データフロー計算機による論理シミュレータ

深沢友雄　栗原謙　鈴木達郎　田中英彦　元岡達
（東大　工学部）

§1　はじめに

論理回路の設計の一段階として、シミュレーションは欠かす事のできない一段階である。しかし、対象システムが複雑になると、シミュレーション時間が膨大となるのが通常である。これは、論理回路の各要素が、各々同時に、並列に動作するのに対し、従来の計算機でシミュレーションを行なう場合、各回路要素の動作をシーケンシャルに処理せざるを得ない事が一因となっている。この為、本来論理回路とは無関係な制御が必要となり、対象回路の規模が大きくなると、この制御時間の増大によるオーバヘッドが大きくなる。

こういった従来の計算機の欠点を補うべく登場した計算機アーキテクチャの一つに、データフロー計算機がある。

データフロー計算機は、従来のコントロールフローに対し、「データの依存性」という中心概念を用いて、データの流れによって制御を行なう計算機ということができる。各処理は、その入力データが揃いさえすれば実行可能となる様に分散制御されるので、任意の並列性を取り出して高速処理を行なう事が可能である。従って、データフロー計算機を用いれば、論理回路のシミュレーションを、回路要素間の本質的な結合関係の制御のみで実行することができる。

更に、データフロー計算機では、プログラムを「データフローグラフ」と呼ぶグラフで表現する事ができるので、これと論理回路との自然な対応づけによって、論理回路の記述が行なえる。従って、大規模なコンパイラや、トランスレータなしに、容易に実行形式のプログラムを作成できる。

そこで、本研究室で開発した、プロシージャレベルデータフロー計算機"TOPSTAR"上に、論理シミュレーション用のシステムプログラムを実装したので、その特徴、使用例、評価・検討等について、以下に述べる。

§2　TOPSTARのアーキテクチャ

TOPSTARは、プロシージャレベルのデータフローによって分散制御される、マルチマイクロプロセッサシステムである。

TOPSTARのアーキテクチャの特徴は、以下の3点にある。

1) CM（Communication/Control Module）と、PM（Processing Module）の2種類のモジュール多数から成り、それぞれの役割は、ペトリネットの"place"と"transition"

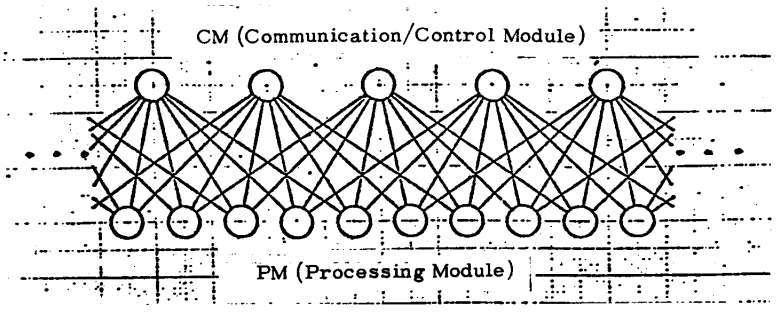


図2-1　TOPSTARのシステム構成

の役割に対応する。

2) CMとPMの結合は、大規模システム実現の為、部分結合とする。この際データフローネットワークとの対応を柔軟に行なう為、この結合をオーバラップさせ、可変構造を可能にしている。

3) プロシージャレベルのデータフロー処理においては、比較的多量のデータを扱うので、DMAによって、メモリ対メモリの高速転送を行なう。

図2-1にシステムの構成を示す。

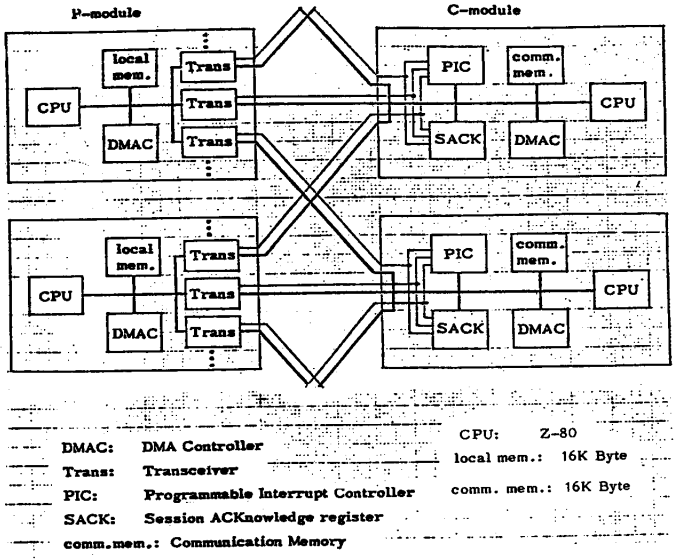


図 2-2 TOPSTARの各モジュール

各モジュールには、汎用のマイクロプロセッサ、Z80を使用している。(図2-2参照)

