データフローマシン TOPSTAR－ＩＩによる コンカレント故障シミュレーション

中田 恒夫 ， 田中 英彦 ， 元岡 達
（ 東京大学 工学部 ）

§１．はじめに

コンカレント故障シミュレーションは、
・機能レベルのモデル化に適している
・多値論理への拡張が容易である
・タイミングに関する故障も扱える
・１パスですべての故障を扱える
といった特徴を持ち、いくつかある故障シミュレーションの方式の中で最も優れた方式であるとされている。が、反面、ひとつの結果を得るまでに相当の時間を要するという難点も持っている。これは、従来の計算機の場合、各論理要素をシーケンシャルにシミュレートすることによるところが大きい。この点については、コンカレント故障シミュレーションに限らず、通常の回路シミュレーションあるいは故障シミュレーションの他方式でも同様であるが、アルゴリズムそのものが相当複雑なコンカレント故障シミュレーションにおいてはまさに致命的であると言えよう。

元来、論理回路では回路を構成する各要素は、並列に動作しうるものが多く、又、出力は入力にのみ依存して決まる。従って論理回路とデータ駆動によるデータフロー処理は自然な対応づけが可能であり、データフローマシンでシミュレーションを行なうことにより並列度が生かされ、処理の高速化が期待される。

そこで、本研究室で開発したプロシージャレベル・データフローマシン TOPSTAR－II 上にコンカレント故障シミュレータの実装を行なったので、実装法、特徴、性能などについて報告する。

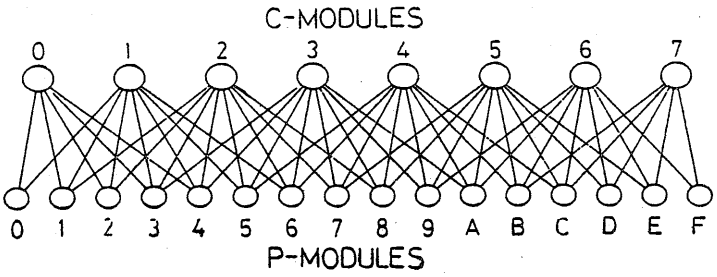


図 2.1 TOPSTAR－ＩＩのモジュール構成

§２．TOPSTAR－ＩＩのシステム構成

2.1 アーキテクチャ

TOPSTAR－ＩＩ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。

アーキテクチャの特徴としては以下の３点が挙げられる。

（１）モジュール構成

TOPSTAR－ＩＩは、CM（Communication／Control Module ）およびPM（Processing Module ）の２種類のモジュール多数から構成されている。各モジュールは、CPUとしてZ－８０を搭載し、１６KBのメモリを持つ。現在のシステムは、CM８台、PM１６台となっている。

（２）オーバラップした部分結合

CMとPMの結合は、図 2.1にあるように、部分結合となっていて、大規模システムへの拡張性を考慮している。この際、データフローネットワークへの対応を柔軟に行なうことができるよう結合をオーバラップしている。現在のところ、CM１台にPM８台、PM１台にCM４台が結合している。この結合は、リンク・ボードによって実現されている。（図 2.2参照）

（３）DMA結合

CMとPMの間のデータ転送量は、比較的多いのでDMAによりメモリ間の高速データ転送を実現している。

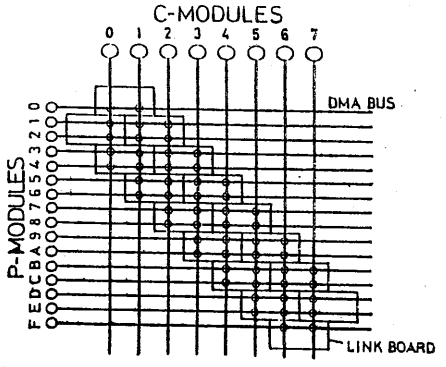
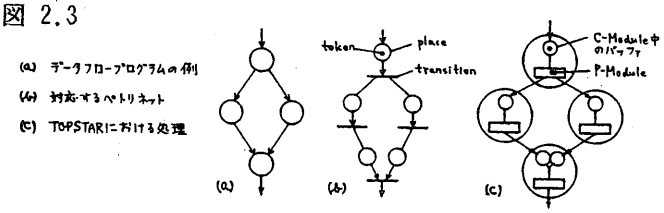


