計算アーキテクチャ 36－2
（1979. 11. 28）

# データ・フロー計算機の周辺をめぐって

横井俊夫*, 島田俊夫**, 内田俊一***, 塚本享治+, 山口喜教**, 元吉文男*, 三国一郎*, 樋口哲野++
（電総研 パターン情報部*, 計算機部**, ソフトウェア部***, 制御部+, 慶応大工学部++）

## 1. はじめに

新しいプログラム言語、新しい方式の計算機を求める動きが顕着になってきた。電総研の中にも研究グループが作られ、活発な活動を行なっている。研究グループの議論の一端を紹介することによって、この動きの活発化への一助とするのが本稿の目的である。したがって、いわゆるサーベイではない。未消化は覚悟の上の方向付けや断定をも披瀝したものである。なほ、時間の都合で2章以降は、きちんとした文章化はされていない。

新しい計算機を求める動きを現在代表するものの一つがデータ・フロー計算機の研究である。そこで、データ・フロー計算機を核に、出来るだけ広く色々な話題をその周辺に関連づけることによって、その動きの全体像を浮び上がらせることにする。データ・フロー計算機に対する評価は様々である。それを是とする立場にも様々な色合がある。我々の評価、基本的な立場は次の様なものである。

〃従来の計算機（フォン・ノイマン型）は、逐次計算機構（プログラム・カウンタ）に基づいてハードウェアが作られ、その上にプログラム言語を含むソフトウェアの体系が組上げられてきた。これを並列（駆動型）計算機構に基づくハードウェアに置き換え、その上でソフトウェアの体系を再構築し直すことがその目的である。〃

この再構築によって得られるものとして、次の様なものを想定している。

(a) 自然で高度な記述系
誤りの少ない、効率の良いプログラムの作成を容易にし、すでに確立されているソフトウェア技術とうまく融合し、さらに高度化することができる。人工知能研究等の成果とも結びつき、高度な応用システムを作るための強固な土台を提供する。

(b) 並列処理による高速化
様々なレベルでの自然な並列計算の記述を可能にするため、無理なく並列処理機能を活用できるようになる。ただし、並列計算に関しては自然な記述系を求めるのが第一義で、すぐに高い高速処理機能を求めるのは二義的目標である。

(c) ＶＬＳＩ技術への適合性の良さ
ＶＬＳＩ技術を有効に活用できる。というより、ＶＬＳＩ技術の進歩によって、はじめて可能になったといって良い。

(d) 高い適応性
大規模なものから、小規模なものへと、各種の応用に適した規模へ自由に変化しうる計算機が得られる。したがって、古い意味でのsuper

computer を作るのではない。高い可変性をもった計算機方式を開発することである。

(e) 高い信頼度
(マイクロ)分散処理システムとして、障害に対し高い信頼性を持つようになる。今までに議論されてきた信頼性技術が真に生かされるし、また、より高度な技術を要求する。

重要なことは、上記の様な利点を得るだけの技術上の基本的な蓄積がすでになされているという判断である。

## 2. 駆動型計算機構

### 2.1. 基本単位(オペレータ)

< condition > → < action >

< condition > が成立すると対応する < action > が実行される。
< action > の実行は、自分自身を含めいつかのオペレータの < condition > を成立させる。

### 2.2. 分類のための属性

(a) < condition > や < action > を具体的にどのようなものとするか。
(b) < condition > の成立が、成立要件の消滅に結びつくか(副作用なし)、結びつかないか(副作用有り)。
(c) 複数の単位オペレータの < condition > が成立した場合、並列実行を主とするか、非決定的実行を主とするか。

### 2.3. 実例

データ駆動(data driven, data flow)

(a) < condition > : 計算に必要なデータがすべてそろう。
< action > : 計算し、結果(データ)を必要としているところに送る。
(b) 結びつく
(c) 並列実行 (data link による複数の送り先の明示)

要求駆動(demand driven, data flow)

[phase I flow graph の作成]

(a) < condition > : 計算結果に対する要求がある。
< action > : 計算に必要なデータを得るため、要求を送る。
(b) 結びつく
(c) 並列実行

[phase II flow graph の data driven による実行]