本研究に使用したTOPSTAR-Iは、PM3台、CM2台の構成である(現在、CM8台、PM16台から成るTOPSTAR-IIへ、システムの移行中である。)

§3 プログラムの表現

3.1 プロシージャレベル データフロープログラム

TOPSTARの処理対象は、プロシージャをノードとする、データフロープログラムである。図3-1(a)にその一例を示す。一方、このデータフロープログラムは、同図(b)のペトリネットと等価である。

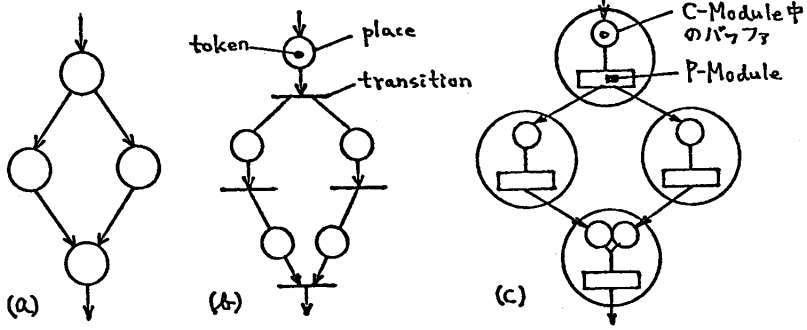
TOPSTARにおいては、このデータフロープログラムは、同図(c)の様に実行される。ペトリネットの"place"は、CM中のバッファに対応し、"transition"は、PMが「雇われて仕事を行なう」ことに対応する。

図 3-1

(a) データフロープログラムの例

(b) 対応するペトリネット

(c) TOPSTARにおける処理



3.2 論理回路ネットワークの表現

前節で述べたデータフロープログラムと、自然な対応づけを行なう事によって、論理回路ネットワークを表現する事ができる。すなわち、論理回路の各構成要素を、データフロープログラムの各ノードと対応させ、回路要素間の配線を、ノード間のリンクに対応づける。この時、ノードに対応する論理回路要素の規模は任意でよく、これを「モジュール」(注1)と呼ぶ事にする。

§4 TOPSTARの制御方式

TOPSTARにおける、論理シミュレーション用システムプログラムによる制御方式の特徴としては、以下の各点が挙げられる。

1) データ駆動による分散制御
2) PMの自由競争による負荷分散
3) パイプライン処理
4) ループ構造を許す
5) 変化しないデータの転送の軽減
6) 対象回路のクロックの管理

これらのうち、1)～3)については、既存のシステムプログラムV1以来のTOPSTARにおける基本的制御方式である。

本章では、これら1)～6)の特徴を中心に、制御方式を解説する。

4.1 システムの動作の概要

各PMは、(結合している範囲の)1つのCMに割り込みをかけ、(注2)DEQ(DEQueue)コマンドを送る事により、「仕事」を要求する。各CMには、幾つかのノードが割り当てられており、DEQコマンドを受けたCMは、管理テーブルを調べ、実行可能な仕事があれば、そのプロシージャ(注3)とデータ、及び結果の送り先に関する情報をPMに送る。実行可能な「仕事」がない場合には、その旨知らせる。

PMは、「仕事」が貰えなかった場合には、他のCMに尋ねて回る。「仕事」を貰ったPMは、その「仕事」を実行し、結果の送り先ノードが割り当てられているCMに割り込みをかけ(注2)ENQ(ENQueue)コマンドを送り(注4)、引き続いて、結果のデータを送る。idleになったPMは、再び「仕事」を求めて、CMに割り込みをかける。

以上の様に、各PMが「仕事」を求めて(オーバラップした部分結合の範囲で)自由競争を行なう事により、「忙しい」(実行可能な「仕事」が多く存在する)CMに、自然に多くのPMが集まり、適切な負荷分散が行なわれる。

ここで言う、「実行可能」とは、そのノードが必要とする全ての入力データが揃っている事を意味する。既ち、データ依存性のみによって制御が行なわれる、データ駆動方式が用いられている。又、データ依存性の管理は、各々のCMが単独に行なえ、集中的な制御は不要である。