図 2.2 リンク・ボードによる CM－PM結合

2.2 制御方式

ＴＯＰＳＴＡＲ－ＩＩ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（図 2.3（ａ））である。このデータフロープログラムは図 2.3（ｂ）のペトリネットと等価である。ＴＯＰＳＴＡＲ－ＩＩではこのプログラムは図 2.3（ｃ）のように実行される。ＣＭとＰＭの役割はそれぞれペトリネットの”place ”と”transition”に対応している。

図 2.3



ＴＯＰＳＴＡＲ－ＩＩの動作の概要は以下の通りである。

各ＣＭには１個ないし複数のノードが割り当てられている。各ノードについて入力データを蓄えるバッファ（queue ）が用意されている。ＰＭは結合範囲にあるＣＭに割り込みをかけ“仕事”を求めるコマンド（ＤＥＱコマンド）を送る。ＣＭでは実行可能な“仕事”があるかどうかを調べ、あればデータとプロシージャを送り、なければその旨知らせる。ここである“仕事”が実行可能であるとは、

・プロシージャを実行するときに必要な入力データがすべて揃っている。

・結果の送り先のノードすべてについて、バッファが少なくともひとつずつは空いている。

という２つの条件がともに満たされたことを指す。

ＰＭの方は、“仕事”をもらえなかったときには次のＣＭに割り込みをかけるが、もらえたときは、まずqueue がひとつ空いたことをインセットのノードすべてに知らせる（Ｖ－op）。なお、部分結合の影響でＶ－opが届かないことが起こり得るが、あとで別のＰＭにやってもらうことにして、とりあえずそのＶ－opは省略する。

次に、プロシージャの実行を行なう。最後に、実行結果をアウトセットのノードに送り（ＥＮＱ）、一連の動作を完了する。

そしてアイドルとなったＰＭは再び“仕事”を求めてＣＭに割り込みをかける。

図 2.4は、図 2.5のようなデータフローグラフでノードＮＯの処理をＰ５が行なうときの一連の動作を図式化したものである。図中の破線はＶ－opが届かないことを意味するが、このＶ－opはあとでＰ３，Ｐ４などが行なうことになる。

ＴＯＰＳＴＡＲ－ＩＩの制御方式の特徴は次の３点である。

（１）データ駆動による分散制御

各ノードが発火可能であるかどうかは、必要とする入力データがすべて到着したかどうかに依る。すなわち、データ依存性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。また、制御は各ＣＭで独立に行なわれる。

（２）ＰＭの自由競争による負荷分散

各ＰＭが“仕事”を求めて結合範囲にあるＣＭに次々と割り込みをかけていくため、実行可能な“仕事”を多く持つＣＭに、ＰＭが自然と集まり適切な負荷分散が行なわれる。

（３）追い越しを許すパイプライン処理

各ノードのqueue の容量を複数にすることにより並列性がより良く取り出せると考えられるが、ＴＯＰＳＴＡＲ－ＩＩではＣＭとＰＭに分かれているため複数のＰＭに順次ＤＥＱを行なっても、その処理結果がＤＥＱした順に次段のノードにＥＮＱされるとは限らない。すなわち、追い越しを生じる。

データの値によって処理時間が大きく異なるような場合にはこのような追い越しを認めるようにすると並列処理性が向上すると考えられる。

この際、各データにシリアル・ナンバを施して混乱を避ける必要がある。また、queue 容量が有限であることから、追い越しを無制限に認めるとデッドロックを起こすことがあるのでqueue 内でデータの順序を揃えるような制御を行なっている。

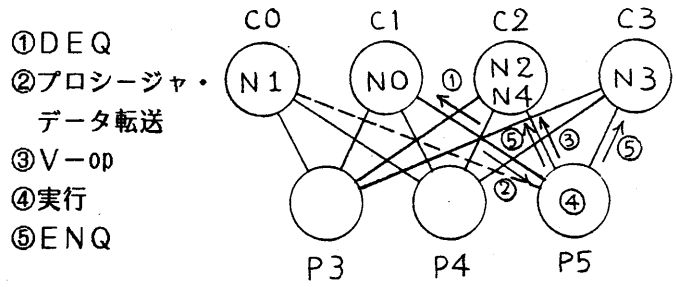


図 2.4 ＴＯＰＳＴＡＲ－ＩＩの動作

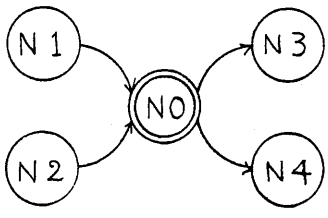


図 2.5 データフローグラフ

§３．シミュレータの構成

シミュレータは、
・システム・プログラム
・制御テーブル生成コンパイラ
・コンカレント故障シミュレーション・プログラム
により構成されている。以下でそれぞれについて簡単に説明する。