利点: ① 最終結果を得るに必要な flow graph のみが起動される。
② 基本データ構造を flow graph としてみると、データは本来要求駆動型とみた方が良い。

メッセージ駆動（massage driven, message passing）
(a) <condition>：必要なメッセージをすべて受け取る。
<action>：結果をメッセージとして受け手(複数)に送る。
(b) 結びつく。
(c) 並列実行（複数の受け手の内，受信用パターンにメッセージが一致したものすべてが起動される。）

見張付命令（guarded command）
(a) <condition>：共有変数に対する述語論理式。
(b) 結びつかない。
(c) 非決定的実行。

プロダクション・システム（production system）
(a) <condition>：作業メモリに対するパターン照合
<action>：作業メモリへの追加，変更，削除操作を含むaction系列。
(b) 結びつかない。
(c) 非決定的実行（回避規則による決定化）

Micro-Plannerの定理
(a) <condition>：どのような操作がデータ・ベース上に行なわれるか。
<action>：データ・ベースへの操作（検索，書き込み，削除）を含むプログラム。
(b) 結びつかない。
(c) 非決定的実行（後戻りによる実行）

2.4 整理
(a) 駆動条件
(1) 必要な情報の到着によって駆動される。
（データ駆動，要求駆動，メッセージ駆動）
(2) データ・ベースの状態（変化）によって駆動される。
（見張付命令，プロダクション・システム，Micro-Plannerの定理）

(b) 複数の受け手の実行モード
(1) OR（非決定的）：どれか
(2) AND：すべて

(c) 具体的な処理
(a)-(1)
(b)-(1) 条件文と見なすことができる。
(b)-(2) 並列実行

(a)-(2)

(b)-(1) 条件文、ORゴール
(b)-(2) 相互に矛盾が生じぬ場合は並列実行可（定理証明）.
ANDゴール.

3. メッセージ送受による計算の表現

3.1. actor（送受信し合う者）の階層

A. Network
① global network上の communicating sequential process.（csp）
② local network上のｃｓｐ.

B. Computer内
③ task, processといわれるｃｓｐ.
④ procedureといわれるｃｓｐ.

C. Computation
⑤ computationの構成要素としてのｃｓｐ.

(注) (a) ①から⑤にいくにしたがって、
単位actor：大→小
メッセージの大きさ：大→小
交信距離：大→小
交信速度：遅→速
交信頻度：少→多
(b) ①, ②においては、動的なactorの生成, 消滅は考えない。
(c) ③, ④をｃｓｐとしてみる見方は、monitor概念によって助長された。
(d) ①から⑤へと,交信の系列（会話）は無構造となる。したがって ①から④においては、会話の構造化と、取り出された会話構造とｃｓｐ間の関係、障害の事前検知、防止等との対応づけが重要な研究課題となる。

3.2. 送受信行為の階層化（abstraction）

Aにおけるメッセージの送受信行為
↑
Bにおけるメッセージの送受信行為
↑
Cにおけるメッセージの送受信行為

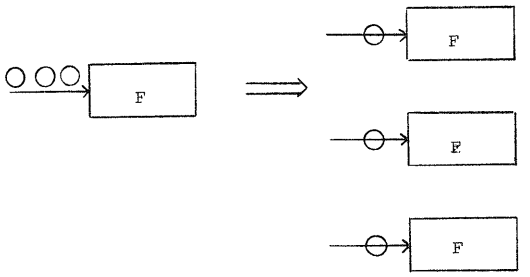
上位のメッセージの送受信行為は、下位の一連の送受信行為によって表わされるか、下位における、上位受信メッセージへの参照は、そのメッセージをactorと見なしての送受信行為として表わされる。
（例）
Aにおけるファイル転送は、Bにおける入出力動作の繰返しによって

表わされる。Bにおけるリストや配列の送信は、Cにおける数語の送受信行為の繰返しとしてか、リストや配列への参照は、それらをactorと見たてての送受信行為として表わされる。

