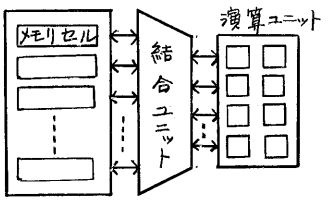
データフロー計算機における故障検出の一方式

浅田邦博　グェン・ニット　坂浩一　斉藤忠夫　猪瀬博
（東京大学　工学部）

1 はじめに

データフロー計算機（以下DFC）は、フローチャートプログラムとは双対的関係のアルゴリズムの1記述法であるデータフロープログラム（以下DFP）[1][2]を直接実行するものである。これは従来のフォン・ノイマン形計算機と異なり、プログラムを構成する各種の演算子がオペランドデータが整い次第、非同期に実行されるという、データ駆動による制御を特徴としており、本質的な並列処理性を備えている。単位となる演算子のレベルには、四則演算から手続き、タスクレベルまで種々のものが有り得るが、これまで提案されてきたいくつかのDFC[3][4]では、主にこの並列処理性を最大限に生かして処理能力を向上させることに主眼がおかれてきたのも当然と言える。従って、これらの計算機では一般に、複数の演算ユニットと、複数のメモリセル、及び、その両者を結合するための多重バッファ機能を有する強力な結合ユニットから構成されている。（第1図）従来の計算機の制御回路に相当するものはこの結合ユニットに分散化されて置かれており、これにより、演算の並列化が実現されている。このことは又DFCは自然な形の冗長系を成していることを意味し、若干の故障検出と、故障ユニット分離機能等を付加することにより、自律的故障診断、修復機能をもつ系が構成できることが期待される。本研究はこのような立場から、DFCの特徴を生かした、故障診断・修復の一方式について検討したものである。以下、第2章では、故障診断の手法、及び、それに適するハードウェア構成の一方式として、分散形DFCを提案し、第3章では、シミュレーションにより、本方式の評価を行なった結果について述べている。

第1図 DFCの構成要素



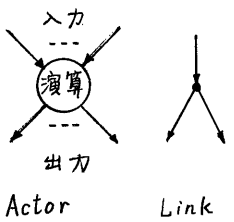
2 自律的故障診断方式と分散形DFCの構成

2-1 準備

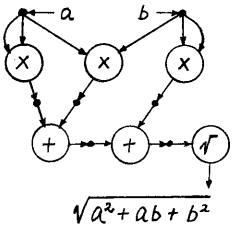
診断方式について述べる前に、前提となるDFCの機械語に相当するDFP言語について簡単に眺める。これは、J. B. Dennis等[1][2]の提案に従う、有向グラフ表現のものである。第2図に示すように、プログラムは、演算子であるActorとデータ伝搬作用をもつLinkの2要素から構成される。第3図の例に示すように、DFPはActorとLinkとをノードとしてもつ有向グラフで表現されるが、プログラムの実行はデータ駆動の原則に基づき、各Actor，Linkが非同期的に動作（fire）することによって進行する。各Actor，Linkが動作する条件は、第4図に示すように、基本的に入力データがすべて到着し、出力が空であることである。このデータの到着を表わすために第4図のように黒丸（トークン）を用いる。各Actor，Linkの動作は(i)入力トークンの吸収、(ii)内部処理遅延，(iii)トークンの出力　の過

様としてとらえられる。(第4図) DFPでは、従来の計算機における計算状況を示すプログラムステータスワード(PSW)に相当するものは、プログラム中のデータの所在を示すトークンマップである。なお、第3図の例にあるようなDFPを直接実行するDFCでは、各Actorの演算を実行する複数の演算ユニットと、プログラムやトークンを格納するメモリセル、プログラムの中のLink情報に基づきトークン(データ)を演算ユニットとメモリセル間でやりとりする結合ユニットから構成されるが、一般には、Actorの数だけ演算ユニットが用意されてはおらず、何らかの競合制御機構を用いて演算ユニットを多重使用するのが普通である。すなわち、DFP中のActorと実際の演算ユニットとは、1対1対応はしていない。

