解説B

非ノイマン型コンピュータ［I］

田中 英彦

田中英彦：正員　東京大学工学部電気工学科
Non-von Neumann Computers [I]. By Hidehiko TANAKA, Member (Faculty of Engineering, The University of Tokyo, Tokyo).

1. はじめに

現在世の中で使われているコンピュータは，ほとんどがノイマン型コンピュータである．1946 年に von Neumann が計算機方式として，プログラムもデータと同様記憶装置に置く方式，いわゆる蓄積プログラム方式を提案して以来，その優れた柔軟性，明快さ，使いやすさのために広く使われるところとなった．コンピュータアーキテクチャの研究もその改良という形をとって進められてきたが，ここ数年その問題点がクローズアップされるところとなっている．その要因は，コンピュータ性能に対する限りの無い要求，プログラミングの困難さに対処する方法の要求等であるが，直接的には，データフローマシンに代表される新しいマシンモデルの出現，関数型や論理型に代表される新しいプログラミングスタイルの提案等がその引金となっている．本解説では，このような問題点を解決することを目指した新しい枠組，非ノイマン型コンピュータ研究の概要を 2 回に分けて紹介する．

2. では非ノイマン型の意味を述べ，**3.** ではコンピュータを考えるときに重要な計算モデルの分類と概要を示す．続く三つの章は具体的なマシンの紹介で，**4.** はデータフローマシン，**5.** はリダクションマシン，**6.** はその他の非ノイマン型マシンを述べる．**7.** は本文のむすびである．これらのうち 4 章までが今回の内容で，以降は次回で述べる．

2. 非ノイマン型コンピータ

2.1 ノイマン型コンピュータの特徴

いわゆるノイマン型コンピュータの特徴は，以下のとおりである[1],[8]．

① 蓄積プログラム方式
② 制御駆動逐次制御方式
③ 線形アドレスの記憶装置
④ 命令に対するデータの従属性
⑤ 固定的なハードウェア構造
⑥ 決定的処理

①はマシンの汎用性に重要な役割を果たした特徴で，②は，命令の実行権が次々とプログラム内を移動してゆく方式を指す．すなわち，プログラム内の命令は，一つずつ順次実行されるが，次にどの命令が実行されるかは，現命令の内容とその実行結果および計算機の内部状態等で定まる．③はメモリが格納位置だけでアクセスされる単純な装置であることを意味し，④は，処理の制御が命令だけで行われ，各データ語の種別や意味はそれを使う命令の流れの中に暗黙に埋め込まれており，メモリ内にその種の情報は置かれていないことを意味する．⑤は命令セットが固定的であることを，⑥は，計算機の動作が決定的であって，確率的ではないことを述べている．

このような特徴は，計算機をさまざまな目的で使う汎用性，計算機構造の単純明快さ，処理の再現性等の点で計算機を広く普及させるのに役立った．しかし，現在は素子技術が発展し，従来程，ハードウェアのコストや複雑さは問題にならなくなってきているほか，計算機の応用も単なる数値計算から文字列の処理やパ

ターン処理，更には知識情報の処理が重要視されるようになってきているが，このような新しい応用に，従来の計算機構造が必ずしも適しているとはいえない．

一方，処理の高速性や扱うデータの多量性に関しては要求に限りが無い．計算機の高性能化やコストの低減は，それがまた新しく応用分野を開拓する原因となり，拡大再生産の様相を呈している．しかし，従来のアーキテクチャでは，プロセッサとメモリ間のデータ転送速度で計算機の処理速度上限が抑えられ（von Neumann ボトルネック）るほか，プログラミングの容易化にも限界があるといわれている．

2.2 非ノイマン型コンピュータ

前項で述べた状況を打破するための新しい枠組として期待されているのが非ノイマン型コンピュータである．これは，新しいアーキテクチャという言葉とほとんど変わらない意味で用いられることもあるが，ここではもう少ししぼって，ノイマン型の特徴と対比させながら述べることにする．

非ノイマン型への展開には多くの研究があるが，計算機制御方式や記憶装置の変革という形をとるものがほとんどである．ノイマン型は，制御駆動で逐次制御を基本としているが，命令の駆動方式からみると他にデータ駆動，および要求駆動がある．これらの駆動原理は，並列処理をうまく制御するのに適していると考えられている．

