知識情報処理システムのアーキテクチャ

並列リダクション・マシン

田中英彦

1. はじめに

リダクション・マシンの提案は，Klaus Berkling が1971年に行ったものが最初と考えられるが[1]，文字列を入力するとそのマシンは文字列の適当な部分を次々と書き換えてゆく．その書換えをリダクション(reduction)と呼び，リダクションを次々と行って最終の文字列を得ることを計算処理と考える．したがってマシンは，各時点において書換え可能な部分(これをリデックス：redex と呼ぶ)を見い出すこと，それに対して書換えを実際に適用することの二つを仕事とする．

この場合，一般の計算機のプログラムは書換え規則の集まりに相当するが，計算可能な関数はすべて有限の書換え規則で表現可能なことがわかっている．したがって，非常に簡単な機構ではあるが，これで汎用の計算機が少なくとも原理的には構成可能である．

一般に計算機を，操作の駆動原理によって分類すれば，通常の命令カウンタによる制御駆動のほかに，データ駆動と要求駆動がある．データ駆動は，データが利用可能になった時にそれを使う操作を始める(駆動する)が，要求駆動とは，あるデータが必要になった時に，それを作り出す操作に処理要求を出すものである．リダクション・マシンは，データ駆動，要求駆動，そのどちらによっても実現が可能である．

リダクション・マシンの提案は1971年ごろであるが，その具体的なマシンの研究は1970年代の終りごろから始められた．当初，関数型言語との親和性等から，マシンの対象言語は関数型言語とするものが多かったが，述語論理に基づいた論理型言語の発展とともに，それ用のマシンをリダクション原理に基づいて構成する研究も幾つか見られるようになって来ている．

2. リダクション・システム

2・1 一般的性質

文字列の書換えを処理のベースとした系をリダクション・システム[2]と呼ぶ．その例を**図1**に示す．□で囲まれたものが定義で，これがプログラムに相当し，(…a…)が入力された文字列である．その文字列中のaに対する要求が発生すると，定義プログラム中にあるaの定義を探し，見つかると入力文字列のaをその定義で書き換える．すなわち，aを(＊ i1 c)で置換する．ここに，(＊ i1 c)は，i1×cで与えられる値を意味しているが，これを求めようとすると，i1とcの値が必要であることがわかり，また定義中を探索し見つかったもので文字列を書き換える．という手続きを繰り返してゆくと，文字列中に演算可能な部分(この場合(＋4 1)，(＊5 2)等)が現れ，それを実行して求まった値(今の場合，それぞれ5, 10)で置き換える．そして

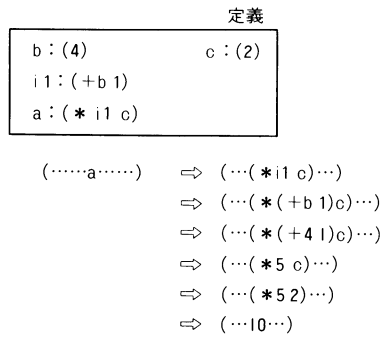

図1 リダクションの例

最終的に得られた文字列が(…10…)である．ここに，文字列の書換えは一つの定義(それぞれを節と呼ぶ)で行うか，＋，＊等の演算実行で行っているが，両者とも，より簡単な形への変換であることに変わりはない．後者の演算は定義中に置かれていないが，システムで前もって定められているプリミティブ操作である．

このようなリダクション・システムを考えるとき，問題となるのは次のような事項であろう

(1) 停止性　(2) 合流性　(3) 最適性

停止性とは，リダクション操作を行ってゆけば最終的にもはやリデックスが存在しない形(これを正規形という)になり，リダクションが終了することをいう．一般には停止性が満たされず，無限にリダクションを続ける場合がある．また，停止性を判定する一般的な方法は存在しないが，与えられた特定のシステムの停止性判定手法は幾つか提案されている．

書換えを次々と行ってゆくとき，一般にリデックスは複数存在する．したがって，そのどれから実行してゆくかによってリダクションは複数存在する．そのおのおのに対する最終結果がすべて一致する性質が合流性である．一般には合流性を満たさないが，これを満たせばどのような方法で答を求めても常に正しい答が得られることを保証できる．

最適性とは，正規形に至るまでの書換え回数が最も少ないことをいい，リダクションの効率を考えるよい指標となる．

2・2 リダクション戦略

上述のように，リダクションの各段階においてリデックスは複数存在する．このうち，どれを実行するかを決定する規則をリダクション戦略という．これによってリダクションの諸性質が定まる．リダクション戦略を定める要素としては次のようなものがある．