4.2 フローコントロール

前節から、原理的には、CMが行なう制御は、入力データが揃った「仕事」を、次々とPMにDEQする事のみである。

しかし、実際には、結果の送り先のCMやのバッファは有限であり、これ

(注1) PM, CMを指す時のモジュールとは区別して考える。
(注2) この際の競合は、PIC(Programmable Interrupt Controller)によって解決される。
(注3) PMが前回実行したプロシージャと同一ならば、送る必要はない。PMは、そのプロシージャを優占指定する。
(注4) これにより、CMは、データの格納場所の計算を行なう。

がオーバフローしない様に制御する必要がある。これがフローコントロールである。

TOPSTARでは、各CMは、結果の送り先のCMのバッファに空きがある時のみDEQを行なう。この為、CMは、各ノード毎にセマフォを持つ。セマフォは、次段のノードの数だけ用意され、その値は、0から次段のノードのバッファサイズまでの値をとる変数である。DEQ時に、次段のノードに対応する各セマフォは、1減じられ(P-operation)、V-op(V-operation)コマンドにより増やされる。V-opコマンドは、次段のノード(を管理しているCM)で、バッファが幾つか空いた時、その空き数を、DEQを受けたPMが、前段のノード(を管理しているCM)に知らせるものである。

セマフォは次段のノードのバッファの空き数を示しており、CMは、セマフォの値が0の時にはDEQを行なわない。

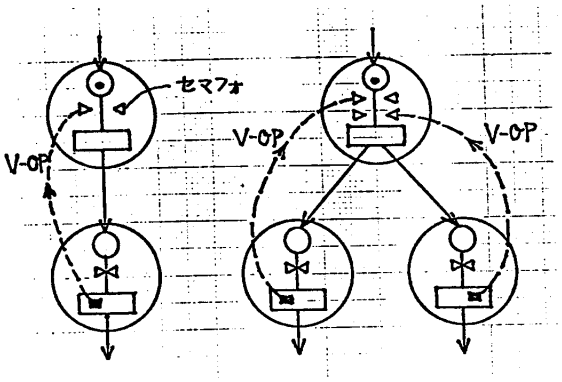


図4-1 セマフォによるフローコントロール

4.3 パイプライン処理と追い越し問題

マルチプロセッサシステムにおける並列処理性には、独立した演算が並行して行なわれる、いわば横方向の並列処理性の他に、パイプライン処理による、いわば縦方向の並列処理性がある。

TOPSTARにおけるパイプライン処理の場合、CMとPMに分かれている事により、複数のPMに順次DEQを行なっても、その処理結果が、DEQした順序に次段のノードにENQされるとは限らない。即ち、追い越しが起ってしまう。この追い越しを無制限に許すと、FORK-JOINの際にデッドロックを生ずる。(図4-2参照)

本システムプログラムでは、追い越しが起こっても、結果としてデータの順序が保存される様に制御する様にした。すなわち、各ノードの出力バッファ(次段のノードの入力バッファ)内のデータの順序が、入力バッファ内のデータの順序と同じになる様に、データを識別して、(いわゆるcolored-token)DEQした順序に応じて、queueとなっているバッファの指定した位置にENQする様に制御する。

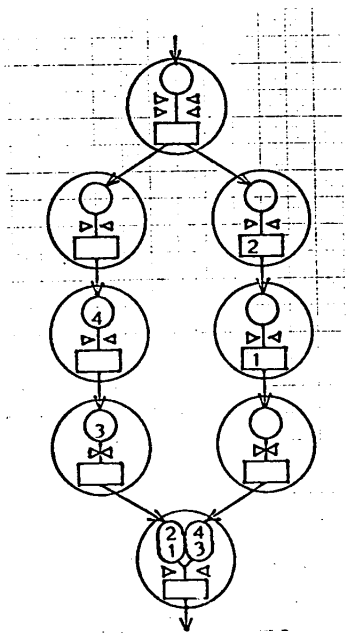


図4-2
追い越しによるFORK-JOINの際のデッドロークの例
(バッファサイズ=2の場合)

4.4 データ転送回数の軽減

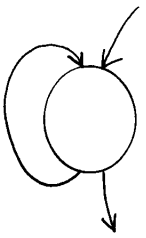
対象回路におけるクロックをデータとして扱った場合、シミュレーションクロック数に応じて、膨大なデータの転送が必要となる。一方、TOPSTARでは、DMA転送を採用している為、データの転送回数を減らさないと効率が落ちる。

本システムでは、DEQ時に入力バッファ内で引き続くクロックにおけるデータの値が等しい場合、それらをまとめて一回だけDEQする様にしている。この時、何クロック分まとめたかの管理をシステムが行なう。