記憶の変革という意味からは，データにその種類等の自己定義記述を含めたタグやデスクリプタの利用，更に，連想機能を基本にした連想処理マシン等があげられる．その他，知識構造の一表現形式である意味ネットワークを直接数多くの要素素子で実現する試み等も従来のノイマン型のボトルネックを回避しようとした研究である．

ところで，同一のコンピュータでもそれを見る観点によってノイマン型に見えたり，非ノイマン型に見えたりすることがある．すなわち，コンピュータアーキテクチャを規定するマシンのレベルの問題で，マイクロコードレベル，機械語レベル，仮想マシンレベル等，各レベルでマシンを眺めた見方が可能である．

3. 計 算 モ デ ル

3.1 計算の構造

計算機の中で“計算”が進行する状況は，“計算”の構成要素となる仕事の各々を命令とよんだとき，一般に，実行対象となる命令の選択，次にそれが実行可能であるか否かの検査，そして可能な命令の実行という3フェーズの繰返しと考えることができる[2]．選択フェーズの動作を定めるのは計算ルールとよばれるが，その代表的なものは次の三つである．

① 強制選択，② 最内選択，③ 最外選択

①は，プログラムカウンタで命令を指定するような通常の計算機のルールをいい，②は，例えば関数 $f(g(k(a)))$ の値を求めるのに，最も内側の計算 $k(a)$ から順に計算（評価するという）することを指す．③は，これとは逆に最も外側の計算 $f(\ )$ から始めるやり方を指す．

次の検査フェーズを規定するものは，命令の発火規則であり，演算命令の場合はその必要とするオペランドすべての準備ができていること，条件命令の場合は条件の準備ができていること等がその内容である．発火可能であればそれは次の実行フェーズに移されるが，そうでない場合は，その命令の実行を遅らせたり，必要なオペランドに対する要求を出したりする．実行フェーズは通常の実行である．

一方，命令の実行が次の命令を定める機構（駆動原理）に着目してマシンを分類すれば，制御駆動，データ駆動，要求駆動の三つになる．制御駆動とは前述のようにプログラムカウンタを用いる方式である．データ駆動では，各命令はそのオペランドの到着を待っており，それのそろった命令は他の命令とは無関係に実行される．すなわちこれは，すべての命令を選択する（実行の対象とする）と共に，オペランドの準備完了を発火ルールとした場合，または，計算ルールに最内選択を用いた場合（引数がすべて評価ずみの命令を選択）と考えることができる．要求駆動では，ある命令を実行しようとしたとき，それに必要なオペランドを生成する他の命令に要求が伝搬する．従って，計算ルールとして最外選択を用いた場合に相当する．先程の例では，$f(\ )$ の命令がまず選ばれ，それには $(\ )$ の中の値が必要なので $g(k(a))$ に処理要求が出され，次に $k(a)$ に要求が出される．ここに至って，a は既知なので $k(a)$ が実行されて値が求まり，次にその値が $g(\ )$ に送られるということを繰り返し最終的に $f(g(k(a)))$ の値が求められる．

3.2 プログラム構造

ここでいうプログラムとは，今着目しているマシンの“機械語プログラム”を指すが，その要素である命令がデータを他の命令に引き渡すデータ機構には，基本的に次の2方法がある．

① by value（値渡し）
② by reference（アドレス渡し）

①は実行時生成された値を動的にコピーして必要な所に配る方法，②は，メモリ内に値を保持し，そのアドレスを他の命令に与えることで値を渡したことにする方法である．②はメモリ使用効率が良いが，並列実行する場合は注意が必要である．①は逆に，並列実行が容易であるが，オーバヘッドが大きくなる可能性がある．

3.3 代表的な計算モデル

計算機を計算モデルで大まかに三つに分けると次のようになろう．

① 制御フロー計算機，② データフロー計算機，
③ リダクション計算機

制御フロー計算機は，制御駆動を駆動原理とするものである．逐次処理が基本であるが，複雑なデータ構造をメモリ上で操作するのに便利で，非常に柔軟性に富んだ複雑な制御が可能である．データフロー計算機は，データ駆動，値渡し，が基本であり，単純な処理を高並列で実行するのに適しているが，データ構造の共有，逐次制御等には工夫を要する．しかし，最近はアドレス渡し，またそれを用いた要求駆動等の導入も可能となっており，柔軟性が増してきている．