(1) 最内/最外　(2) 逐次処理/並列処理

(1)は関数の，より内側にあるリデックスを優先する(最内)か，外側のリデックスを優先するかということで，例えば，関数 $f(\ \ ),g(\ \ )$ があった時，$f(g(x))$ で定義される値の計算をするのに，x の値を定めてから $g(x)$ を求めるのが最内，f から先に計算を始めるのを最外という．(2)は，複数のリデックスがあった時，それらから一つだけを選んで処理するのが逐次処理，複数同時に処理するのが並列処理である．例えば，(1)の戦略を適用してもなお，同一レベルのリデックスが複数存在しうるので，その中から一つ(例えば一番左側にあるもの)選ぶことが逐次処理では必要で，それらを並列に処理することも考えられよう．したがって，(1)と(2)の組合せとして4種ありうるが，これらのほかに並列処理では，最内/最小を問わず，リデックスとして存在するものすべてを処理の対象とする戦略もある．

また，書換えを実行するとき，同じデータ構造が何回か出現した場合，毎回その構造をコピーして新たにデータ構造を作り上げるか，ポインタを用いてその構造を共有するかの選択がある．一般にコピー方式は単純であるがオーバヘッドが多くなる可能性があり，ポインタ方式は能率がよいが，その制御，特に並列処理上の制御が複雑となる．前者をストリング・リダクション，後者をグラフ・リダクションと呼んでいる．

入力ストリングが正規形を持つ場合，必ず正規形が得られる戦略は，ある制約下では最外リダクションであり，書換え回数が最も少なくなる戦略は，やはりその制約下で，コピー無し(ポインタ方式)最外リダクションであることが一般にわかっている．

並列性の観点から眺めると，並列性の源としては幾つか考えられる．まず，引数を複数持つ関数があった時，それらの引数の値を並列に求めることから並列性が生じる．また，ある関数を処理するとき，まずその答を入れるべきメモリ領域を確保して，実際に値の書込みが終わっていなくてもそのアドレスを返し，これを受けた次の関数は，あたかも値をもらったかのように処理を始め，可能な所まで処理を進めることが考えられる．このような工夫によって，等価的な並列性が非常に増すことがある．

3. 関数型言語のリダクション・マシン

3・1 概　　要

LISP や FFP 等の関数型言語を対象としたリダクション・マシンの研究には次のようなものがある．

(1) GMD リダクション・マシン(Berkling マシン)
(2) Newcastle リダクション・マシン
(3) Magó マシン
(4) AMPS(Keller マシン)
(5) SKIM マシン
(6) ALICE マシン

(1)は，K. Berkling のアイデアを実装したマシンで，1978 年ごろより稼動しているが，λ計算に基づいた言

語を直接実行するストリング・リダクション・マシンである．逐次制御ではあるが，リダクション・マシンの原型である．

(2)は，Newcastle Upon Tyne 大学より提案されたペーパーマシンで，最内リダクションに基づく並列ストリング・リダクション・マシンである．多くのプロセッサがシフトレジスタをはさんで直列に接続された構成をしている[3]．

(3)は，North Carolina 大学で研究されているトリー構造のセルラーコンピュータで[4]，Backus の FFP[5]に似た言語を実行する．並列最内リダクションに基づくストリング・リダクション・マシンである．

(4)は，Utah 大学で研究されているマシンで[6]，lenient cons を含んだ LISP 様の言語(FGL)を対象とする並列グラフリダクション・マシンであり，＋のようなオペレータの引数を並列に要求することで並列性を得ている．ペーパーマシンであるがその構造は，プロセッサ・メモリ対を葉とし，負荷バランスや通信を司どる部分を中間ノードとする 2 進木になっている．

(5)は Cambridge 大学で作成されたマシンで[7]，組合せ子(combinators)と呼ばれる基本的な関数からあらゆる関数を構成するという考えに基づき，組合せ子の簡約によりプログラムの実行を行う．これは最初に作られたグラフ・リダクション・マシンである．遅延評価を取り入れ，逐次最外リダクションを実行する．

(6)は Imperial College of London で試作中のマシンで[8]，関数型言語のみならず論理型言語や従来型言語等広範な言語を支援することを目指したグラフ・リダクション・マシンである．Inmos 社の Transputer を要素に用いて実装が行われることになっている．