4.5 ループ構造による記憶の表現

データフローマシンには、本来「変数」による「記憶」という概念はない。しかし、対象回路において、記憶回路は不可避である。

本システムでは、プログラムのループ構造を許しているので、図4-に示す様な、自分の出力を自分の入力とするノードによって記憶を持たせている。



記憶ノード

図4-3 記憶の表現

4.6 その他

前節までに述べた、制御の分散性により、対象回路に応じて柔軟に任意の並列性を取り出してシミュレーションを行なう事ができる。

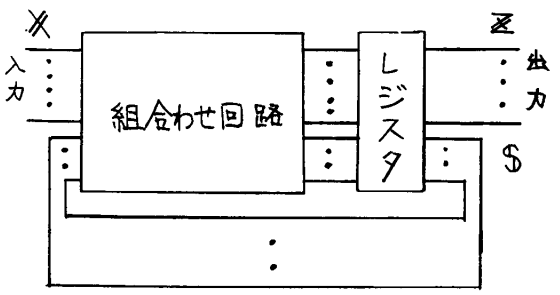
しかし、シミュレータとして見た場合、その出力を知る為の集中制御的な要素も必要である。本システムでは、出力を知る必要のあるノードから出力パスを取り出して、それを入力パスとする様な、「出力ノード」をユーザプログラムとして記述する事によって、これを実現する事にした。

§5 シミュレーションの方法

前章で述べた特徴を有するシステムプログラムを、TOPSTAR上に実装したので、その使用法、及び使用例を以下に述べる。

5.1 シミュレーションのモデル

シミュレーションの対象は、原則として、(5-1)式で示すブール式の形で記述できる2値論理モデルを各モジュールとする論理回路とする。



$$(Z_{i+1},\ S_{i+1}) = F(X_i,\ S_i) \quad \cdots(5\text{-}1)$$

図5-1 シミュレーションのモデル

5.2 回路の記述

回路の記述には最低限、次の2種類の情報が必要である。

1) 各モジュールの機能
2) 各モジュール間の結合関係

本システムでは、1)については、PL/Mで記述し、2)については、CMのテーブルにアセンブラで記述する様

になっている。

1)は、PL/Mの許す範囲内で、任意のプロシージャを記述する事ができるので、多レベルに渡った記述(例えば、ゲートレベルと機能レベル)が可能である。

2)は、アセンブラのマクロ機能を利用する事により、次節に述べる数種類のマクロ命令によって、容易に記述できる。

1)、2)の他に、前章で述べたシステム制御の為の情報の記述が必要であるが、これは、2)の記述の際のマクロ命令によって自動的に生成される様になっている。従って、ユーザは、システムの制御方式を殆んど意識する事なく、回路の本質的な部分の記述をすればよい。

尚、これらの情報は、全て(各モジュールに対応するノードを管理する)CMに記述し、PMには、何も記述しなくてよい。

5.3 CMのテーブル記述マクロ

ユーザがマクロを用いてCMのテーブルに記述すべき内容は、次の2点である。

1) プロシージャとモジュール(ノード)との対応関係
2) 各ノード間の結合関係

1)の記述の為に、次の2つのマクロ命令を用いる。

```
PNT   <<pn,<nn,･･･>>,･･･>
PROC  <<pn,pnleng>,･･･>
```

pn … プロシージャ番号
nn … ノード番号
pnleng … プロシージャ長(Byte単位)

PNTは、プロシージャと、ノードとの対応をリスト形式で定義する。各リストの要素(<pn,<nn,･･･>>)において、その外2要素(<nn,･･･>)のリストで列記されたノードが、外1要素(pn)の番号を持つプロシージャを処理する。各プロシージャは、PROCによって定義される。

2)の記述の為に、次の5つのマクロ命令を用いる。

```
NCB    nn, pn, mn
IN     nn, <<in,cn,<il,･･>>,･･>
OUT    nn, <<on,cn,<tn,･･>>,･･>
TMN    nn, <<tn,dl>,･･>
NCBE   nn
```

nn … ノード番号(定義名)
pn … プロシージャ番号
mn … マスク
cn … CM番号
in … インプットノード番号
on … アウトプットノード番号
il … インプットデータ長(Byte)
tn … ターミナル名
dl … アウトプットデータ長(Byte)

NCB(Node Control Block)は、ノードの定義命令であり、nnというノード番号を定義する。mnはマスクと呼び、定義したノードにおけるプロシージャの処理を、特定のPMにのみ実行させる為のものである。