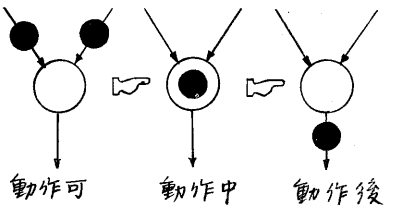
第2図 DFPの構成要素



第3図 DFPの例



第4図 Actor, Linkの動作過程



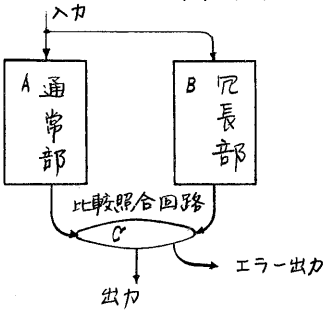
2-2 自律的故障検出の概要

計算機システムにおいて実時間の故障検出機能を達成するために従来用いられてきた手法の中には、

i) 第5図に示すように冗長ハードウェアを付加し両者(A, B)の出力を比較検査する。

ii) ソフトウェア, ファームウェアの診断プログラムを通常プログラムと時分割に並行して実行し故障診断を行なう。

第5図 ハードウェア冗長系



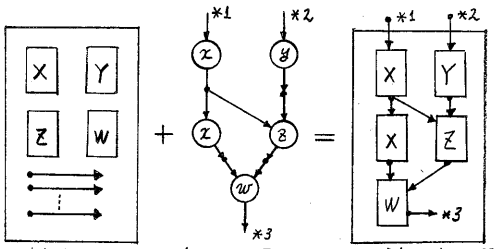
等がある。(i)の場合、冗長部(B)は(A)と同一のものを用いる(2重化)こともあり何らかの縮退したものを用いること(パリティチェック等)もあるが、(ii)に比較して故障検出の時間遅れも少なく、間欠故障に対しても有効であるのが利点であるが、ハードウェアコストの増加するのが欠点である。DFCにおいても、この(i)(ii)の手法を用いることは可能であるが[7]はじめに述べたようにDFCではプログラムのもつ並列性を最大限に生かすため複数の演算、メモリー, 結合ユニットから構成されそれ自身冗長構成となっていると見ることができる。もちろん, はじめからあるユニットが他のユニットの冗長部として定まっているのではなく、それぞれは独立非同期的に動作できるのが第5図とは異なり、又, 同図の比較部(C)に相当するものも表面的には存在しない。しかし、第6図に模式的に表わされるように、DFC(a)は与えられたプログラム(b)により、(b)と同形の論理的ハードウェア構成(c)をとるものであると見ることができる。すなわち、DFCは、プログラムによりハードウェア構成を適応変化できる機能をもつと考えることができる。第5図に示すような冗長系をハードウェアで固定的に用意するかわりに、DFCではプログラ

ムにActorレベルでそれと等価な冗長部を導入することにより同様の効果が期待できることになる。ここで注意すべき点は、2-1節に述べたように、プログラムレベルの各Actor, Linkは、ハードウェアレベルの各要素に1対1に対応してはいないため、第6図(c)の論理的計算機を構成する各要素は、(a)の物理的計算機の各要素を時分割多重使用していることである。このため、(c)の計算機では故障は見かけ上多重故障として現われることになり、第5図のような冗長構成になっていても故障が検出できないおそれがある。このことの極端な例は、従来の単一プロセッサでDFPをシミュレートした場合であり、この場合にはプログラムに冗長部を導入しても(開欠故障の特殊な場合を除き)故障検出能力は期待できない。この問題を解決するには、第6図(c)の論理計算機の中で通常部と冗長部に同時に同じ故障が生じないように(a)の要素を割当てればよい。これは見方をかえれば、(b)のプログラムを構成する各Actor, Linkへ(a)の計算機エレメントを割当てる問題となっている。

本論文で提案するDFCの自律的故障診断の方式は、上述の考え方に基づくもので、次の2つの部分からできている。

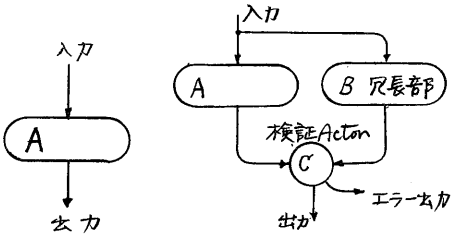
第1段階：与えられたDFPから、自律診断機能をもつ冗長プログラムへの変換
第2段階：冗長プログラムを構成する各Actor, Linkのハードウェア要素への割付け