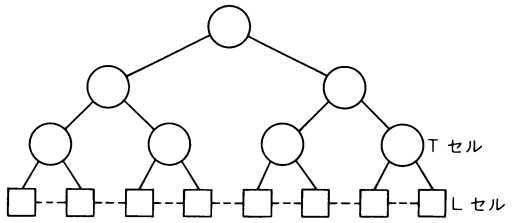

図2　Magó マシンの全体構造

以下ここではこれらのうち，Magó マシンと ALICE についてより詳細に述べる．

3・2　Magó マシン

Magó マシンは図 **2** のような木構造のマシンである[4]．葉に当たる所を L セル，他のノードを T セルと呼ぶ．このような構造は拡張性に富み VLSI に向いている．すなわち，木を寄せ集めたものはまた木であり，セル数が増しても部分的な複雑性は増さない．各セルは数十個のレジスタを含む程度の簡単なもので，それらを幾つか集めた部分木を一つの VLSI チップとすることが想定されている．

対象とする言語は Backus の FFP であり，プログラムは，application と sequence と呼ばれるものからなる．application とは，一つのオペレータとオペランドの対で，sequence は，要素の羅列である．例えば，〈7,

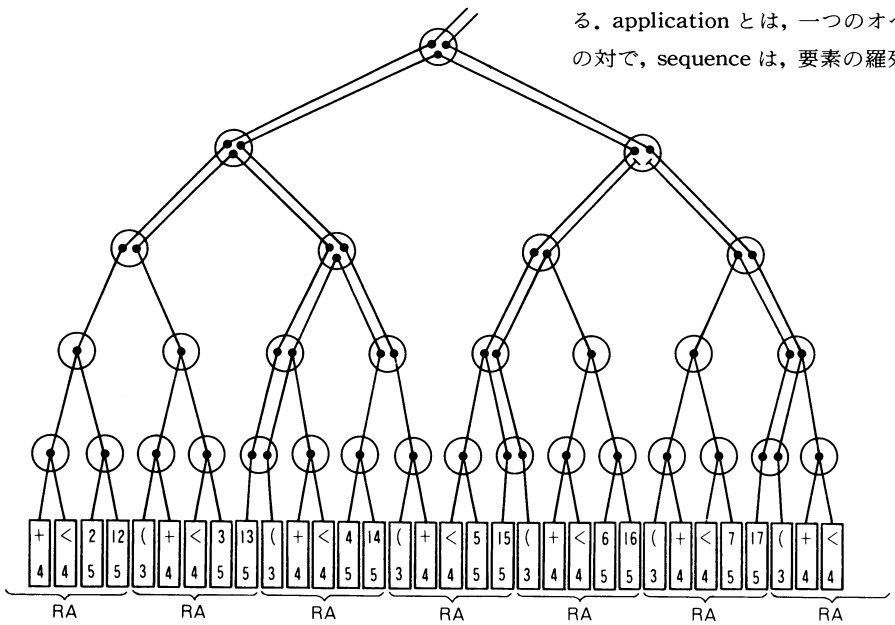

図3　トリーの分割による並列処理の例

(＋：〈2,5〉)〉という式は，2要素のsequenceであって，第1要素は7，第2要素は一つのapplication (＋：〈2,5〉) である．このオペレータは＋であり，オペランドは2と5のsequenceである．このFFP言語は，いわゆるChurch-Rosserの性質を満たすので，リダクションの順序によらず唯一の解が得られる．今，ある文字列が与えられた時，その中でリダクション可能なapplicationをRA (Reducible Application) と呼ぶことにすると，それらを並列に処理してゆくのがこのマシンである．

図3は，(〈AA, ＋〉：〈〈2,12〉, 〈3,13〉, 〈4,14〉, …〉) のテキストを処理している例で，オペレータ＋をsequenceのすべての要素に施す処理(apply to all)を表している．これを1回リダクションすると，〈(＋：〈2,12〉)，(＋：〈3,13〉)，(＋：〈4,14〉)，…〉となりその結果が図3のLセルに収められている．図のように，Lセルは一つずつシンボル(とネストの深さ)を保持し，不要なデリミタ(右側のデリミタ)を除いた形でテキストが収められる．Tセルは通信に使われるとともに，各RAに対応するトリー情報を持つ．

したがって図の場合，各RAはそれぞれ異なるLセルに存在し独立に実行可能である．通信径路長はLセル数の対数に比例するので，あまり長くはならない．各RAに対応するサブトリーは通信し合って一つのリダクションを実行するが，その結果，テキスト長が変化する．したがって，テキストの再配置操作が必要となり，それが処理速度を落とすことになるが，検討によると，このマシンの並列性はこのような損失を十分に補うことが可能といわれている．