INは、定義したノードの前段のノードを記述する為のものである。第2オペランドのリストの意味は、CM番号cnに管理されているノードinから(il+･･)Byteのデータが ENQされるという事である。

OUTは、定義したノードの次段のノードを記述する為のものである。第2オペランドは、CM番号cnに管理されているノードonに、出力ターミナルtnのデータをENQするという意味である。

TMNは、定義したノードの出力ターミナルの記述に用いる。第2オペラ

ンドは、thという名前を持つターミナルを定義し、そこにddByteのデータが出ているという意味である。

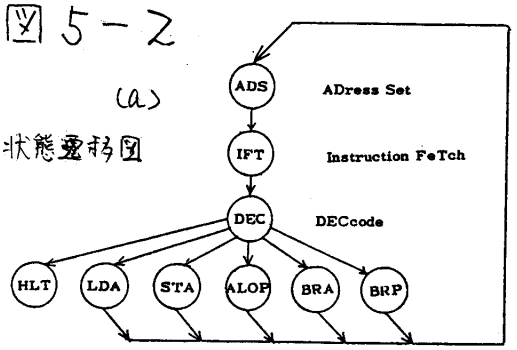
NCBEは、1つのノードの記述の終了を示す。

以上のマクロ命令によって、回路のネットワークを記述でき、同時に、システムが用いる制御テーブルも自動的に生成できる。

図5-2

(a)

状態遷移図



ALOP(Arithmetic & Logical OPeration)
=one of ADD or SUB or AND or OR or XOR.

5.4 シミュレーション例

設計途中段階の、簡単な計算機のシミュレーション例を述べる。

1) 対象回路例

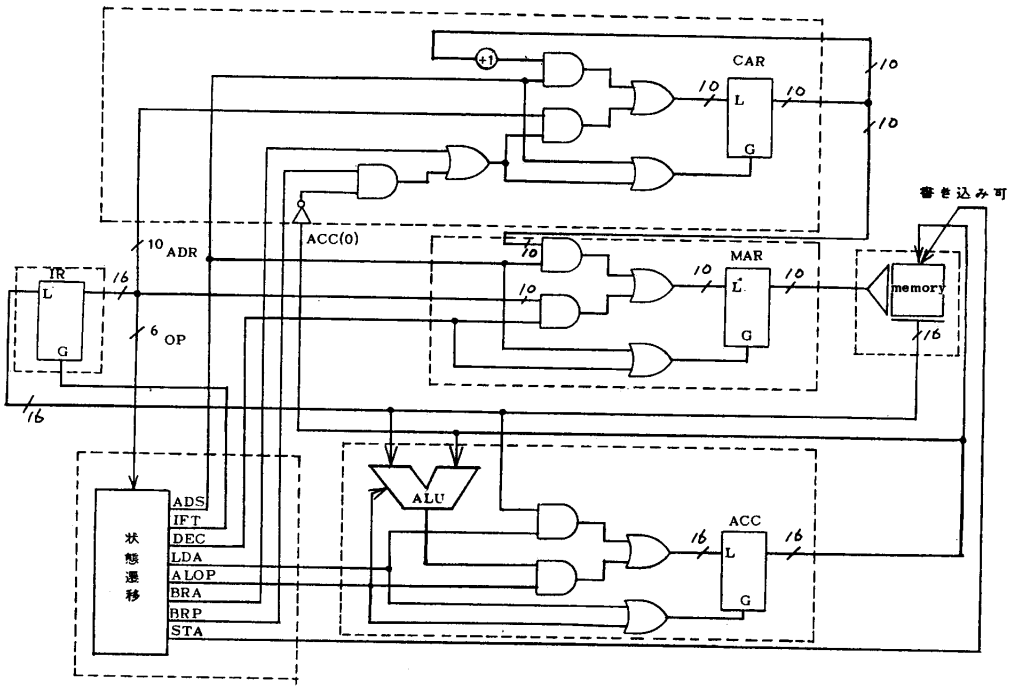
図5-2(a)に対象回路の状態遷移図を示す。同図(b)に命令表を示す。この計算機を、同図(c)の段階まで設計した時のシミュレーションを行なう事にする。この段階では、まだ状態遷移部の設計はゲートレベルまで進んでいない。

Operation Set

Mnemonic	OPcode	Operation
LDA	000100	ACC := (M)
STA	001000	(M) := ACC
ADD	010000	ACC := ACC + (M)
SUB	010001	ACC := ACC - (M)
AND	010100	ACC := ACC AND (M)
OR	010101	ACC := ACC OR (M)
XOR	010110	ACC := ACC XOR (M)
BRA	100000	PC := M
BRP	100001	IF ACC >= 0 THEN PC := M
HLT	111111	HALT