一方，リダクション計算機は，式の書換えをベースに計算を行うもので，駆動法は要求駆動，データ駆動どちらも可能であり，データの渡し方としては値渡し，アドレス渡しどちらでも可能である．値渡しの場合を特にストリングリダクションとよび，アドレス渡しをグラフリダクションとよぶ（図1）．関数型言語との親和性に富み，並列処理にも向いていると考えられる．

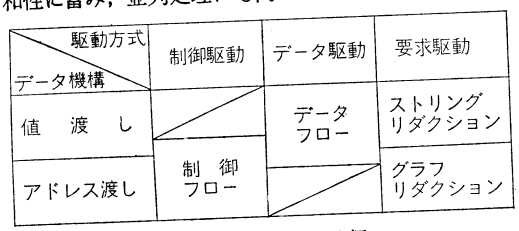


図1 計算モデルの分類

4. データフローマシン

4.1 概 要

データフローの概念は古い．1960年代後半におけるプログラム最適化や並列処理の研究分野では，多くの研究がデータ依存関係解析を基礎に置いていた．しかし，データフローアーキテクチャの提案は少し遅れ1974年のMITのJ.B. Dennisによるもの[7]が最初であるといわれている．その後，米，英，仏，日等各国において活発な研究が行われた．

研究は，データフロー言語，制御方式，アーキテクチャにまたがり，データフローマシンの応用分野も偏微分方程式等に代表される科学技術計算から，最近ではリスト処理等の非数値処理に至るまでその汎用性が検討されている．

データフローマシンの動きを規定する記法としてよく用いられているものは，グラフ方式の言語で，それをデータフローグラフとよんでいる．これは通常の計算機の機械語に相当する．例えば図2はその一例で，a, b を数値としたとき，それらから $(a+b)*(a-b)$ を計算する手順を示している．上方から値 a, b が入ってきて，それらから $a+b, a-b$ を X, Y として求め，次にそれらを掛け合せて結果を得ている．ここで，a, b, X, Y, Z というのはグラフ上の線の“名前”であり，その線上を“値”が流れてくると考えて，その値一つ一つをトークン（token）とよぶ．注目すべきは，図2中の四角で示される各演算で，それらは入力端子のトークンがそろえば開始される（発火する）ことである．すなわち，他所の動作には無関係に，その局所的なデータの存在だけで動作が規定される．この例では，＋，－の演算は a と b のトークンがそろえば開始され，＊の演算はそれらの結果が出そろったところで始められる．従って，＋と－は並列に実行可能である．

4.2 制御方式

データ駆動のマシンを実現する上で重要な設計上の選択項目は次のようなポイントである．

（1） 処理の起動方式

データ駆動における原則は，各演算の起動を，それに必要な全データがそろったことで行うことであるが，それを拡張することができる．一つは，一部のデータが到着しさえすれば全データがそろっていなくて

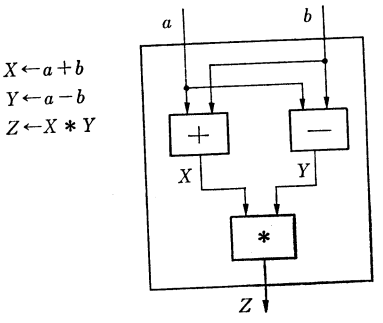


図2 データフローグラフ

も，できるところまで先に処理を進めておくという拡張である．これは，先行評価（Eager Evaluation），部分実行，非同期式（Non-strict）等とよばれる．この方法の利点は二つの局面で現れる．一つは，複数の基本演算から構成される関数や繰返し構造の起動法に適用する場合で，関数の全引数がそろわなくても一部の到着によって処理を開始し，できるところまで先に処理をしておくことである．

例えば，関数が図2のような場合は両データ a, b がそろわなければ処理の開始ができないが，図3のような場合，Y の到着を待たずに X の到着だけで一部の実行が可能である．