3・3 ALICE

階乗計算のプログラムを，HOPEと呼ばれる言語で書いたのが**図4**である．これは階乗を並列に計算するアルゴリズムを用いているが，ALICE[8]では，このような処理をグラフ・リダクションで実行する．すなわち，マシン内で処理の動きを記述するグラフは，パケット

```
Factorial : Integer −>Integer
   Factorial (n)<=FactB(0,n)
FactB : Integer α Integer −>Integer
   FactB(i, j) <=      1 if  i=j
                  else j if  j=i+1
                  else FactB(i,mid) *FactB(mid,j)
                       where mid==Round ((i+j)/2)
```

図4　HOPEによるプログラム記述例(階乗計算)

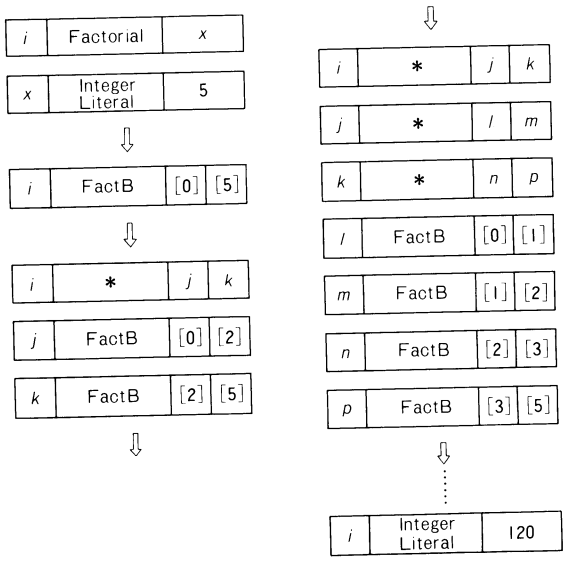

図5　階乗計算の動作例

と呼ばれるデータ構造をポインタで相互に結ぶことで表現され，reducibleなパケットを次々と変形してゆくことにより処理が進行する．例えば，図4のプログラムを用いて5！を計算した時の，グラフの移り変りを示したのが**図5**である．図中の各長方形は一つ一つがパケットを表し，先頭にはアドレスに相当するID番号が入り，次に関数名や，タグが入り，最後にオペランドのID番号や値それ自身が収められる．

図5では，まず，関数Factorialが，関数FactBに置き換えられ，次にFactBは，図4のプログラムで示すように二つのFactBで置き換えられる(FactB(0,5)が，FactB(0,2)＊FactB(2,5)で)．これを繰り返し，FactBのオペランドが(i,i+1)や(i,i)になったとき，

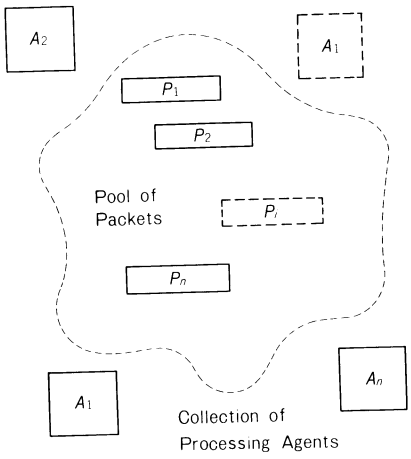

図6　ALICEの抽象アーキテクチャ

値 i+1 や i にそれぞれ置き換えられ，更に，整数値どうしの乗算は reducible なので演算が行われて最終的に値 120(=5！)が得られる．

したがって，ALICE の抽象アーキテクチャは**図6**に示すようなものである．真中に数多くのパケットを蓄えるパケットプールがあり，囲りには Agent と呼ばれるプロセッサが複数存在する．Agent は，書換え規則を保持しており，各パケットが reducible かどうかを調べ，reducible ならば書換えを行って新しいパケットを生成してプールに戻す．

現在，ALICE のプロトタイプの作成が行われているが，16 台の Agent と 16 台のパケットプールセグメントを網で結んだ形が考えられており，各 Agent には汎用マイクロコンピュータ Transputer が複数用いられる予定となっている．

また ALICE では，関数型言語のほかに，PARLOG と呼ばれる論理型言語の処理も考えられるが，この言語は論理型言語の特徴である値結合の双方向性を制限しているので，実質的には関数型言語と大差はない．