3.1　システム・プログラム

2.2で述べた制御法を実現するものであり、Ｚ－８０アセンブリ言語により記述されている。

3.2　制御テーブル生成コンパイラ

制御テーブルには、プロシージャとノードの対応関係、ノード間の接続情報が記載されており、データフロー処理を行なう際に参照される。

ユーザはテーブル記述言語をもちいてこれらを記述しコンパイラにかけることにより、機械で用いるテーブルを得る。

現段階では、接続情報の記述はデータフローグラフのレベルで行なうため、回路からデータフローグラフへの変換はユーザが行なわなければならない。但し、回路を構成する論理要素内の処理をひとつのノードに対応させているので、変換は容易である。（図 3.1参照）

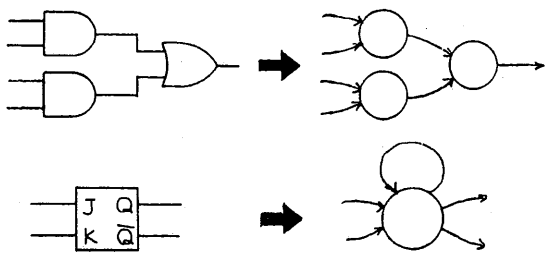


図 3.1　論理要素とノードの対応例

テーブル記述言語の仕様について簡単に説明する。
oノード名、プロシージャ名、ファイル名、ターミナル名、ラベル名……先頭がアルファベットで始まる６文字以内の英数字の並び。なお、７文字以上の場合、７文字目以降は無視される。
oターミナル長、初期値、繰り返し数、定数、クロック遅延時間……ラベル名もしくは数値
o数値は１０進と１６進が許される。１６進数は末尾にＨをつけなければならない。
oステートメントが１行に収まらないときは、行末に“，＆”をつければ何行でも継続できる。
o初期値設定の際、同じ値が繰り返されるときは、‘＊’を用いることで簡単に行なえる。
oステートメントは次の１３種が用意されている。
・ＥＱＵ文
ラベル名と値を対応づける。
・ＤＥＦＱＳＺ文
queue の容量をすべてのノードについて定める。デフォルト値は４。
・ＬＯＣＡＴＥ文
各ノードをＣＭに割り付ける。
・ＰＲＯＣ文
プロシージャが格納されているファイルの名前と、プロシージャ名を対応づける。
・ＤＥＦＰＤＡ文
ＰＭにおけるデータの先頭番地を指定する。
・ＮＯＤＥ文
ノード名とプロシージャ名の対応およびノードでの時間遅れ、セマフォのチェックの必要性を記述する。
・ＴＭＮ文
ノードの出力ターミナル名と、その長さをバイト単位で定義する。また、‘／’で囲むことにより初期値を設定することもできる。
・ＩＮＦＲＯＭ文
入力データを送ってくるノード名と、そのターミナル名を示す。
・ＯＵＴＴＯ文
出力データを送るノード名と、対応する自分のターミナル名を示す。
・ＭＡＳＫ文
ノードを実行するＰＭを特別に指定する。
・ＱＳＩＺＥ文
あるノードのみqueue の容量を変えたいときにそれを指定する。
・ＥＮＤＮＯＤＥ文
ひとつのノードの記述が終わることを示す。
・ＥＮＤ文
すべての記述が終了したことを示す。

