計算機アーキテクチャ　57－3
（1985. 3. 28）

並列推論マシン

田中　英彦
（東京大学　工学部）

1. はじめに

第5世代コンピュータプロジェクトの発足以来，新しいコンピュータアーキテクチャとして推論マシンが注目されている．即ち，コンピュータ処理の基本を推論操作に置き，あらゆる処理をそれによって実現するという新しい枠組の研究である．これによると，コンピュータシステムは推論マシンと知識ベースマシンとからなり相互の密接なやりとりを通して処理が進行する．この枠組によってセマンティックギャップが減少しソフトウェア開発が容易になるとともに，人間により近づいた高級な処理が可能になると期待されている．マシンアーキテクチャ側から見ると，推論には多くの並列性が含まれることが期待され，プログラマに負担をかけることなく自然な形でその並列性を生かすことが望まれる．

本文では推論処理の一般的説明から入り，その並列性の種類，推論処理に向いた推論マシン設計上のポイントを述べ，現在研究が行なわれている幾つかのマシン例を紹介し，最後に今後の展望について触れる．

2. 推論処理

推論には一般に，帰納と演繹とがあるが，通常のプログラム処理に使われるのは演繹である．帰納は，知識獲得等の基本と考えられ，知識情報処理の別の局面に於ける重要な操作である．演繹の代表的なプログラミング言語に Prolog がある．これを用いて，例えば，taro は masao の親であることの言明は，

```
parent ( taro , masao ).                    (1)
```

と書き，X が Y の grandparent である為の条件は，X が Z の親であることの言明 parent(X, Z) 等を用いて

```
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).    (2)
```

と書かれる．ここに，:- の右側は，左側を成立させる為の条件を与え，2つの parent リテラルを結ぶ "," は論理の AND を表わしている．即ち，X が Y の grandparent である為には，X が Z の親であり且つ Z が Y の親であることが必要である．Prolog のプログラムは基本的に(1)や(2)の集まりからなり，(1)のような形の節を fact，(2)のような形の節を rule と呼ぶ．今，プログラムの fact として(1)の他に

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

を含んでいるとする．このプログラムに対して，"hideki の grandparent は誰か？" という質問

```
?- grandparent ( X , hideki ).              (6)
```

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述語名（fact，又は rule の左辺の述語名）を持つ節をプログラム内で捜して(2)が見つかる．質問の条件を(2)の左辺に代入すれば，元の質問文を成立させる条件は(2)の右辺より，

?- parent(X, Z), parent(Z, hideki). (7)

となる．即ち，(6) に代って新たな質問(7)が得られる．次に，(7) のオ1リテラルと一致する節をプログラム中で捜すと，(1), (3), (4), (5) より

(X, Z) = (taro, masao) または (taro, yoshiko)
または (yoshiko, hideki) または (yoshiko, kaori) (8)

の4条件が得られる．これら各々を(7)に代入すれば，

?- parent(masao, hideki). X = taro (9)
?- parent(yoshiko, hideki). X = taro (10)
?- parent(hideki, hideki). X = yoshiko (11)
?- parent(kaori, hideki). X = yoshiko (12)

が得られる．これら各々が成立するか否かをプログラム内でチェックすれば（対応するfactが存在するか否か），(10)のみが成り立つことが解り，従って，元の(6)を満たすXはtaroであることが得られる．

以上のような処理が推論処理であるが，これを並列性という観点から眺めれば次のような並列性が存在する．

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）

①は，(7)のオ1リテラルと一致する節を並列に捜す（この場合，並列度4）ことに相当し，②は，(7)の2つの条件を並列に捜す（従って，後程捜した結果をつき合わせてつじつまの合うものを選ぶ処理が必要）ことに相当する．③は，(6)と(2)との一致を取るとき，(6)のオ1引数と(2)のオ1引数の一致操作とオ2引数同の一致操作を並列に実行することに相当し，④は(7)のオ1リテラルの処理結果（例えば（X, Z）=（taro, masao））をオ2リテラルに代入してそれが成立するか否かを調べる処理と，(7)のオ1リテラルに対するオ2番目の処理結果（今の場合，（X, Z）=（taro, yoshiko））を求める処理を並列に実行することに相当する．ここに，引数についての一致性検査と必要ならば変数に値を代入すること（例えば(7)のXにtaroという値を代入）を統合化（unification）と呼んでいる．