4. 論理型言語のリダクション・マシン

4・1 論理型言語の並列処理

PROLOG 等の論理型言語は 1 階述語論理に基づいたもので，例えば，X が Y の grandparent であることの条件は，X が Z の parent であることの言明

parent (X,Z)

を用いて次のように書かれる．

grandparent (X, Y) if parent (X, Z) AND
parent (Z, Y). (1)

また，変数 X, Y, Z 等の具体的な値として，taro は masao の親であることの言明は次のように表される．

parent (taro, masao). (2)

論理型言語のプログラムは基本的に(1)や(2)の形の記述(節と言う)の集まりからなり，(1)の形の節をルール，(2)の形の節をファクトと呼ぶ．今，プログラムとして(1)，(2)のほかに

parent (taro, yoshiko). (3)
parent (yoshiko, hideki). (4)
parent (yoshiko, kaori). (5)

を含んでいるとしよう．これらに対し，hideki の grandparent は誰か？ という質問 grandparent (X, hideki)？ が与えられたとすると，その解き方は次のようになる．まず，質問の述語名と一致する節をプログラム中で捜し，(1)の左辺が見つかる．そして(1)の Y に hideki を代入すれば，元の質問文は，新たな質問

parent (X, Z) AND parent (Z, *hideki*)? (6)

に変換される．次に(6)の第 1 条件を満たす節をプログラム中で捜すと，X と Z のペアの値として，(2)〜(5)より 4 組の候補値が得られる．

(X, Z)=(taro, masao) または (taro, yoshiko)
または
(yoshiko, hideki) または (yoshiko, kaori) (7)

これらおのおのを(6)の第 2 条件に代入して，実際にそれらが満たされる(プログラム中にファクトとして含まれている)かどうかチェックすれば，最終的に，

(X, Z)= (taro, yoshiko)

が得られ，元の答は taro であることがわかる．

一般に論理型言語といっても様々で，並列リダクション・マシンの対象となっているものは PROLOG，その中でも上記例題でも示したような pure PROLOG が中心である．並列論理型言語としては，Concurrent PROLOG, PARLOG, KL 1, PARALOG 等幾つかあるが，いまだ固まっているとは言いがたい．以下，pure PROLOG を中心として話を進める．

pure PROLOG の処理で取り出すことのできる代表的な並列性には次のものがある．

① OR 並列　② AND 並列
③ 引数の並列統合化
④ AND のパイプライン処理(ストリーム並列)

前記例から，①は，一つの述語に対し複数の候補条件((7)のような)を求め，それらを並列に試す並列性であり，②は(6)のような質問に答えるのに AND で結ばれた二つの述語から並列に条件を求め，それらをつき合わせて所望の解を得る並列性である．③は，質問の各述語をプログラム内の節と一致をとる時，引数についての一致性検査と値の代入(unification：統合化という)を各引数に対し並列に行う並列性をいう．一般に②や③は，並列に動く仕事が完全に独立とは限らないため，これらの実装は結果の無予盾性チェックにかなりの時間がかかる可能性があり，引数間に関係が無い場合や，引数間のデータの流れが決定的である場合以外は，その一般的な採用は難しいと言われる．④はパイプライン方式による①や②の実現法を指す．すなわち，(6)を解く場合，第 1 条件を満たす一つの組(taro, masao)が得られると，それを直ちに第 2 条件のチェック処理にま

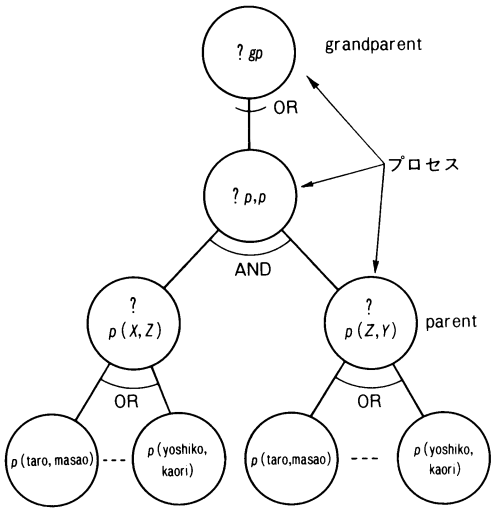

図7　AND-OR プロセスモデル

わし，同時に第1条件を満たす二つ目の組の値を求める．こうすれば，二つの条件の処理は並列に(パイプラインで)進めることができる．

4•2　論理型リダクションモデルの研究

この方面の研究はここ数年始まったばかりであり，研究の内容としては(並列)PROLOG の処理モデルの検討が多く，それに基づいた具体的なマシンアーキテクチャの検討はまだあまり多くはない．