以上で述べたテーブル記述言語で実際のテーブルを記述した例を図 3.2に示す。

（注）　このテーブルは 4.2の項で取り上げる回路（ＢＣＤ加算器）を記述したものである。

```
  1: !        FAULT SIMULATION NO.2
  2: !              BCD ADDER
  3: !
  4: DT0      EQU     0
  5: DT1      EQU     1
  6: !
  7: L1       EQU     1
  8: LVEC     EQU     8
  9: LVFL     EQU     82
 10: !
 11: DEFUSZ   4
 12: !
 13: LOCATE   C1,INPUT,ADD1,ADD2,ADD3
 14: LOCATE   C2,ADD0,BCD1,BCD2,BCD3
 15: LOCATE   C3,CD,OUTPUT
 16: !
 17: PROC     PROCIN(B:CFSIN), &
 18:          PROCOUT(B:CFSOUT), &
 19:          PROCFULL(B:CFSFULL), &
 20:          PROCHALF(B:CFSHALF), &
 21:          PROCCD(B:CFSCD), &
 22:          PROCRN(B:CFSRN)
 23: !
 24: NODE     INPUT(PROCIN),DT1
 25: TRN      CLOCK(L1) /0/ ,INVEC(LVEC),A0(LVFL), &
 26:          A1(LVFL),A2(LVFL),A3(LVFL),B0(LVFL), &
 27:          B1(LVFL),B2(LVFL),B3(LVFL)
 28: INFROM   INPUT(CLOCK)
 29: OUTTO    INPUT(CLOCK),OUTPUT(INVEC),ADD0(A0,B0), &
 30:          ADD1(A1,B1),ADD2(A2,B2),ADD3(A3,B3)
 31: ENDNODE
 32: !
 33: NODE     ADD0(PROCHALF),DT0
 34: TRN      S0(LVFL),C0(LVFL)
 35: INFROM   INPUT(A0,B0)
 36: OUTTO    OUTPUT(S0),ADD1(C0)
 37: ENDNODE
 38: !
 39: NODE     ADD1(PROCFULL),DT0
 40: TRN      S1(LVFL),C1(LVFL)
 41: INFROM   INPUT(A1,B1),ADD0(C0)
 42: OUTTO    BCD1(S1),CD(S1),ADD2(C1)
 43: ENDNODE
 44: !
 45: NODE     ADD2(PROCFULL),DT0
 46: TRN      S2(LVFL),C2(LVFL)
 47: INFROM   INPUT(A2,B2),ADD1(C1)
 48: OUTTO    BCD2(S2),CD(S2),ADD3(C2)
 49: ENDNODE
 50: !
 51: NODE     ADD3(PROCFULL),DT0
 52: TRN      S3(LVFL),C3(LVFL)
 53: INFROM   INPUT(A3,B3),ADD2(C2)
 54: OUTTO    BCD3(S3),CD(S3,C3)
 55: ENDNODE
 56: !
 57: NODE     BCD1(PROCHALF),DT0
 58: TRN      D1(LVFL),CD1(LVFL)
 59: INFROM   ADD1(S1),CD(CU)
 60: OUTTO    OUTPUT(D1),BCD2(CD1)
 61: ENDNODE
 62: !
 63: NODE     BCD2(PROCFULL),DT0
 64: TRN      D2(LVFL),CD2(LVFL)
 65: INFROM   ADD2(S2),CD(CU),BCD1(CD1)
 66: OUTTO    OUTPUT(D2),BCD3(CD2)
 67: ENDNODE
 68: !
 69: NODE     BCD3(PROCHALF),DT0
 70: TRN      D3(LVFL)
 71: INFROM   ADD3(S3),BCD2(CD2)
 72: OUTTO    OUTPUT(D3)
 73: ENDNODE
 74: !
 75: NODE     CD(PROCCD),DT0
 76: TRN      CU(LVFL)
 77: INFROM   ADD1(S1),ADD2(S2),ADD3(S3,C3)
 78: OUTTO    OUTPUT(CU),BCD1(CU),BCD2(CU)
 79: ENDNODE
 80: !
 81: NODE     OUTPUT(PROCOUT),DT0,NOSEM
 82: INFROM   INPUT(INVEC),ADD0(S0),BCD1(D1), &
 83:          BCD2(D2),BCD3(D3),CD(CU)
 84: MASK     P9
 85: ENDNODE
 86: !
 87: END
```

図 3.2　テーブルの記述例

3.3　コンカレント故障シミュレーション・プログラム

ひとつの論理要素について行なうべき処理を記述したプログラムで、C言語で書かれている。

処理の主要部はすべての論理要素について共通であるため、ユーザは論理関数をC言語で記述し本プログラムとリンクさせて、ノードに対応するプロシージャを作り出すようにしている。