一般に，①や④は直接的で比較的容易であるが，②や③は，引数同に関係が無い場合は容易であるが，引数同に関係があると処理が複雑となり，並列に処理することによって得られるメリットを処理の複雑化が打消す結果となり勝ちであるといわれている．

3. 推論マシンの設計ポイント

3.1 推論マシンのモデル

上記例からも明らかなように，推論は質問（goalと呼ぶ）を次々と書き換えてゆく処理に対応し，1つのゴールから複数のゴールが生成されたり，(9)～(12)のゴールの内，(10)だけ残って他が消滅するように消えていったりする．従って，その論理的なイメージ

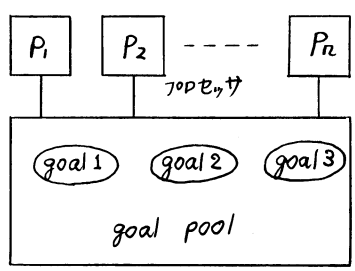

図1 推論マシンのイメージ

は図1に示すようなものとなろう．即ち，ゴールが幾つか蓄わえられたゴールプールがあってその周りに複数のプロセッサが存在し，各プロセッサは適当な基準によって処理すべきゴールを決定してそれにアクセスし，統合化操作を実行してそのゴールを書き換える．書き換えが次々と行なわれ，全ゴールに決着（真か偽か）が付いた所で処理は終了し，残された情報（変数への値代入状況，変数間の関係等）が結果となる．このような論理モデルを実際のマシンとして実現する場合，以下の節で述べるような項目が設計ポイントとなる．

3.2 処理モデル

計算モデルを駆動機構で分類すれば，

① 制御駆動
② データ駆動
③ 要求駆動

に分けられるが，上記推論の論理モデルをどのような計算モデルで動かすかによって分類すれば，やはりこの3種に分けられよう．この内，①は従来からのマルチプロセッサ研究に相当するので，ここで注目したいのは後2者である．②は，Prolog等に代表される論理型言語をデータフローグラフに落とし，それをデータフローマシン上で動かすことに相当する．従って研究の中心は，その変換方式となる．一方③は，リダクションマシン上で論理型言語を動かすことに対応するが，見方を変えれば，論理型言語を直接実行する高級言語マシンを作ることに相当する．従ってこのアプローチでは，言語の実行制御を如何にハードウェアで能率良く実現するかが問題となり，3.3節以下で述べるポイントが密接に関係する．

3.3 対象とする並列性

2章で述べた種々の並列性の内，どれを実現の対象とするかということで，一般的にはOR並列やストリーム並列がよく用いられる．AND並列は，プログラマが明示的に指定した場合や，変数間に関係が無い場合に限ることが多い．引数間の並列性についてもAND並列と似たような状況にあり，引数統合器を複数設けることによる利得は2程度と言われている．ストリーム並列は基本的にOR並列と相補的な関係にあるが，パイプライン方式自体をOR並列内の各所に組み込むことは可能である．

3.4 ゴールデータの構造

図1に示すゴールプールには多くのゴールが収められているが，そのデータ構造をどうするか，又，それへのアクセス単位をどう選ぶかという問題である．一般にゴールは，幾つかのリテラルのANDとして表現されている為，ゴール相互には同一リテラルが含まれるという状況が多く生じる．従って，各ゴールを記憶上で表現するとき，

① リテラル構造をポインタで共有する．
② リテラル構造をコピーする．

の2通りが考えられる．①を用いれば記憶量の削減に役立つ．一方②は，マルチプロセッサの制御が簡単になる．又，プロセッサとゴールプール間の授受単位としては，

① 1つの完全なゴール
② あるゴール内の1つのリテラル
③ 記憶セル（例えば40bit）

の3通りが基本的なものである．①は制御が単純であり負荷分散が容易であるが，プロセッサ／メモリ間の転送量が多くなる可能性がある．②はゴールのデータ構造とも関連が深く，リテラル共有と合わせて使用すると能率が向上することが期待できよう．③は通常の逐次型マシンで使われている方式であり能率が良いが，高並列なマシンに用いる場合，プロセッサ／メモリ間結合網がネックになる可能性がある．