一方，トークンとして流れるデータは，単なる一つの値だけでなく，一般には複雑な構造を持ったデータ構造である．このような場合，そのデータ構造内のすべての値が定まっていなくともそのデータ構造を使うことができることがある．例えば図4のように，二つの要素を結合（cons*）して構造 $[a, b]$ を作成し次の操作に引き渡す場合，次の操作で第1要素だけしか必要としなければ（リストの第1要素を取り出す car† 操作のように），第2要素 b の到着を待たずにその実行を開始できる．これを可能とするためには，図中の cons 時，値が求まっていなくとも それを収める 記憶

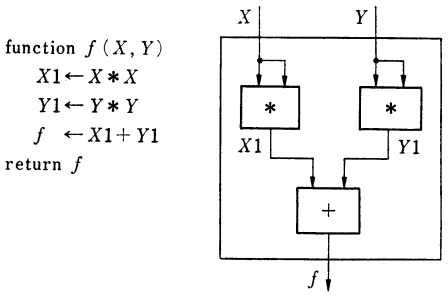


図 3　先行評価の例

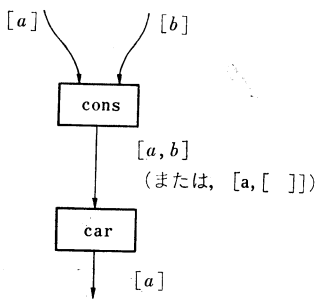


図 4　“不完全な構造データ”の利用

セルだけを確保し，空きのまま構造 $[a, [\ ]]$ を下に流してやればよい．続く処理はそれを用いて可能なところまで処理を進め，不可能になったところで待ちに入る．第2要素が何時か到着すればその時点で空きのセルに値が収められ，そこでまた待っていた処理を起動する．

このような不完全な構造を structure with hole[4] とか擬結果[12]等とよび，上述のような cons 操作を Lenient cons [11] という．これらは，等価的な並列性を増すのに大変有効であることが示されている．

ところで，上述のような先行評価方式は，処理の並列性を増すのに役立つが，むやみにそれを行うと無駄な処理が増えたり場合によっては永久に処理が止まらなくなることがある．計算機の資源は有限であるから，そのような無駄はできるだけ省く工夫が必要である．それには要求駆動が使われる．すなわち，図5のように，ある関数 f が実際にその値を必要としたときに，それを生成する関数の入力ゲートを開けて処理を起動するものである．これを一般に遅延評価（Lazy Evaluation）とよぶ．

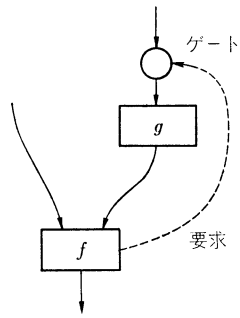
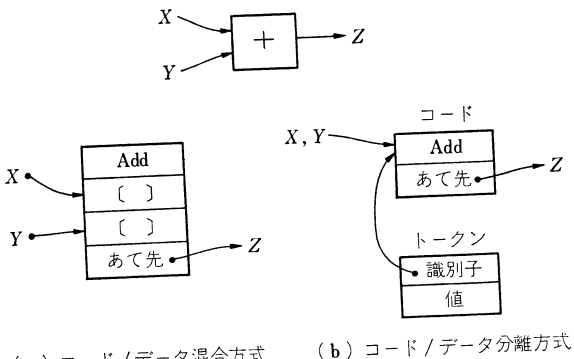


図 5　要求駆動の導入

（2）　プログラムコードの扱い

同じ関数を何度も使用する場合，それに対応するプログラムコードをどうするかが問題となる．これはプログラムの表現法と密接な関係がある．図6はそれを表しているが，同図（a）はコード内にトークンの値を入れる領域を設けた表現法，（b）はコードとデータとを分離した表現法である．（a）では領域の使用権管理が必要で，空き状況を前段の処理に伝えるとか，必要なだけこのコードを複製しておく等の工夫が要るが，簡単で固定的な処理に向いている．一方（b）ではコード部は常に一つしか存在しない．流れるトークン自身に識別子（色）を付けてプログラムのどの部分（繰返しプログラムでは何周目のループか等）に対応するかを区別する．プログラム内の各演算は，その識別子が同じトークンの間で実行される．再帰構造等，動的な処理に向いた方式であり，色付きトークン（Colored Token）方式とよばれる．