## 4. 送受信機構

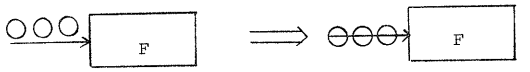
### 4.1. 複数メッセージの受信法

① 分身法




(例) 再帰呼び出し、色々な初期化によるactorの生成

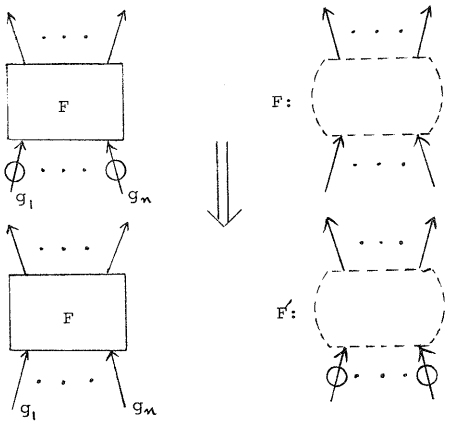
② 待合せ法




(例) マクロな待合せは、monitor, ミクロな待合せはパイプライン制御を表わす。

### 4.2. 送受信行為の対応づけ

① メッセージを実際に送信し、それを受信する。
flow graphで表わすと




関数計算で表わすと、

F(g1, ... ,gn)　　F: λ(x1, .. ,xn)<body>

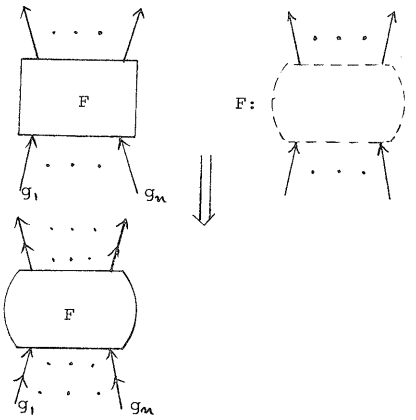
⇓

<body>[g1°/x1, ... ,gn°/xn]　g°=value of g

いわゆるapplicative order (=parallel)である。実現技術としては、SECDマシンのStackをHeapにし並列実行を行なうものと

考えることができる。

② 送受信行為を論理的に対応づける。
flow graphで表わすと，




関数計算で表わすと，

F(g1, ... ,gn)　　F:λ(x1, ... ,xn)<body>

⇓

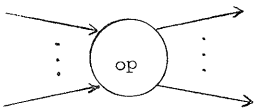
<body>[g1/x1, ... ,gn/xn]

いわゆるnormal order(+parallel)である。実現技術としては、置き換え(reduction, substitution)となる。

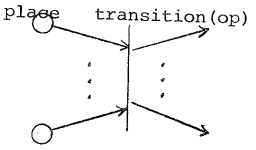
③ 受信要求を送信行為に伝える。
要求駆動となる。

5. 高度な記述系との対応
① 論理式 → PROLOG → 駆動型
② 等式 → 単一割当て → 駆動型
③ 数式 → 単一割当て＋(数式処理) → 駆動型
④ structured object(高度なデータ構造) → KRL → 駆動型

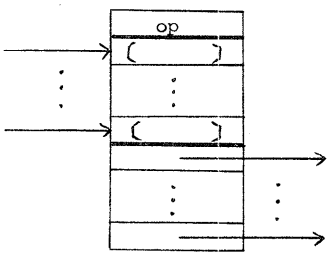
6. 計算の構成要素としての単位actor
① data flow actor




② Petri net



③ activity template (Dennis)




④ machine code (Dennis)