一方，各リテラルに含まれる引数は，単なる1つの数値だけに止まず，複雑で大きなデータ構造となり得る．そのようなとき，データ構造を各リテラルの引数からポインタを用いて共有するようにすると，記憶の節約になるしヌ，プロセッサがメモリをアクセスするときデータ構造を置いてきぼりにすることによって転送量が減少することが期待できる．

勿論，このデータ構造の共有と，前記リテラルの共有とは共存可能である．図2にその様子の一例を示す．

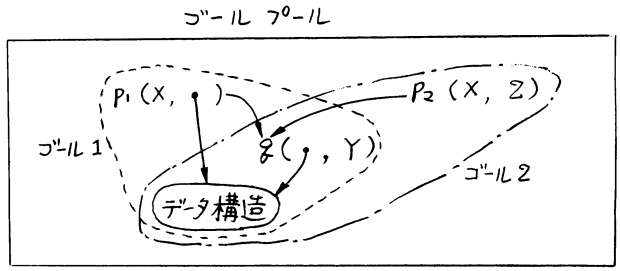

図2 ゴールのデータ構造

共有されるデータ構造はリテラルを記憶するのと同じメモリに収めることもできるし，ヌ，独立した記憶（構造記憶）とすることも可能である．

3.5 制御アルゴリズム

同一のマシンアーキテクチャでもその細かな制御方式によって特性が異なる．代表的な制御項目をあげれば次のようになろう．

① ゴール選択方式
② 負荷分散方式
③ 高階機能実現方式
④ 全体制御方式

①は，ゴールプール内にある多くのゴールの内，どれを先に処理すべきか，又同一ゴール内でもどのリテラルから処理を始めるべきかの選択である．文献(1)に示すように，方式としてはFIFO，depth-first，breadth-first，conclusion-precedence等が考えられるが，総合的に優れているのはconclusion-precedenceであるという報告もある．

②は，多くのプロセッサを能率良く働かせる為の方式で，メモリ競合等を避ける為にゴールプールが多くのメモリバンクで構成されることを考慮するとき，各プロセッサはどのバンクから仕事をアクセスすべきか，又，処理結果として生じる幾つかの新ゴールをどのバンクに収めるべきか等が問題となる．これについても文献(1)のデータがある．負荷分散を新ゴールの生成時のみに限ったとき，新

ゴールは，若しローカルなメモリバンクがある程度以上空いていた場合はローカルなバンクに，そうでなければ他のバンクにランダムにばらまく方式（empty-self と称する）が勝れているといわれている．その他，空いているプロセッサが仕事要求をリングバスで出し，忙しいプロセッサは自分の仕事を切りの良い所で分割するという方式（文献(2)）も提案されている．

③は，NOT，set_of，bag_of 等のいわゆる高階機能を実現することで，通常のプログラミング言語としては単なる pure Prolog では不充分であり，この種の機能を取り込むことが必要である．一般に，pure Prolog だけを対象とする場合に比して高階機能を取り入れようとすると，制御が複雑となったり，元のアーキテクチャのままでは不可能であったりする．実用マシンとして広く使われる為には最初の内からこれらの機能の実現可能性を考慮しておくべきである．

④も③と同種の機能であるが，高並列マシンを実現する場合，制御上の都合から幾つかのゴールの実行を止めたり，あるユーザジョブに割当てるべき資源の量を制御したりすることが必要となる．マシンはこれらにも柔軟に対処し得る能力を備えておらねばならない．

3.6 その他の機能

前節で述べた各種の制御アルゴリズムは，全自動ですべてマシンが状況を把握しながら実行することも考えられるが，それでは限界があるし望ましいことでもない．むしろ必要な所にはプログラマの手を入れることができるようにするべきであろう．例えば，変数の結合順序に関する指定としての Read-only-annotation，AND や OR 条件に対し逐次処理すべきか並列処理すべきかの指定，ゴール選択に対するユーザ指定等で，これらを組み込み易い形としておく必要があろう．

又，マシン動作中の各種の最適化処理，例えば，ゴールを構成するデータ構造中の不要な中間変数の除去，解くべきリテラルが決定的である場合（算術演算等）の転送省略等がある．