*† cons, car 等はリスト処理の基本操作のことで，詳しくはリスト処理向き言語 Lisp の参考書（42）等を参照されたい．



（a）コード／データ混合方式　　（b）コード／データ分離方式

図 6　データフロープログラムの表現2形式

4.3　マシン構造

データフローマシンの全体構造は，図7のように何台かのプロセッサエレメント（PE）をスイッチ網で結合した形をとる．各PE自身の一般的な内部構造は図8のとおりである．すなわち，命令コードとオペランドとを受け取って，各種の演算をする実行ユニット（EXU），その結果とそのときの状態とから発火命令を作り出し，また状態を記憶しておく命令制御ユニット（ICU），および，網とのインタフェースをつかさどる網インタフェースユニット（NIU）とからなる．

現在までにデータフローマシンは幾つか試作され，そのどれもが上記構成をとっているが，異なるのは命令制御ユニットに相当する部分の内部構造である．これは前述のプログラムコードのコピー方式か共有方式かで大きな影響を受ける．コピー方式の場合は流れるトークンにあて先が付いているのでそこに値を書き込むと同時に発火のチェックをすればよい．共有方式の場合はもう少し複雑になる．その一例を示したのが図9である．各トークンには，データフローグラフの線部分（リンク ID）に相当する識別子と，繰返し回数等からなるタグが付いており，まず前者から，そのあて先すべてをリンクメモリ内の表で求めて複数のあて先用トークンを作る．その各々に対し，タグをキーにして対応するオペランドの到着状況を探す．オペランドメモリ（OM）はそれ用のメモリで，マッチングユニットはハッシュ等の手法を用いて連想探索を行う．対応オペランドが見つかれば，その命令の発火条件が整ったことになり，命令コードを IM から取り出して命令パケットを作りあげる．

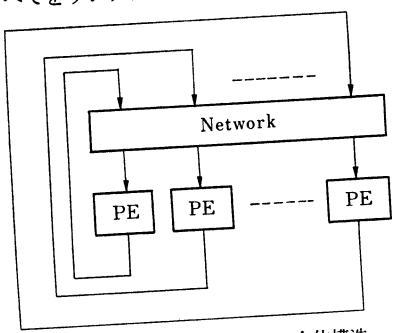
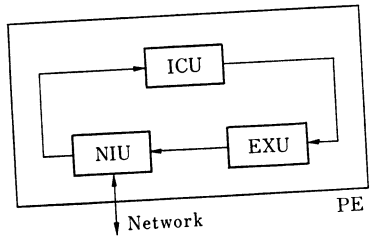


図 7　データフローマシンの全体構造



ICU：命令制御ユニット
EXU：命令実行ユニット
NIU：網インタフェースユニット

図 8　プロセッサエレメントの内部構造

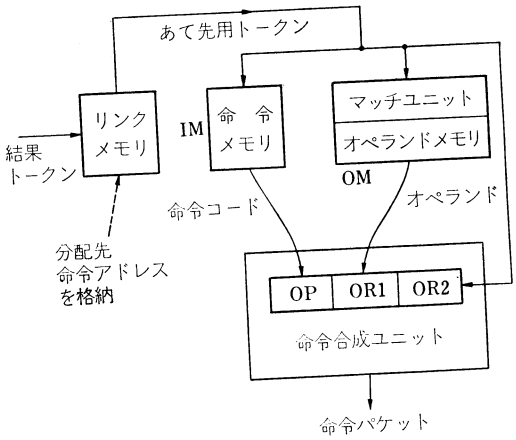


図 9　命令制御ユニットの構成例

マシン内を流れるデータが構造データのような一塊りのデータである場合，それを実際に流すよりもメモリ内に置いておき，そのアドレスだけを流した方が能率が良い．このために最近のマシン[11),(14),(16),(17)]では構造メモリ（SM）が設けられる．すなわち，構造メモリをPEと同様なものとして扱い，それに命令パケットを送り込んで内容のアクセス/更新等を実行する．

4.4　研究動向

（1）　データフロー用言語

データフローマシンの細かな動きを記述するために

は前述のデータフローグラフ（多くの変形がある）が用いられるが，データフロー用高級言語としては，1969年ごろ並列処理記述用言語として提案された単一代入言語（Single Assignment Language）に源を持つ幾つかの言語がある．単一代入とは，一つの変数への代入操作を1回しか許さないことで，変数の値は一つの代入文でのみ定まるため，複数の式の依存関係（計算順序）を見出しやすく，また並列性が向上する．一方，数学の関数概念に基礎を置いた関数型言語では，値に関数を作用させて別の値を求める言語スタイルをとるので，データが到着すると演算を開始するデータ駆動と相性が良い．