M = ADR

(b) 対象回路の命令セット

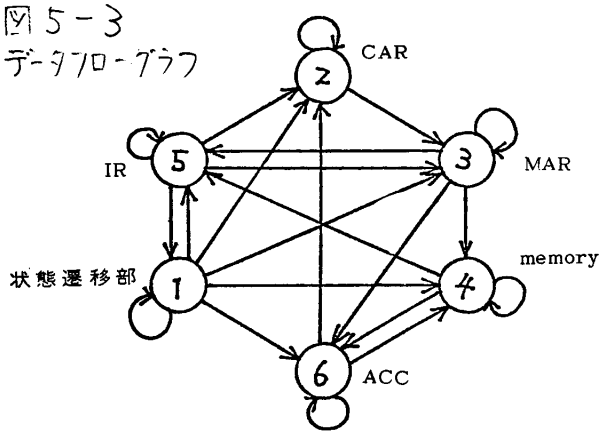


(C) 対象回路例

2) データフロープログラムとの対応

図5－2(c)の、点線で囲った部分を各モジュールとした場合の、対応するデータフロープログラムを図5－3に示す。図5－2(c)と図5－3とは、グラフとしては等価である。

図5－3
データフローグラフ



3) プログラムの記述例

i) プロシージャの記述

図5－2(c)の回路のMAR(Memory Address Register)モジュールをPL/Mで記述した例を図5－4(a)に示す。

ii) テーブルの記述

図5－3に示したノードのうち、ノード1～4をCMΦに割り付けた場合のマクロによる記述例を図5－4(b)に示す。

図5-4 (a) プロシージャの記述例

```
MAR:PROCEDURE;
   /* INPUT DATA */
DECLARE ADS ADDRESS;
DECLARE DEC ADDRESS;
DECLARE CAR ADDRESS;
DECLARE ADR ADDRESS;
DECLARE MAROLD ADDRESS;
   /* OUTPUT DATA */
DECLARE MARNEW ADDRESS;
   /*                */
ADR=ADR AND 03FFH;
   /*             */
MARNEW=(ADS AND CAR) OR (DEC AND ADR)
       OR (NOT(ADS OR DEC) AND MAROLD);
   /*                    */
RETURN;
END;
EOF;
```

図5-4 (b)

テーブルの記述例

● このテーブルの前部に、マクロの定義部と、変数の値の定義部がある。

```
  702:  ;
  703:          PNT     ((P1,(N1)),(P2,(N2)),(P3,(N3)),(P4,(N4)))
  704:  ;
  705:  ;
  706:  ;       NCB F8R N8DE1 (CBNT)
  707:  ;
  708:          NCB     N1,P1,M1
  709:          IN      N1,((N5,C1,(L51)),(N1,C0,(L1,L2,L3)))
  710:          8UT     N1,((N1,C0,(A,I,D)),(N2,C0,(A,B)),(N3,C0,(A,D)),(N4,C0,(
S)),(N5,C1,(I)),(N6,C1,(L,8)),(N7,C1,(A,I,D,L,S,8,B,H)))
  711:          TMN     N1,((A,L1),(I,L2),(D,L3),(L,L4),(S,L5),(8,L6),(B,L7),(H,
L8))
  712:          NCBE    N1
  713:  ;
  714:  ;       NCB F8R N8DE2 (CAR)
  715:  ;
  716:          NCB     N2,P2,M2
  717:          IN      N2,((N1,C0,(L1,L7)),(N5,C1,(L51,L52)),(N6,C1,(L61)),(N2,
C0,(L21)))
  718:          8UT     N2,((N2,C0,(CAR)),(N3,C0,(CAR)),(N7,C1,(CAR)))
  719:          TMN     N2,((CAR,L21))
  720:          NCBE    N2
  721:  ;
  722:  ;       NCB F8R N8DE3 (MAR)
  723:  ;
  724:          NCB     N3,P3,M3
  725:          IN      N3,((N1,C0,(L1,L3)),(N2,C0,(L21)),(N5,C1,(L51,L52)),(N3,
C0,(L31)))
  726:          8UT     N3,((N3,C0,(MAR)),(N4,C0,(MAR)),(N5,C1,(MAR)),(N6,C1,(MA
R)),(N7,C1,(MAR)))
  727:          TMN     N3,((MAR,L31))
  728:          NCBE    N3
  729:  ;
  730:  ;       NCB F8R N8DE4 (MEM)
  731:  ;
  732:          NCB     N4,P4,M4
  733:          IN      N4,((N1,C0,(L5)),(N3,C0,(L31)),(N6,C1,(L61)),(N4,C0,(L41
)))
  734:          8UT     N4,((N4,C0,(MEM)),(N5,C1,(MEM)),(N6,C1,(MEM)),(N7,C1,(ME
M)))
  735:          TMN     N4,((MEM,L41))
  736:          NCBE    N4
  737:  ;
  738:  ;
  739:          8R6     12FCH
  740:  ;
  741:  ;
  742:  ;       PR8CEDURE LEN8TH DEFINE
  743:  ;
  744:          PR8C    ((P1,PL1),(P2,PL2),(P3,PL3),(P4,PL4))
  745:  ;
  746:  ;
  747:          END
```