更に，統合化操作を続けてゆくとき，過去の統合化失敗の経験を記憶しておいて，そのパタンが生じればそのゴールを消去するということも考えられる．文献(3)によると，その機能を組み込むことにより統合化操作の回数を大幅に減らし得る可能性のあること，更にそれに必要なオーバヘッド時間や記憶容量の増加は十分少ない等の結果が得られている．

3.7 物理構造

上記諸方式を実現するハードウェアアーキテクチャの設計問題である．これに対しては以下のような項目がある．

① モジュール構造
② 結合網
③ プログラム記憶
④ パイプライン制御
⑤ ガーベッジコレクション
⑥ 中央制御装置

①は，高並列マシンを作るとき，何をモジュールとして網を作るかということで，モジュール内はそのローカリティを生かした高速処理，モジュール間はそれより

も応答性が遅くても済む仕事という切分けの問題である．当初はマイクロコンピュータに近いものをモジュールとして使うことが多く，内部の動きが明快になるにつれてよりこの処理にTuneした構造が検討されよう．

②は結合網に何を使うかという問題で，候補としては，バス，多段結合網，リング，等がある．バスは，比較的少数のモジュールを結合するのに適し，多段結合網は同時に複数パス上を高スループットで結合する場合に適し，リングは資源分配等の逐次制御を必要とするものに向く．並列結合数が10数以下でトラヒックが比較的低ければバスで可能であるが，数10台の結合とか高トラヒックの場合は多段結合網，100台以上となると階層構造が必要となろう．高並列マシンの総合性能に対して，これらの網をある程度常識的に選べばよく，細かな網の詳細は余り効かないと言われている．即ち，網のトポロジよりもむしろその制御法や実装法をより考慮すべきであろう．しかし，網自身の機能として負荷分散機能を持たせたり，障害に対する機能を持たせることは有効であろう．

③は，各プロセッサで推論を実行するとき必要となるプログラム（ruleとfact）を置いておくべき記憶の構成である．各プロセッサで必要とするので，各プロセッサの近くにプログラム記憶を置きそれに同一プログラムのコピーを置くことが当面最も容易で且つ実現性のある（1MBのメモリでも十分安価）方式であるが，将来的には大容量の知識ベースから必要な部分を抽出する方法や，cacheのように動的に必要とするものを収めておく機構の検討が必要である．

④は，一巡の処理を何段かのステージに分け，パイプライン的に実行することで，いずれ各所の動作が明らかになればシステムの性能向上策として考慮されるべき項目である．そのステージとしては例えば

i) ゴールデータのフェッチ
ii) 統合化候補節の選択
iii) 各変数の統合化
iv) 新ゴールデータの生成
v) メモリバンクへの転送

が考えられる．

⑤は，ゴールデータや構造データ等，可変長のデータがシステム中で生成・消滅を繰返す．従ってガーベッジコレクションが必要となる．複数メモリバンクが並列に使用される（互にポインタで結ばれながら）状況でのガーベッジコレクションとなり，その制御は一般に複雑である．従って可変長データは集中化して1ヶ所で管理するとか，出来る限り固定長で済ませる等の工夫が為されるが，より一般的な状況下での解決策は今後に待つ所が大きい．

⑥は多くのモジュールからなるシステム全体を統括する機構のことであるが，分散制御で可能な所はできるだけ分散して実行し，必要最小限（零ではない）の部分のみ集中した制御とするべきであろう．並列度が高くなると，各モジュールの集中制御に対する負荷は小さくても，統合すればかなりの負荷となり得るので注意が必要である．

4. 並列推論マシンの研究例

主として並列推論を目的としたマシンの研究例について述べる．研究は，データフローマシンとリダクションマシンに分けて紹介する．

4.1 データフロー マシン

一般のデータフロー マシンの研究は，数値計算を目的としたものが多いが，以下に述べるのは推論処理や記号処理を主目的としたものである．

① PIM/D

ICOTで開発中のマシンで，pure Prologや Concurrent Prologをコンパイルして動かす．本年中にプロセッサ8台，構造メモリ8台のシステムが稼動する予定．

② EM-3

電総研で開発された Lisp 処理向きデータフローマシンで，8台のプロセッサからなり，EM-Lisp を EMIL に変換して実行する（1984年稼動）．

③ DFM-S

武蔵野通研で開発中のマシンで，データフロー制御による OR 並列，AND パイプラインをインタプリタ形式で実行する．プロセッサ8台，構造メモリ8台の試作機が出来上っている．