instruction

<op, destination>

destination

<address, input>

⑤ Π-式

L: ⫤ x1 ⋮ Then ⋮ ⫤ xn

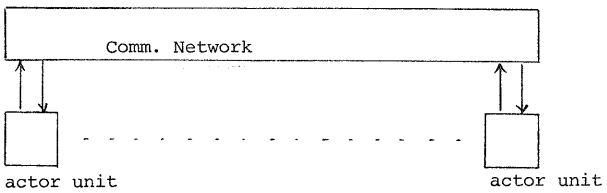
y1,C1 ⫤ R1 ⋮ ⋮ yn,Cn ⫤ Rn

(注) (1) 図的な表現は直観的に理解しやすいが、記号による表現の方が形式的な取扱いを容易にする。

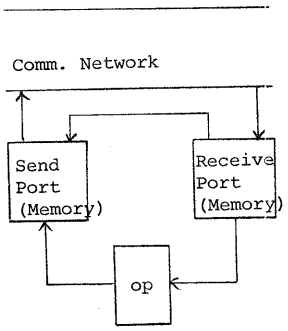
(2) 単位 actor と基本データ構造は同一であるべき。

7. 計算機の構造

① 単位actorの定義から直接得られるイメージ




actor unitは、

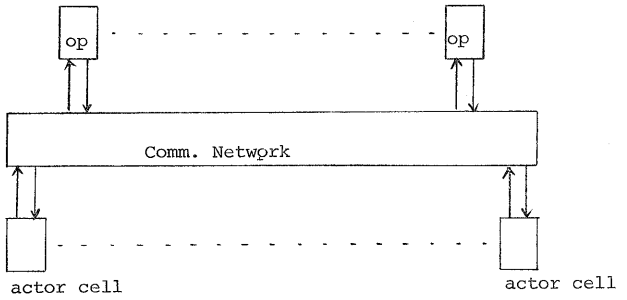



問題点

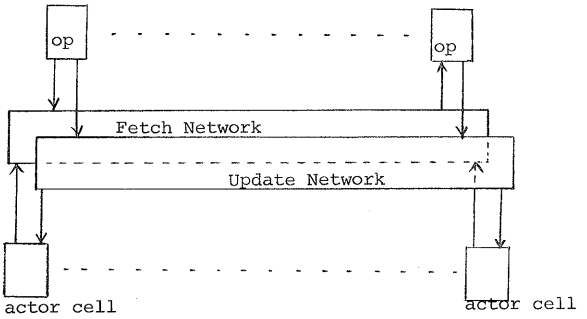
(1) 一般的で高速のネットワーク。

(2) actor unit の管理機構(配分, ゴミ集め)。

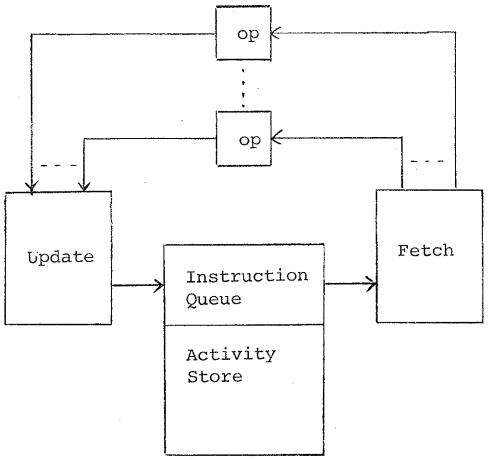
② op装置を共有化すると,




あるいは,

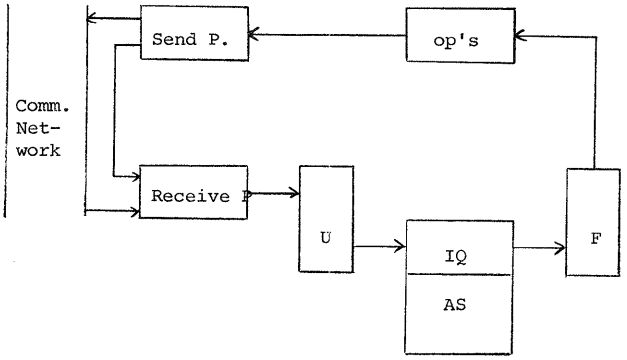



③ cellを1つにまとめると,



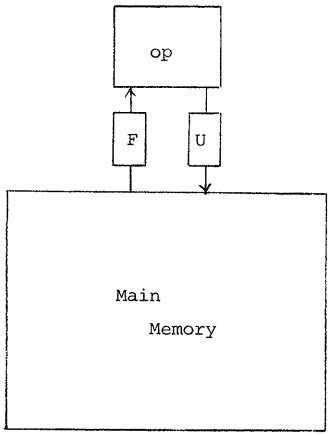

データの流れをパイプライン化するとMITのデータ・フロー計算となる。

④ ③を①のactor unitとみなすと,




⑤ ①の Comm. Network をどうするかで，色々の方式が考えられる。通信用計算機で階層状のネットワークを組むと Kellerマシンとなる．

⑥ 参考までに，従来の計算機上で考えると，




さて，①と⑥の違いは？