なお、メモリ容量の制約からシミュレータの機能を以下のように定めている。

- 論理値は、（0，1）の2値
- 単一縮退故障を対象とする。
- 原則として、機能レベルのシミュレーションとする。
- 遅延の問題は扱わない。

3.4　シミュレーション手順

ある論理回路をシミュレータにかけて出力を得るまでの手順をまとめると、図 3.3のようになる。

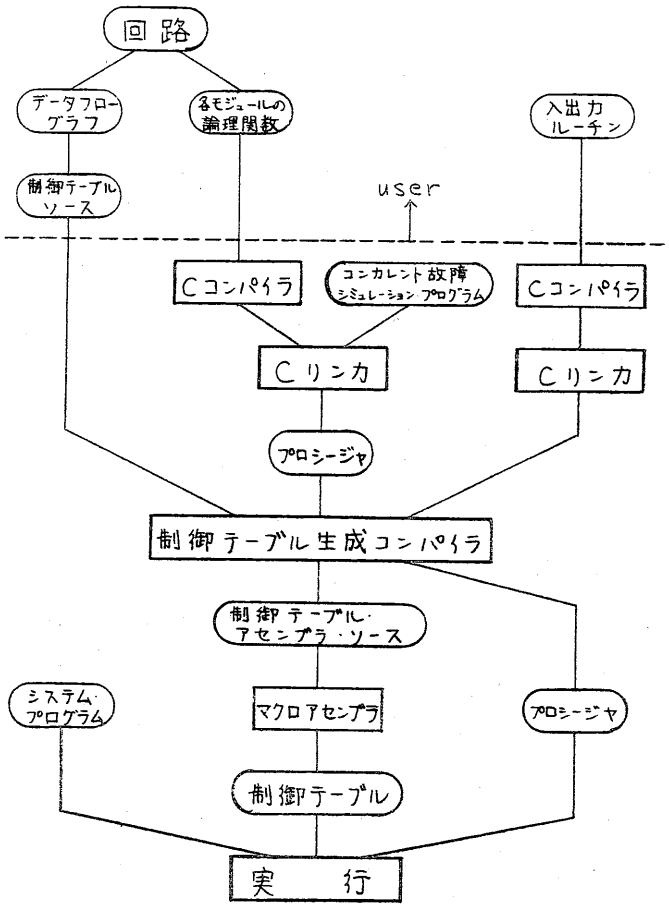


図 3.3　シミュレーションの手順

§4．シミュレーション例

実際の回路にシミュレーションを行なった例を2種類挙げる。

4.1　10進同期カウンタ

順序回路に応用した例である。

回路図を図 4.1、データフローグラフを図 4.2に示す。図 4.2のINPUT，OUTPUTの2種のノードはそれぞれ入力データの生成、結果の印刷のために設けた擬似ノードである。