④ プロセス グラフ モデル

電総研で提案されたモデルで，リテラルレベルでのデータフロー制御による OR 並列処理モデルで，Prolog の節に対してプロセスを対応させている．

⑤ Paralog

東大で試作されたもので，TOPSTAR II と呼ぶデータフローマシン上に，OR 並列の Prolog が実装された．

4.2 リダクションマシン

リダクションマシンの研究として以前より（一部は1971年）行なわれているものには，Berkling マシン，Mago マシン，AMPS，SKIM，ARM，ALICE 等があるが，これらは FP 等の関数型言語や Combinator を対象としたものである．ここで述べるものは，主として論理型言語を対象にしたものである．

① AND/OR プロセスモデル

UC Irvine で提案されたもので，論理型プログラムの並列実行のモデルであり，1981 年に発表された．

② PIE

東大で開発中のマシンで，ゴール単位の処理と集中化した構造記憶が特徴である．シミュレーション結果が発表されている他，モジュール1台が試作された．

③ PIM/R

ICOTで開発中のマシンで，リテラル単位の共有データ構造と，Concurrent Prolog のサポートに特徴がある．現在8プロセッサのシステムを試作中である．

④ ALICE

Imperial College of London で研究中の関数型マシンであるが，Parlog等の論理型言語の実装も考えられている．Transputer が使われる予定である．

⑤ OR-Parallel Token Machine

スェーデンの Royal Institute of Technology より提案されたマシンで，OR 並列を取り出すことを目的としている．1983年発表．

⑥ 株分けモデル

富士通研と ICOTより提案されたモデルで，逐次型マシンの並列実行という形をとる．16台のプロセッサよりなるシステムを開発中である．

⑦ 並列リダクションモデル

京大で研究中のマシンで，OR 並列とストリーム並列の支援を目的とし，OR 処理と AND 処理，及びデータベース処理それぞれに異なったハードを用いる．

5. おわりに

高並列推論マシンに関係の深い他の研究としては，Columbia 大学の Dado マシン，MIT の Connection Machine 等がある．Dado マシンは production system を高速に実行するマシンで，現在 1023 台のマイクロプロセッサによる Dado2 を開発中である．Connection Machine はセマンティックネットワークをハードウェアで実現することの提案で 10^6 台位のモジュールが想定されている．

以上のような状況から現在を評価すれば，並列推論マシンの研究は未だ第1段階が済んだ所である．論理言語の振舞が明らかになりそれをハードウェアで支援する様々なアイデアが出始めたばかりであり，マルチプロセッサによるハードウェアシミュレータもようやく動き始める状況にある．しかし，並列推論の諸統計データは評価プログラムが乏しいこともあって未だ十分であるとは言えない．

今後の研究方向としては，諸並列推論モデルの評価，評価プログラムの収集，多数台プロセッサよりなるハードウェアシミュレータの完成，各モジュール内部構造の検討がまず行なわれる必要がある．勿論，論理型言語の研究は基本であり単なる pure Prolog に止らずより実用的本格的な言語の設定が重要なことは論をまたない．勝れた言語があって始めて計算機アーキテクチャの具体的な目標が定まる．KBM との機能分担／融合問題と合わせて重要な課題である．

並列推論のアーキテクチャとしては前述のような本格的な並列マシン研究の他に，従来からのマルチプロセッサに基づく多重逐次型推論マシンというボトムアップ的アプローチも存在する．これらのマシンのどれが生き残るか予断を許さないが，将来，VLSI によって有り余るゲートを如何に使いこなすかは未だ未だ永続的な課題であり，並列マシン研究が1つのブレークスルーを与えることを期待したい．

［参考文献］

(1) T. Moto-Oka, H. Tanaka, H. Aida, K. Hirata, and T. Maruyama : "The Architecture of A Parallel Inference Engine - PIE", Proc. Intl. Conf. on Fifth Generation Computer Systems, PP. 479-488, Nov. 1984. ICOT.
(2) 久内，板敷，佐藤，増沢，相馬：並列推論処理システム－株分け方式－，第30回情報処理学会全国大会，1985.
(3) 松原，相田，後藤，田中，元岡：論理型プログラムの OR 並列処理における冗長計算の除去方式，ICOT Logic Programming Conference '84，9-4，March 1984.