4) 出力例

図5－3(c)の対象回路によって、図5－5(a)に示すフィボナッチの数列を計算するプログラムの実行をシミュレートした時の出力例を図5－5(b)に示す。

出力ノードでは、対象回路が停止した時に出力を停止する様プロシージャによって記述してある。出力ノードの記述のしかたにより、任意のクロック、条件の下での、任意のノードの出力を見る事ができる。

図5-5 (a) 対象回路による演算例

```
Program of Fibonacci

PC    Machine Code    OP    ADR
00H   1021H           LDA   21H
01H   4020H           ADD   20H
02H   2022H           STA   22H
03H   8405H           BRP   05H
04H   FC00H           HLT
05H   4021H           ADD   21H
06H   2020H           STA   20H
07H   8409H           BRP   9H
08H   FC00H           HLT
09H   4022H           ADD   22H
0AH   2021H           STA   21H
0BH   8401H           BRP   1H
0CH   FC00H           HLT

Initial Data

ADR   Value

20H   0000H
21H   0001H
22H   0000H
```

CLOCK	CAR	MAR	IR	ACC	(20)	(21)	(22)	STATE
00000000	0000	0000	0000	0000	0000	0001	0000	/* ADS */
00000001	0001	0000	0000	0000	0000	0001	0000	/* IFT */
00000002	0001	0000	1021	0000	0000	0001	0000	/* DEC */
00000003	0001	0021	1021	0000	0000	0001	0000	/* LDA */
00000004	0001	0021	1021	0001	0000	0001	0000	/* ADS */
00000005	0002	0001	1021	0001	0000	0001	0000	/* IFT */
00000006	0002	0001	4020	0001	0000	0001	0000	/* DEC */
00000007	0002	0020	4020	0001	0000	0001	0000	/* ADD */
00000008	0002	0020	4020	0001	0000	0001	0000	/* ADS */
00000009	0003	0002	4020	0001	0000	0001	0000	/* IFT */
0000000A	0003	0002	2022	0001	0000	0001	0000	/* DEC */
0000000B	0003	0022	2022	0001	0000	0001	0000	/* STA */
0000000C	0003	0022	2022	0001	0000	0001	0001	/* ADS */
0000000D	0004	0003	2023	0001	0000	0001	0001	/* IFT */
0000000E	0004	0003	8405	0001	0000	0001	0001	/* DEC */
0000000F	0004	0005	8405	0001	0000	0001	0001	/* BRP */
00000010	0005	0005	8405	0001	0000	0001	0001	/* ADS */
00000011	0006	0005	8405	0001	0000	0001	0001	/* IFT */
00000012	0006	0005	4021	0001	0000	0001	0001	/* DEC */
00000013	0006	0021	4021	0001	0000	0001	0001	/* ADD */
00000014	0006	0021	4021	0002	0000	0001	0001	/* ADS */
00000015	0007	0006	4021	0002	0000	0001	0001	/* IFT */
00000016	0007	0006	2020	0002	0000	0001	0001	/* DEC */
00000017	0007	0020	2020	0002	0000	0001	0001	/* STA */
00000018	0007	0020	2020	0002	0002	0001	0001	/* ADS */
00000019	0008	0007	2020	0002	0002	0001	0001	/* IFT */
			中 略					
00000116	0008	0007	8409	B520	B520	452F	6FF1	/* DEC */
00000117	0008	0009	8409	B520	B520	452F	6FF1	/* BRP */
00000118	0008	0009	8409	B520	B520	452F	6FF1	/* ADS */
00000119	0009	0008	8409	B520	B520	452F	6FF1	/* IFT */
0000011A	0009	0008	FC00	B520	B520	452F	6FF1	/* DEC */
0000011B	0009	0000	FC00	B520	B520	452F	6FF1	/* END */

```
TERM OR RESUME ? T
SIMULATION END
```

