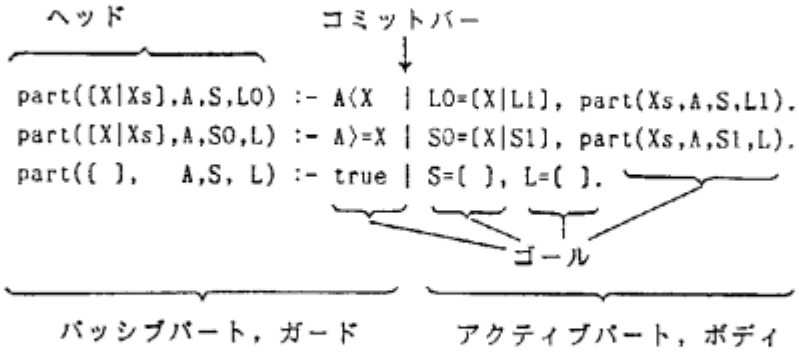
[14] Y.Kimura and T.Chikayama: An abstract KL1 machine and its instruction set, Proc. of SLP87, Sept. 1987.

[15] K.Taki: The parallel software research and development tool: Multi-PSI system, ICOT TR-237, 1987.

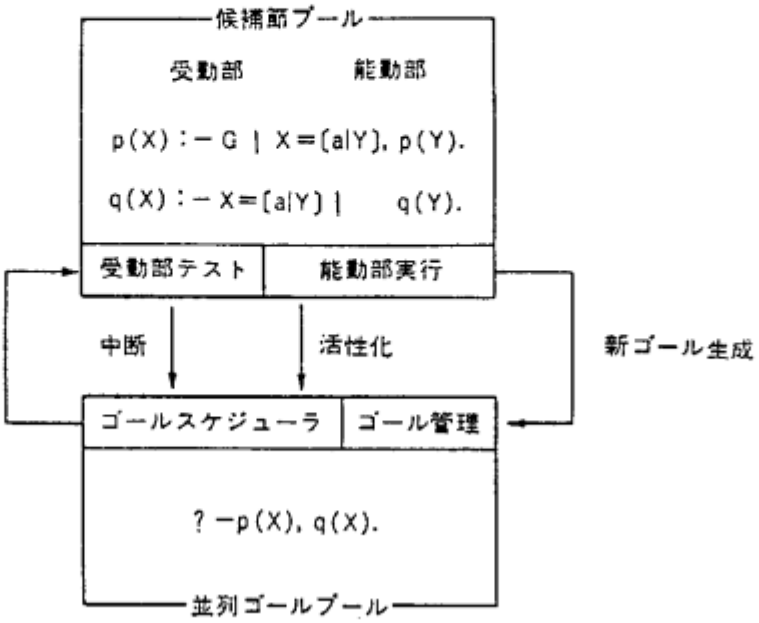
[16] ICOT第4研究室編: PIMOS第一版概念仕様書, ICOT TM-290, 1987.

[17] M.Sato: KL1 Execution Model for PIM Cluster with Shared Memory, ICOT TR-250, and Proc. of 4th Int'l Conf. on Logic Programming, p338-355, May 1987.

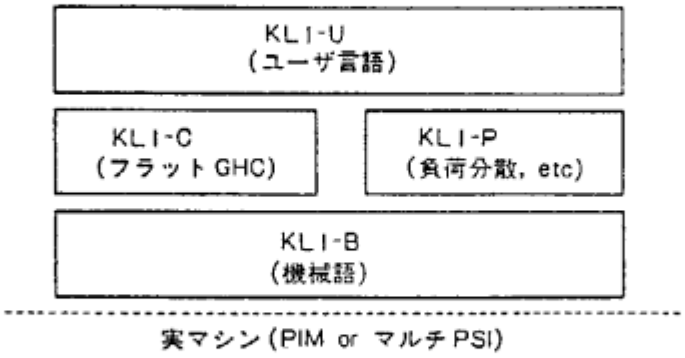
91



図－1　　ＧＨＣの節の各部の名称



図－２　ＧＨＣの実行メカニズム



図－３　並列推論マシンの言語系

```
class  stack
    slot top.
    :initiate ->
        !top = nil.
    :pop(Data) ->
        (!top \= nil) [
            :true ->
                !top :get_data(^Data) :get_next(^Next),
                !top = Next.
            :false -> .
        ] .
    :push(^Data) ->
        #element :new(^Element),
        !top = Element :set_data(Data) :set_next(!top) .
    : -> !top : ..
end.

class  element
    slot data, next.
    :set_data(^Data) -> !data = Data.
    :get_data(!data) -> .
    :set_next(^Next) -> !next = Next.
    :get_next(!next) -> .
    : -> !next : ..
end.
```

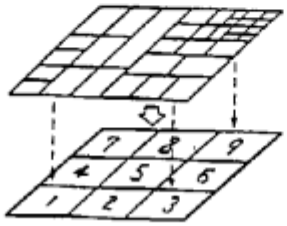
図－4　ＫＬ１－Ｕのプログラム例

```
filter( P, [ X | Xs1 ], Ys0 ) :- X mod P =\= 0 | ...
                    ↓
filter/3:  try_me_else  filter/3/1
           wait_list  A2          % Is 2nd arg list ?
           read_variable  X4      % Decomposition
           read_variable  X6      %      of list
           integer  A1            % Is 1st arg integer ?
           integer  X4            % Is var 'X' integer ?
           mod X4, A1, X5
           put_constant 0, X7
           not_equal X5, X7
                    .
```