まず処理モデルとして基本的なものに UC Irvine の AND-OR プロセスモデル[9]がある．前節の例の処理をこのモデルで図示すれば**図7**のようになる．すなわち，トリーの各ノードは AND と OR の属性を交互に(深さ方向でみれば)持ち，子ノードの結果をそれぞれ AND と OR 関係で結合して親ノードに引き渡すという形をとるが，この処理モデルではその各ノード処理を AND プロセスと OR プロセスに割り当てる．したがって子ノードの生成は対応するプロセスの生成を意味し，結果はプロセス間通信によって引き渡される．

東大のゴール書換えモデル[10]は，OR 関係に着目し，解くべき問題(ゴール)を独立な複数の新しいゴールに書き換えることによって処理を進める．したがってその並列性は，各ゴールを異なったプロセッサで処理するところから生じる．複数ノード間の関連付けは，ゴールの書換え処理とは独立に，ゴール間での木構造(推論木と呼ぶ)を作成•管理することで行っている．

ICOT のリダクションモデル[11]は，ゴールのプールから統合化プロセッサに仕事を送る時，ゴール全体ではなく，各ゴールを構成する述語単位(サブゴールまたはリテラルと言う)で送る．したがってサブゴールには元のゴールへのポインタが入っており，ゴールプールに戻されて来たそのポインタを通して親ゴール内の一つの述語と関連付けられる．また，このモデルでは Concurrent PROLOG の実行支援も検討されている．

富士通•ICOT の株分けモデル[12]は，PROLOG の逐次処理(したがってバックトラックがある)を基本とし，プロセッサが空いていればそこから仕事要求を出す．要求を受けたプロセッサは，独立な仕事の分割を考えてそれを分け与えるというもので，逐次処理のマルチプロセッサへの自然な拡張方式である．

以上のほか，各定義節をデータフローグラフに展開しておく電総研のプロセスグラフモデル[13]，OR 並列とストリーム並列を用いた京大の並列リダクションモデル[14]，OR 並列のみを対象とした神戸大学の K-PROLOG[15]やスエーデン Royal Institute of Technology の OR 並列トークンマシンモデル[16]等がある．

4•3　マシンアーキテクチャ

並列論理型リダクション•マシンの本格的なものはいまだ試作が行われていないが，それを目指した研究

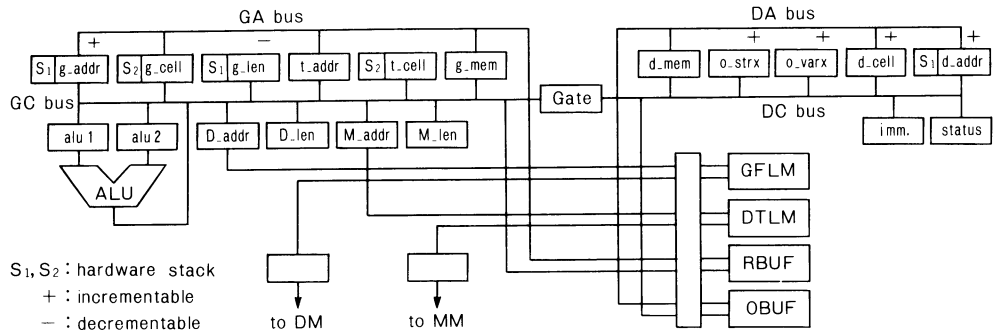

図8　ハードウェア•ユニファイアのデータパス部(PIE)

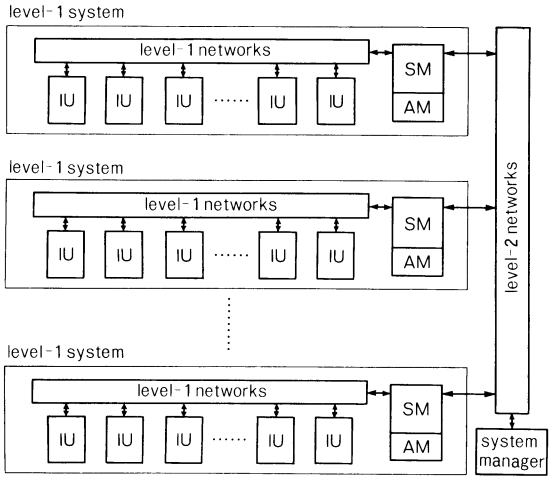

図9　PIE の全体構成

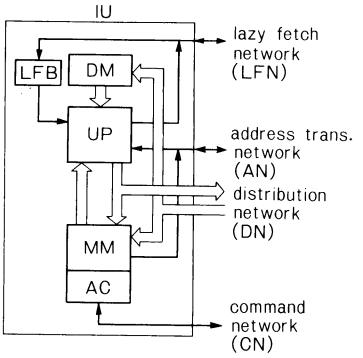

図10　推論ユニットの内部構成(PIE)