図5-5 (b) 出力例

§6 評価・検討

評価の基準は、「並列性」を中心とする。即ち、PMの台数を増やす事によって、シミュレーション速度が向上すれば、並列性を取り出している事になる。図6－1に、前章の例における平均シミュレーション速度の実測値(注5)を示す。(○印でプロット) このグラフから、並列性を取り出してシミュレーションを行なえる事がわかる。

本章では、更に速度向上の面から、2、3の点について検討してみる。

6.1 記憶ノードによる律速

速度がPM台数について飽和するのは、CMのオーバヘッドも一因となるが、critical-pathの存在が原因となる事もある。

前章の例において、メモリを1つのモジュールとして扱った為、毎クロック大量のデータの更新を行なうので、CM及びPM内でのデータの転送時間が問題となり、このノードがcritical pathとなっている可能性がある。

前章の例では、メモリを128 word (256 Byte) としたが、これを32 word (64 Byte) に減らした場合につ

(注5) 前章の例において、途中出力を抑え、シミュレーション開始から終るまでの時間を測定した。

いてのシミュレーション速度の実測値を、図6－1に△印で示す。このグラフは、大量のデータの記憶について、システムが積極的にサポートすべきである事を示唆している。

6.2 ノードの割り付け問題

前章の例の実測値は、CMΦにノード1～5（5個）、CM1にノード6～7（2個）と、片寄った割り付けによるものである。

しかし、CM内における各ノードの処理は、シーケンシャルに行なうので、CMの台数、割り付け方等によって、オーバヘッドの寄与の仕方が異なる。そこで、CMΦにノード1～4（4個）CM1にノード5～7（3個）と割り付けた場合（メモリは32 word）についても、速度を実測した。（図6-1○印）この割り付けの変更によって、約3.5 msec/クロック 速くなる事を確認した。（CMΦの負担が、ノード1つ分軽くなった事による速度向上と考えると、この値が、1つのノードを処理する時の、CMのオーバヘッドと考える事ができる。）

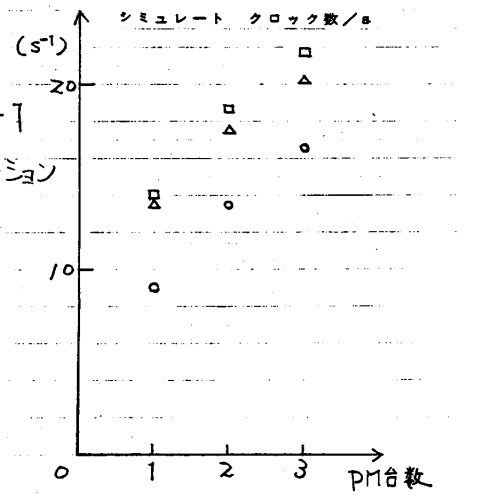


図6-1 シミュレーション速度

§7 おわりに

以上から、データフロー計算機においては、単にシステムプログラムの実装のみで、容易に、かつ論理回路の並列性を縦横に引き出した論理シミュレーションが可能である事がわかった。これは、データフロープログラムと、論理回路ネットワークとの極めて自然な対応づけによるものである。更に、システムプログラムをハードウェア化し、プロセッサ数を十分多くすれば、原理的には、対象システムの規模に依存せず、対象システム内の1つのモジュールのシミュレーション時間の最長時間のみで、対象システム全体をシミュレートする事ができる。

一方、論理シミュレーションにおいても、データフロー計算機の一般的な問題である、「記憶」の問題が、効率に大きく影響する事が明らかとなった。

今後、速度向上を目指し、より実用的な機能を備えたシステムにする為には、データフローの概念を拡張して、「記憶」等の問題を、システムが積極的にサポートする必要もあると考える。

<参考文献>

[1] 栗原、鈴木、元岡、「High Level Data Flow Machine (TOPSTAR) のシステムプログラム」信学技報EC 79-56、情処学会アーキテクチャ研究会資料 '80. 1月

[2] 元岡、鈴木、喜連川、新岡、「SAMD計算機～A High Level Data Flow Machine～」情処学会アーキテクチャ研究会資料34-1, '79、5月

[3] 坪沢、栗原、鈴木、田中、元岡、「データフローマシン"TOPSTAR"による論理回路シミュレーション」昭55情処学会全国大会講演論文集、pp 89-90