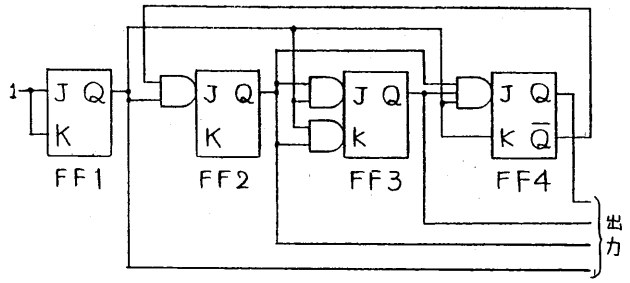


図 4.1　10進同期カウンタ

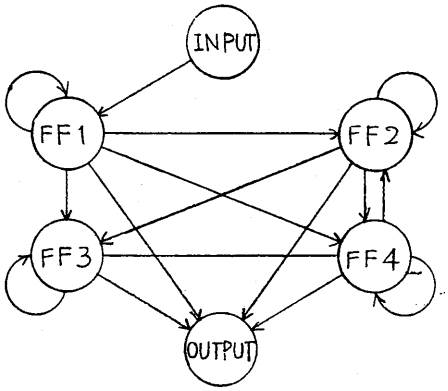


図 4.2　データフローグラフ

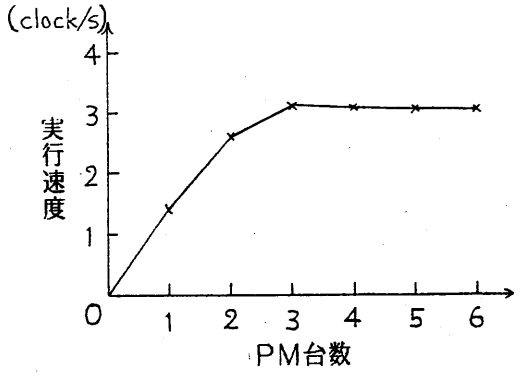


図 4.3　PM台数と実行速度の関係

この回路では入力データが全く変化しないため、各フリップフロップに初期値0を与え、クロックを進めた場合を想定してシミュレーションを行なった。

その際、queue 容量を2としてPM台数を1台から6台まで変化させたときの実行速度を測定した。結果を図 4.3に示す。また、シミュレータの出力結果を図 4.4に示す。出力は、クロックの値、各出力端子における論理値とその値を変えるような故障のリストから成る。

```
******** CONCURRENT FAULT SIMULATION ********

CLOCK : 0
LINE NUMBER : 0
VALUE : 0
FAULT LIST :     MD      IOS      PLACE    STKAT
                 1        0         0        1
LINE NUMBER : 1
VALUE : 0
FAULT LIST :     MD      IOS      PLACE    STKAT
                 2        0         0        1
LINE NUMBER : 2
VALUE : 0
FAULT LIST :     MD      IOS      PLACE    STKAT
                 3        0         0        1
LINE NUMBER : 3
VALUE : 0
FAULT LIST :     MD      IOS      PLACE    STKAT
                 4        0         0        1
CLOCK : 1
LINE NUMBER : 0
VALUE : 1
FAULT LIST :     MD      IOS      PLACE    STKAT
                 1        I         0        0
                 1        0         0        0
                 1        S         0        1
LINE NUMBER : 1
VALUE : 0
FAULT LIST :     MD      IOS      PLACE    STKAT
                 2        0         0        1
LINE NUMBER : 2
                （中　略）
                 4        S         0        0
                 3        I         0        1
                 2        I         0        0
                 2        I         1        0
                 3        I         1        1
                 3        I         0        0
                 3        I         1        0
CLOCK : 50
LINE NUMBER : 0
VALUE : 0
FAULT LIST :     MD      IOS      PLACE    STKAT
                 1        0         0        1
                 1        S         0        0
LINE NUMBER : 1
VALUE : 0
FAULT LIST :     MD      IOS      PLACE    STKAT
                 2        I         1        1
                 2        0         0        1
                 4        I         0        1
                 4        I         0        0
                 4        I         1        1
                 4        I         1        0
                 4        I         2        1
                 4        I         2        0
                 3        I         0        1
                 3        I         0        0
                 3        I         1        0
LINE NUMBER : 2
VALUE : 0
FAULT LIST :     MD      IOS      PLACE    STKAT
                 3        I         1        1
                 3        0         0        1
                 3        S         0        1
                 4        S         0        1
                 2        0         0        1
LINE NUMBER : 3
VALUE : 0
FAULT LIST :     MD      IOS      PLACE    STKAT
                 4        0         0        1
                 1        I         0        0
                 1        S         0        1
******** SIMULATION END ********
```

図 4.4　シミュレータの出力

4.2 ＢＣＤ加算器

組合せ回路に応用した例である。

回路図を図 4.5、データフローグラフを図 4.6に示す。

こちらの例ではクロックがないため、入力として（Ａ，Ｂ）に（０，０）から（９，９）まで１００通りのデータを加えてシミュレーションを行なう。

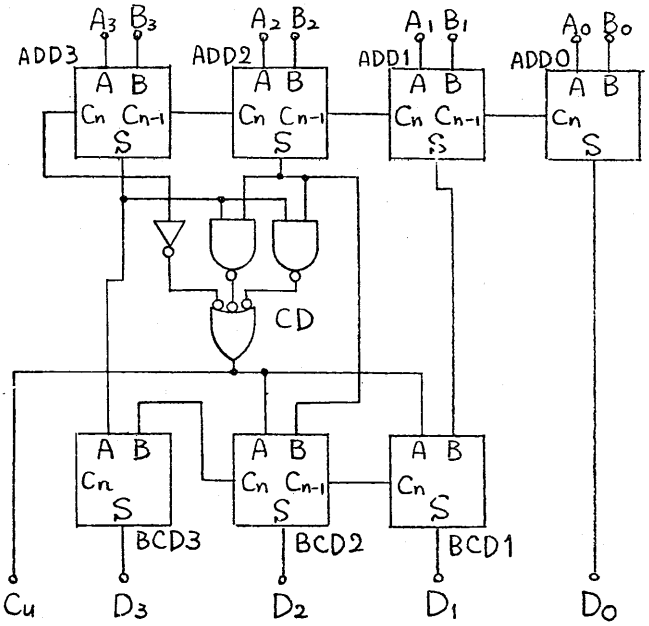


図 4.5 ＢＣＤ加算器

これらのデータはＩＮＰＵＴノードで次々と生成される。

また、CM台数、queue 容量をパラメータとして、PM台数と実行速度の関係を測定した。その結果を図 4.7に示す。なお、CM台数をふやすと部分結合の影響でデータが届かないことが起こるので、適宜リレー・ノードを挿入している。