が幾つか進められている．その研究は二つに分けられる．一つは，各リダクション操作それ自体を高速に実行するためのユニファイアの研究で，他はリダクション操作をいわば単位処理として中味を問わず，むしろ多くのプロセッサを有効に用いて負荷分散を成功させる方法の研究である．

前者に属する研究としては，**図8**のようなハードウェアユニファイアの実装がある[17]ほか，LISP マシン FACOM α を用いてマイクロプログラムでユニファイアを実装する試みがある．

後者の研究は，いわば以前より行われて来たマルチプロセッサの研究と非常に近い．しかし，従来の研究では並列タスクの生成や制御をプログラマが直接行う必要があったのに対し，この研究ではpure PROLOG という枠組を持ち込むことによって，並列性が向上し制御が半自動的に行われるところに特徴がある．

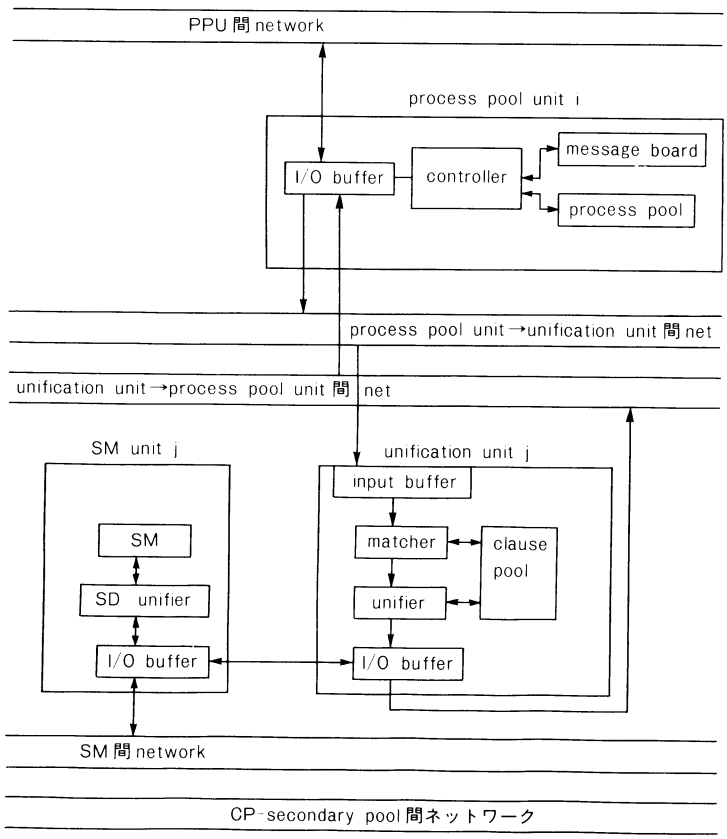

図11　PIM-R のアーキテクチャ

この負荷分散の研究は，シミュレータプログラムを作成することや，汎用マルチマイクロプロセッサによるハードウェアシミュレータ上で実験することにより行われている．ハードウェアシミュレータとしてはZ-80 を24台用いた東大の TOPSTAR-II，8086 を8台用いた神戸大の KUPS-II，M 68000 を16台用いた富士通・ICOT のマシン等が既にあるほか，M 68000 を8台用いたものが日立・ICOT で試作されている．

これらの研究を通して並列マシンの制御アルゴリズムや細部構成が今後明らかにされてゆくものと思われるが，そのターゲットとして提案されているマシンアーキテクチャを二つ図に示す．**図9**は東大の PIE

(Parallel Inference Engine)の全体構造で[18]，各推論ユニット(IU)の内部構造は図**10**のようなものである．図**11**はICOTのPIM-R[11] (Parallel Inference Machine)である．要素プロセッサ数としてそれぞれ，1 024台，100台程度を想定しており，全体としてかなり高性能なマシンとすることが考えられている．