表1 データフローマシン用言語

言語名	研究者	研究所	年	特徴
Id	Arvind et al.	UCI	1978	型なし，動的処理
VAL	W. Ackerman et al.	MIT	1979	型付
CAJOLE	C.L. Hankin et al.	U. of London	1978	非手続言語
VALID	雨宮ほか	武蔵野通研	1981	適用型言語
EM-LISP	山口ほか	電総研	1981	Lispベース
Iconic アセンブラ	相馬ほか	富士通研	1982	ディスプレイ用言語

表2 データフローマシン試作機の例

マシン名	研究所	言語	稼動年	構造
DDMI	パロース＋ユタ大学	有向グラフ	1975	再帰構造
DDP	TI	Fortran 66	1978	4×PE
LAU	ONERA CERT	LAU	1979	32×PE（ALU）
Topstar II	東大	グラフ言語＋C	1980	16×PM＋8×CM
富士通 DFM	富士通研	Id-like	1981	4×PE（ALU）
D^3P	沖電気	DFグラフ	1982	4×PE＋SM
DFNDR-I	群馬大	グラフ言語	1982	4×PE（FU）
Manchester DFM	U. of Manchester	LAPSE	1982	12×PE（ALU）
EDDY	武蔵野通研	VALID	1982	4×4×PE
EM-3	電総研	EM-LISP	1984	8×PE
MEF	MIT	Id	1984	64（一部）×PE
DFM-S	武蔵野通研	VALID	(1985)	8×PE＋8×SMM
PIM-D	ICOT	CP	(1985)	8×PE＋8×SMM
SIGMA-1	電総研	DF-C	(1987)	180×PE

現在までに，データフロー用言語として提案され使われた言語の例を表1に示す．これらのほかに，データフロー用そのものではないが，並列処理記述用として提案されデータフローマシン上に実装が試みられている言語として Concurrent Prolog[48] がある．

（2） データフローマシン

現在までに幾つかのデータフローマシンの試作が行われている．その例を表2に示す．表から明らかなように，我が国における研究は活発である．特に，その対象応用として，数値計算のみならず，リスト処理，知識処理が多くなってきているところに特徴がある．また，試作機も，汎用マイクロプロセッサ等を用いてデータフロー機能をシミュレート（エミュレート）する形のものから，発火検出をハードウェアで支援するものまでさまざまであるが，ここに至ってアーキテクチャ的にはほぼ明確になってきた状況である．スーパコンピュータ用としても研究が開始されており，日本の SIGMA-1 のほか，米国 Illinois 大学の Cedar-32 H がある．後者は，通常のノイマン型マシンを基本とし，do ループにデータフローを適用することが考えられている．商用機としてはまだ現れていないが，特殊用途用のプロセッサにデータフローの考えを取り入れたものが発表されている．日本電気のイメージパイプラインプロセッサ μPD 7281 がそれで，画像を高速に処理することを目的としたものである．

4.5 データフローマシンの位置付け

従来の並列計算機の研究は，プログラムから並行に走らせうるプロセスをいかに見出すか，またそのプロセスを複数のプロセッサにいかに能率良く振り分けるかに力が注がれてきたといえよう．しかし，これらの研究は，プログラム自身に明白な並列性がある場合以外全くといってよい程成功しなかった．その原因は，並行プロセスの自動的な検出が難しかったからで，その結果としてプログラマに並列記述の負担がかかることとなった．

データフローマシンは，正にこのネックを解消する優れた枠組を与えたものと考えられている．すなわち，プログラマに負担を与えること無く，自然に並列性を抽出できるという点である．その意味で，従来の並列計算機研究にブレークスルーを与え，von Neumann 型マシンの特徴である逐次制御から抜け出す可能性を与えたと考えられている．

データフローマシンが商用機に広く用いられるか否かは予断を許さないが，今後何らかの形でその考えが取り入れられてゆくことは確かであろう．

（次号へつづく）