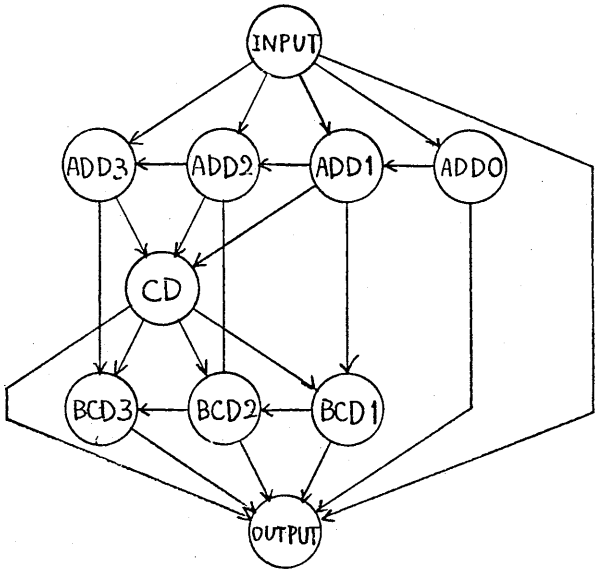
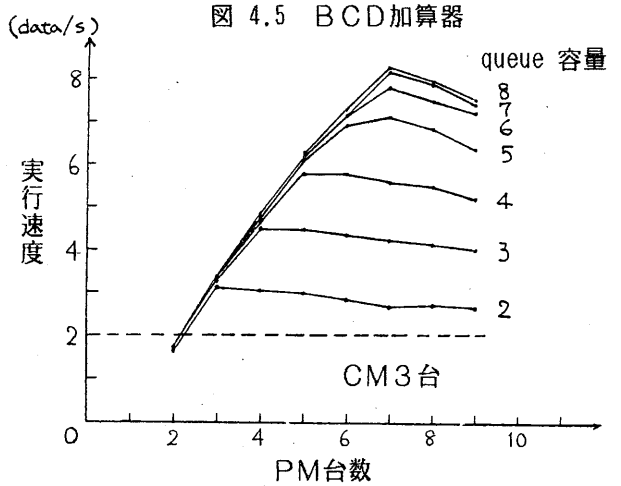
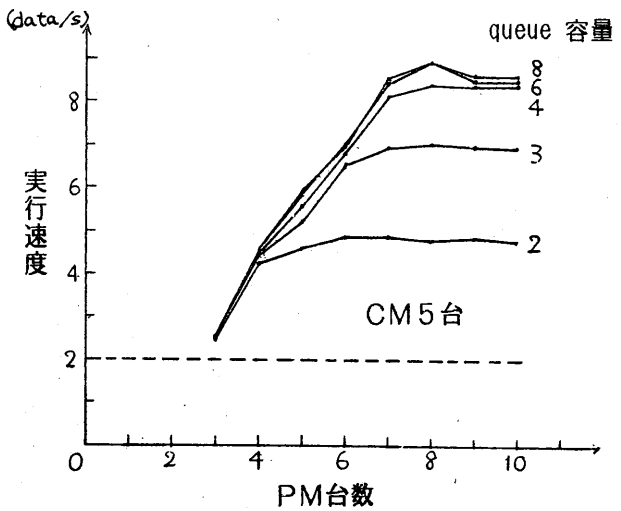