5. おわりに

与えられた問題を書換え規則を用いて次々と書き換えることにより処理を進めるリダクション・マシンについて，特にその並列処理に重点を置いて述べた．様々なマシンが提案されており，それらを関数型言語向きと論理型言語向きとに分けて述べたが，これは対象としている言語で分類した．これらの言語自身最近は中間的なものも現れているほか，中には両言語を対象としたマシンもあり少々あいまいなところも存在するのでこれは便宜上の分類である．ここに述べたマシンのほかに，並列リダクションと関連の深いマシンとしては，Columbia大学のDADOマシン[19]，MITのConnection Machine[20]等がある．前者は，プロダクションシステム用のトリーマシン，後者は意味ネットワークを一様なプロセッサネットワークで実現することの提案で，いずれも大規模な並列マシンである．更に，推論マシンをデータフローで表現しようというICOTや通研の研究も，あるマシンレベルからみれば並列リダクション・マシンと見なすこともできよう．

これらの並列リダクション・マシンの研究はまだ歴史が浅く，基礎的な研究段階にある．実用マシンとして広く世の中で使われるためには，まだ多くの解決すべき問題が残されている．今後の発展に期待したい．

参考文献

1) K. J. Berkling: A Computing Machine Based on Tree Structures, IEEE Trans. Computer, C-20, 4, pp. 404-418, Jan. 1971
2) 二木，外出：項書き換え型計算モデルとその応用，情報処理，24，2，pp. 133-146, Feb. 1983
3) P. C. Treleaven, D. R. Brownbridge, R. P. Hopkins: Data Driven and Demand Driven Computer Architecture, Computing Surveys, 14, 1, pp. 93-143, March 1982
4) G. A. Magó: A Cellular Computer Architecture for Functional Programming, COMPCON 80, pp. 179-187, Feb. 1980
5) J. Backus: Can Programming Be Liberated from the von-Neumann Style ? A Functional Style and Its Algebra of Programs, Comm. ACM, 21, 8, pp. 613-641, 1978
6) R. M. Keller et al.: A Loosely Coupled Applicative Multiprocessing System, AFIPS, NCC, pp. 861-870, 1978
7) T. J. W. Clark et al.: SKIM-The S, K, I Reduction Machine, Proc. LISP-80 Conf., pp. 128-135, Aug. 1980
8) J. Darlington, M. Reeve: ALICE Multiprocessor Reduction Machine for the Parallel Evaluation of Applicative Languages, Proc. 1981 Conf. on Functional Programming Language and Machine Architecture
9) J. S. Conery: The AND/OR Process Model for Parallel Interpretation of Logic Programs, UC Irvine, TR-204, June 1983
10) 後藤，相田，田中，元岡：ゴール書き換えモデルに基づく論理型プログラムの並列処理方式，情報処理学会論文誌，25, 3, pp. 413-419, May 1984
11) 尾内，麻生，清水ほか：並列推論マシンPIM-R，アーキテクチャとソフトウェアシミュレーション，ICOT Technical Report TR-077, 1984
12) 久門，板敷，佐藤，増沢，相馬：並列推論処理システム—株分け方式，情報処理学会第30回全国大会，1985年3月
13) S. Umeyama, K. Tamura: A Parallel Execution Model of Logic Programs, Proc. 10th Annual Intl. Symp. on Computer Architecture, 1983
14) 八田，柴山，萩原：並列リダクションモデルに基づくPrologマシンのハードウェア構成，情報処理学会第30回全国大会，1985年3月
15) 田村，松田，金田，前川：K-Prolog(並列Prolog)の実現方法について，Proc. Logic Programming Conference '83, ICOT, March 1983
16) S. Haridi, A. Ciepielewski: An OR-Parallel Token Machine, Royal Inst. Tech., TRITA-CS-8303, 1983
17) M. Yuhara, H. Koike, H. Tanaka, T. Moto-Oka: A Unify Processor Pilot Machine for PIE, Proc. Logic Programming Conference '84, ICOT, March 1984
18) T. Moto-Oka, H. Tanaka, H. Aida, K. Hirata, T. Maruyama: The Architecture of A Parallel Inference Engine-PIE-, Proc. Fifth Generation Computer Systems '84, pp. 479-488, Nov. 1984
19) S. J. Stolfo, D. E. Shaw: DADO: A Tree Structured Machine Architecture for Reduction Systems, AAAI-82, Angust 1982
20) W. D. Hills: The Connection Machine, MIT, AI Memo, No. 646, September 1981

(たなか ひでひこ　東京大学電気工学科・工博)