第6図 DFCの構造可変性説明図



(a)物理的DFC (b) DFP (c)論理的DFC

第7図 DFPの冗長構成



(a)原DFP (非冗長) (b) 冗長化DFP

□ --- 演算ユニット ⟶ --- メモリセル+結合ユニット

2-3 冗長プログラムの作成

第5図のハードウェア冗長系からの類推より、第7図に示す変換により冗長プログラムを作成することができる。Actor Aは原プログラムを表わし、Bはそれに対する冗長Actorであり、Cは比較検査を行なう検証Actorである。第7図の変換は

i) マクロな冗長プログラム：原プログラムを1個のマクロなActorと見て、変換を行なう。
ii) ミクロな冗長プログラム：原プログラムを構成する各Actor毎に変換を行なう。

の2つの手法が考えられ、又、プログラムの階層に基づき階層的に行なうこともできる。マクロ冗長プログラムの利点は、検証Actor Cが少なくて済むことであり、欠点は故障検出の分解能が低いことであり、ミクロ冗長プログラムではその逆が言える。冗長部Bについてはハードウェアのときと同様、Aと同じActorを用いる(二重化)ことも、縮退したActorを用いることもできることに注意された

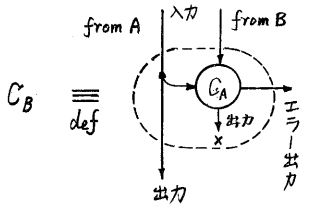
い。検証Actor Cには次の2種のものが考えられる。

(α) 無記憶検証Actor (C_A): 2つの入力がともに到着したとき検証動作を開始し、エラーが発見されればエラー出力へエラートークンを発生し、正常なら通常出力へ通常Actorからの入力トークンをコピーし出力する。(これを用いた冗長プログラムを同期検証モードプログラムと呼ぶ。)

(β) 記憶検証Actor (C_B): (α)の無記憶検証Actor C_AとLinkを用いて、第8図のように構成できるものである。(これを用いた冗長プログラムを非同期検証モードプログラムと呼ぶ。)

無記憶検証Actorを用いたプログラムでは、検証が完了するまでそれ以降のActorは実行されないのに対し、記憶検証Actorを用いた場合には、検証とそれ以降のActorの実行とは並行して行なわれるため、スループットの向上が期待される反面、故障の検出が、計算機出力に対し遅れる欠点をもつため、計算機出力に特に高い信頼性が要求される応用には向かない。しかし、従来、2-2節のii)が用いられてきた分野には適用できるものである。この記憶検証Actorは、第3章に示すように、冗長Actorに対し通常Actorを高い優先度で実行する手法とともに用いるのに適している。

第8図 記憶検証Actor C_B



2-4 冗長プログラムのハードウェアへの割付け

前節で示した手法で作成された冗長プログラムを、ある与えられた日下Cで実行する場合、検出されない故障が最小になるよう実行する必要がある。第7図に従って検出されない故障を挙げると

(i) 通常Actor, Linkと冗長Actor, Linkにおける同一種の多重故障
(両者とも同様に故障すると検証Actorで検出されない)

(ii) 検証Actorの故障(その中でもエラートークンを発生しない方への故障)

の2種が考えられる。(i)のActor, Linkの多重故障とは、2-2節に述べたように、実際には同一のハードウェア要素が割当てられた時におけるその要素の単一故障に起因するものがほとんどで純粋にハードウェア的多重故障は無視してよい。従って本方式で検出されない故障の発生率Mは、次式で評価できる。

$$M \approx \underbrace{\sum_k A_{m(c_k)}}_{\text{(検証Actorの故障率)}} + \underbrace{\sum_j L_{m(\ell_j)}\,\delta(m(\ell_j)-m(\ell'_j))}_{\text{(同一ハードウェア要素に割当てられたLinkの故障率)}} + \underbrace{\sum_i A_{m(a_i)}\,\delta(m(a_i)-m(a'_i))}_{\text{(同一ハードウェア要素に割当てられたActorの故障率)}} \qquad (1)$$