図 4.6 データフローグラフ





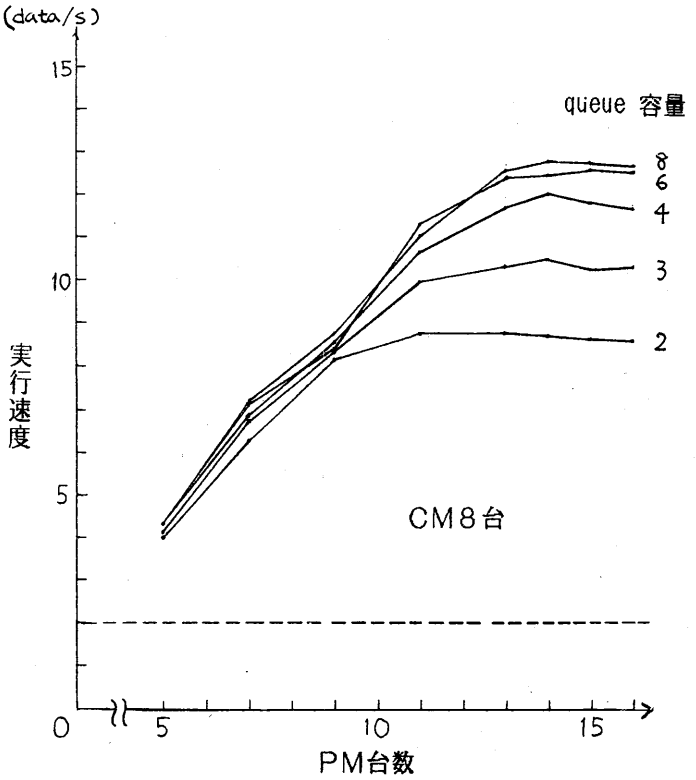


図 4.7 PM台数と実行速度の関係

§５．評価・検討

本章では、４章で述べた２つの例について実行速度の面から性能評価および検討を行なう。

5.1　ノイマン型計算機との比較

ノイマン型計算機上での実装を行なっていないため直接比較はできないが、ＴＯＰＳＴＡＲ－ＩＩの各PMにおいて実行されたプロシージャの実行時間の総和をもってノイマン型計算機の実行時間と考える。これには、各種のテーブルをひく時間などは含まれていないため、真の実行時間はこの“仮想実行時間”よりも大きいと考えられる。

ＢＣＤ加算器の例では、仮想実行時間は１００データについて５０．１秒であり、実行速度に直すと１．９９データ／秒となる。

ＴＯＰＳＴＡＲ－ＩＩで実行した場合、実行速度は、１．６５～１４．１データ／秒であり、最悪でも同程度、非常に良い場合は７倍以上の速度で実行できることがわかる。

5.2　速度飽和の原因

PM台数と実行速度の関係を見ると、PM台数が少ないうちはほぼ台数に比例して速度が上がっていくが、あるところで飽和してしまう。queue 容量をふやすと速度の伸び具合は良くなるが、いずれ飽和に至ることには変わりない（図 5.1）。この原因は、主に次の２点にある。

（１）システム・プログラムのオーバヘッド

テーブルのサーチ、データ転送などに要する時間があるため、速度がある値以上にならない。

４章で挙げた例では、プロシージャの実行時間は平均５０ms程度でありオーバヘッド時間はそれほど問題にならないが、一般のゲートレベル論理シミュレーションを行なう場合には実行時間が平均数msと小さいため影響が出てくる。

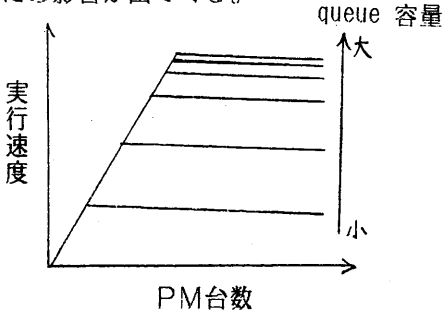


図 5.1　PM台数と実行速度の関係

（２）仕事の不足

対象となる回路自体の並列性が低いような場合には、実行可能なプロシージャの数自体が少なく、PM台数をふやしても大半が遊んでいることになり速度は向上しない。組合せ回路では自分自身に戻るアークを持つノードがないので、queue 容量をふやせば実行可能なプロシージャがふえ速度も向上していく。これに対し、順序回路では、各プロシージャが実行可能になるには、ひとつ前のクロックの処理が終了する必要があるためqueue 容量をふやしても速度が伸びないという問題がある。１０進カウンタの例での飽和が早いのもこのためである。

5.3　速度低下の原因

速度が飽和した状態でPM台数をふやすと、むしろ速度が低下してしまっている。前項で述べたように飽和状態ではPMがほとんど遊んでいるわけであるが、遊んでいるPMも仕事を求めてCMに割り込みをかけるために、全体としての処理の進行を妨げてしまうことが原因であると考えられる。

§６．結論

データフローマシンを用いることにより、コンカレント故障シミュレーションを高速に実行できることがわかった。回路の並列性を容易に引き出すことができたのは回路とデータフローグラフの類似性によるものである。また、回路自体の並列性が低くても多数の入力データを考えれば見かけの並列性が高まり、全体として高速化がなされる。

割り付け方により、同じプログラムでも実行速度が大きく変化することも判明した。回路の特性を生かし、高速に処理できるような割り付け方を確立し自動化することが今後の課題となろう。

《　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”
情報処理学会アーキテクチャ研究会資料　３４－１
１９８０年１月

②深沢、栗原、鈴木、田中、元岡、
“データフロー計算機による論理シミュレータ”
情報処理学会電子装置設計技術研究会資料　６－３
１９８０年１０月