図－５　ＧＨＣ（ＫＬ１－Ｃ）からＫＬ１－Ｂへの
コンパイル例

1つの節の各ゴールに、
P :− A, B, C.
プラグマを付加する。
P :− A↑ : {B→ : C←} ←.
これにより、ＰＰＰを分割

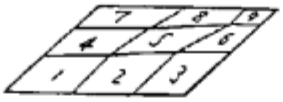
（ａ）　各ゴールへの資源の割当指定



各ゴールとの対応で
分割されたＰＰＰを

物理的プロセッサを
抽象化したＰＰＰへ
マッピングする。（９ＰＥの場合）

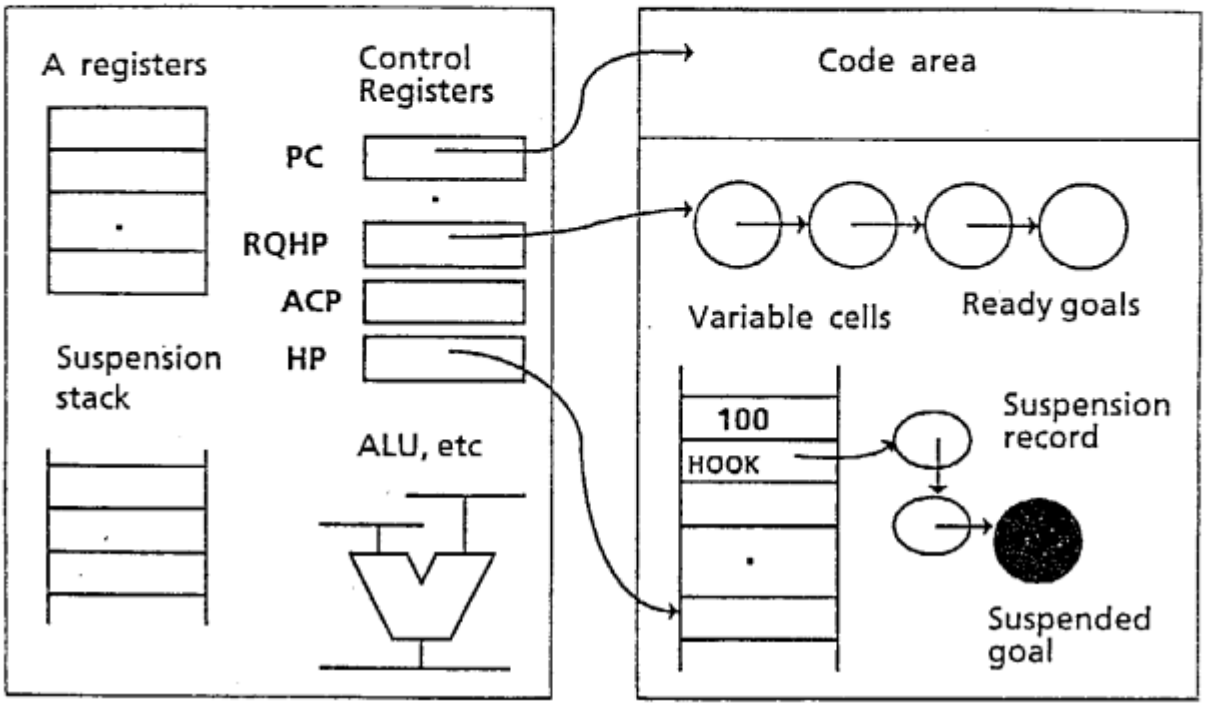
（ｂ）　資源割当と物理的プロセッサ間のマッピング



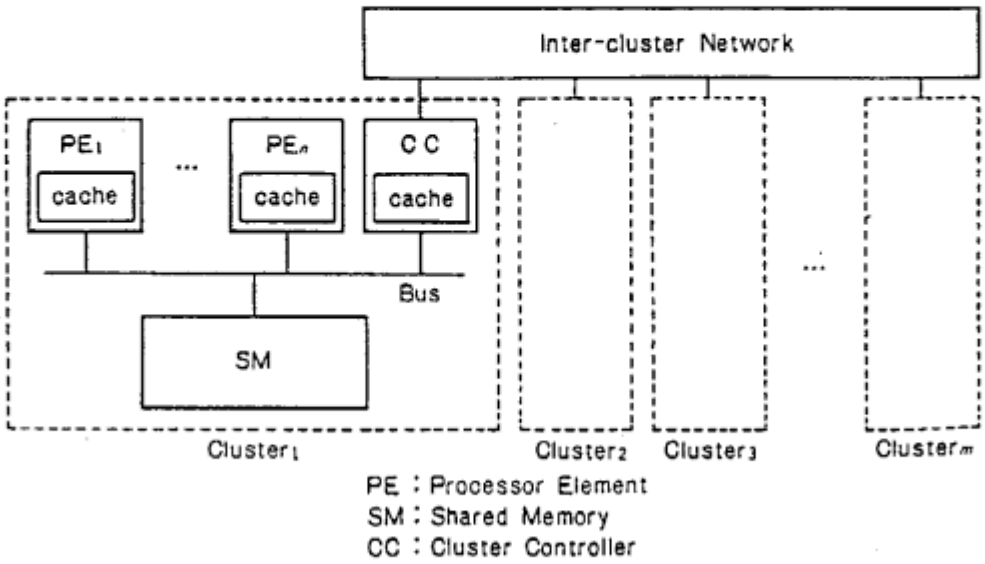
忙しいプロセッサに対応する
区画は縮む

（ｃ）　プロセッサの稼働状況の反映

図－６　　プログラマ（ＫＬ１－Ｐ）による
プロセッシングパワー平面（ＰＰＰ）の分割と
ジョブのマッピング



図－７　　ＰＥの抽象マシンの構造例



図－８　　並列マシンの構成例