ただし
- Σ --- 確率測度の和(同一の A_α, L_αについては重複を許さない)
- $m(\bullet)$ --- Actor, Linkから、ハードウェア要素への関数(割付関数)
- A_α --- α番目のActor実行ハードウェア要素(主に演算ユニット)の故障率
- L_α --- α番目のLink実行ハードウェア要素(主に結合メモリセル)の故障率
- $\delta(i)$ --- ($i=0 \to 1$ else $\to 0$)の関数
- ℓ_j, ℓ'_j --- 相互に対応する通常Link, 冗長Link
- a_i, a'_i --- 相互に対応する通常Actor, 冗長Actor

計算機が与えられたとき、冗長プログラムの各Actor, Linkから、それが実行されるハードウェア要素への写像$m(\cdot)$を計算機の構造が許す範囲で、(1)式を最小とするように定めるのが、本節のテーマである割付問題であるが、(1)式から容易に分かるように、通常部と冗長部の対応するActor, Linkを、それぞれ別のハードウェア要素へ写像するのが、最良の方法である。しかし、一般のDFC（Ex. MIT形(第9図)）においては、Actor, Linkの実行は、自由競争の原理に基づき、空いているハードウェアが行なえるようになっているため、写像$m(\cdot)$は、決定論的には定まらず、確率の形をとる。（従って(1)式の評価も期待値をとる必要がある。）このため、本方式を適用するに当たっては

条件：対応する2種（冗長，通常）のActor, Linkを同一のハードウェア要素で実行しない。(2)

手段を有するDFCを用いることが好ましい。これについては、2-6節で述べる。

本方式に従う自律的故障診断（検出）方式の利点の主なものは、

(a) 第5図のようなハードウェアにおける固定的冗長部を設ける必要がなく系の拡張性，柔軟性に富む。

(b) プログラムの冗長部，通常部の対応している各部を、同一ハードウェア要素で実行しない条件の下で、DFCのハードウェア要素を時分割多重使用できるため、同一ハードウェア量で第5図に比較し高性能（スループット等）となる。

(c) 記憶検証Actorを用い、プログラムの冗長部，通常部の実行優先度に差をつけることにより、原プログラム（非冗長）と同程度のスループットをもち、故障検出能力も合わせもつ系が実現できる。

(d) 故障検出のハードコア的検証Actorも他のActor同様、多重使用されるため、相対的に系の故障検出能力が向上する。

が挙げられる。それに対し欠点は、

(e) 原プログラムから、冗長プログラムを作成する過程が必要である。

(f) DFCに(2)の条件を満足させる必要がある。

(g) 第5図の比較照合部に相当する検証Actorは、それ自身1個のActorとして実行されるため、比較的複雑であり、故障率も高い。（これは利点(d)と相殺関係にあるが、どちらが大きな効力をもつかは、系の構成や、プログラムに依存しよう。）

である。

2-5 故障からの復旧について

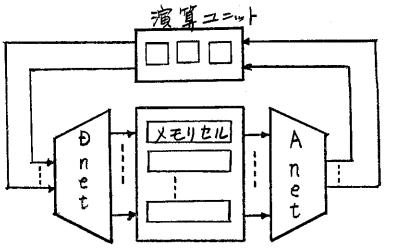
2-2～2-4節に述べた手法により得られる故障情報（i.e. 検証Actorの発生するエラートークン）に基づき、故障ハードウェアの識別を行ない、故障のモード（恒久故障，間欠故障）を判別し、恒久故障の際には、そのハードウェア要素を用いないよう系を再構成する復旧動作を行なうために、本研究では、DFCの外部に従来形式の保守計算機を想定している。現時点においては、DFCで実行すべきプログラムのコンパイル、初期ロード等のために、ミニコン程度のホスト計算機をおくのが普通に用いられる手法であり、保守計算機には、このホスト計算機を用いればよい。故障情報であるエラートークンはすべてこの保守計算機へ集められ処理される。このような独立した保守機能を別に想定する必要は、本方式が、第5図のハードウェア冗長構成をモデルとしていることによる。第5図の構成は故障検

出能力を有するが、復旧能力は持たない。周知のように3重化等のより冗長な構成と、多数決論理に基づく比較照合部を用いることにより、復旧能力を持つ系を構成できるが、本方式を、同様に拡張し、検証Actorに多数決論理機能を組込むことで、自律的復旧機能をもつDFDが構成できることは明らかであろう。(この場合はすべて無記憶検証Actorを用いる。)

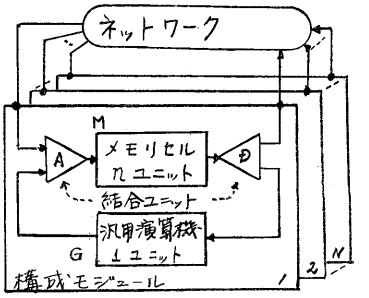
2-6 自律的故障診断向き分散形**DFD

上述の故障診断方式を用いる時、DFDに要求される付加的機能は、2-4節の(2)に述べたものである。第9図はMITで提案されているエレメンタリ形の概念図であるが、このような集中形*計算機では一般に、各Actorは自由競争の原則により任意の演算ユニットで実行されるため、(2)の条件は満足されない。又、故障からの復旧における故障ハードウェアの分離に関しても、集中形は適さないと考えられる。そこで、条件(2)を積極的に実現できる分散形計算機(第10図)について述べる。第10図に示すように、この計算機は複数の同一モジュールがネットワークで結合された、ネットワーク計算機の一種である。各モジュールは、複数のメモリセル(M)と単一の汎用演算ユニット(G)、及び、メモリセルと汎用演算ユニット間と、メモリセルとネットワーク(A, D)の結合ユニットとから構成される。ネットワークはモジュール間のデータ転送を行なう簡単なもので、故障モジュールを系から分離する機能をもっている。又、ネットワークを介し相互に接続されている各モジュールは電気的に絶縁(光結合)されており、外部からの電気ショック等による故障がモジュールをこえて拡大するのを防止している。DFPは、各モジュールのメモリセルへ分割して割付けられ、その汎用演算ユニットでのみ処理される。ネットワーク上にはデータトークンだけが流れ1つのモジュールのメモリセルに格納されているActorの処理を他のモジュールで実行することはしない。従って、2-3節で述べた冗長プログラムの対応する通常部Actorと冗長部Actorをそれぞれ別のモジュールへ割付けることにより、2-4節の条件(2)を満たすことができる。

第9図 MIT形DFC(エレメンタリ形)



第10図 分散形DFCモデル

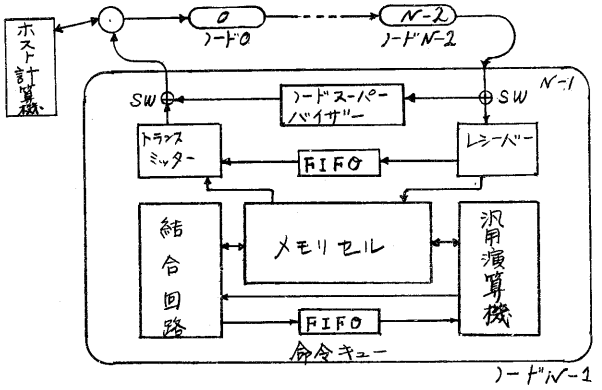


第10図の分散形DFCは、上述のように本論文で提案した自律的故障診断に有利なように構成されているが、反面、第9図のような集合形計算機では生じない新たな問題、「DFPの各モジュールへの効率的割付問題」が生ずる。割付が不適当な場合には処理効率の著しい低下が生ずることは明白であり、これについては次章で詳しく述べている。

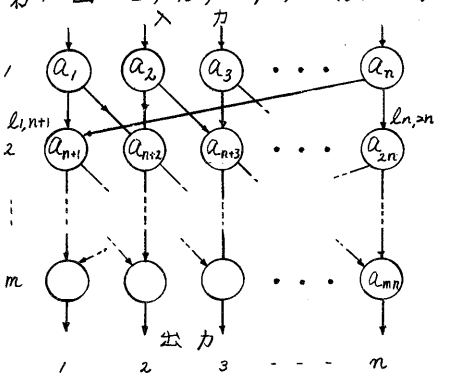
注 **, * : 本論では、すべてのメモリセル(Actor)が任意の演算ユニットで直接実行できるものを集中形と呼び、その他を分散形と呼んでいる。制御はDFCでは、いずれもデータ駆動であり、分散形となっている。

第11図は、第10図のモデルの1つの実現例として、現在作成中のもので、ネットワークとして、DLCN形ループネットワーク(8Mbps)[5]を用い、汎用演算ユニットとしてマイクロプロセッサレベルのものを用いている。これは数百～数千ステップ程度のActor処理時間を想定した、中レベルのDFCであり、本方式に従う、自律的故障検出能力の実現性を実証するとともに、処理効率の測定の目的をもっている。

第11図 ループ結合分散形DFC構成図



第12図 モデルプログラム(DFnm)



3. シミュレーションによる評価

3-1 シミュレーションの目的と方法

シミュレーションの主目的は、2-3節で述べた冗長プログラムが、非冗長プログラムに対し、処理速度の点で、低下する度合を評価することである。評価対象とした計算機は第11図のものであり、モデルプログラムは、第12図に示す構造のものである。このモデルプログラム(DFnm)は、見かけの並列度がnの部分プログラムがmレベル直列につながったものであるが、実際のプログラムでは並列度nは、レベルとともに変化し、以下に示す非同期冗長プログラムのシミュレーションでは、演算ユニットの空きが生じやすくより有利となる。従ってDFnmは安全側の結果を与えると考えられる。

第12図の各Actor a_iの処理に要する時間をe_iで表わす。第11図のループのノードであるモジュールを0～N-1とループの順方向に番号をつける。このとき、各Actorの各ノードへの割付関数を$m(\cdot)$とする。

$$\text{Actor割付関数: } m(a_i) \stackrel{def}{=} \{\text{Actor } a_i \text{の割付けられたノード番号 } 0 \sim N-1\} \quad (3)$$

第12図のLink, l_{ij} (Actor $a_i \rightarrow$ Actor a_j)をトークンが伝搬する遅延時間をd_{ij}とすると、これは、Actor a_iとa_jが、同一ノード内にあるときと、異なるノードにあるときとで、異なってくる。これを次式で近似的に取扱う。

$$\text{リンク遅延時間: } d_{ij} = D_0, \ (m(a_i) = m(a_j)) \text{のとき}$$
$$= D_1 \times ((m(a_j) - m(a_i)) \bmod N) + D_2, \ (\text{その他}) \quad (4)$$

(4)式の意味は、リンク遅延時間が、ノード内のとき固定D₀で、ノード間のときはループ上のノード間距離(中継ノード数)の一次式で定まるとしたものである。第11図の系で用いたDLCN形ループでは、ループ上の遅延は、メッセージ長とトラヒック量に関係することが知られている[6]が、平均的には(4)式で表わされると考えられる。

第11図の各ノードでは、$m(\cdot)$に従って割当てられたActorを実行条件の成立したものから順に(複数あるときは添字の小さいものから)実行していくとする。第12図の全入力に同時に入力トークンが与えられてから、すべての出力にトークンが出そろうまでに要する時間をプログラムのスループット時間Tと定義するとき、

$$\text{等価多重度}: U \overset{\text{def}}{\equiv} \left\{\sum_i e_i\right\} \Big/ T \qquad (5)$$

で定義される等価多重度Uをスループットの評価量に用いる。(5)式の分子は全Actorの処理時間の和であり、従来の単一プロセッサにおける処理時間である。従ってUは従来の単一プロセッサの何倍の速度であるかを示す指標であると言える。

3-2 割付関数$m(\cdot)$の決定

第12図のプログラムが与えられ、各Actorの処理時間e_i, リンク遅延時間を定める(4)式が与えられても、(5)式のUは、割付関数$m(\cdot)$に依存する。割付の種類はループの対称性を考慮しても、N^{A-1}(Nはノード数, AはActor数)となり、最適割付関数$m_{ideal}(\cdot)$をしらみつぶし法で求めることは、Actor数が大きくなるに従い指数的に困難となる。従って何らかの効率的アルゴリズムが必要となるが、ここでは、次のヒューリスティックアルゴリズムを用いた。

割付関数決定アルゴリズム: 与えられたプログラムより、実行可能なActorを順に(複数あるものは、添字の小のものから)1つとり出し、それを後りにすべてのノードで実行してみる。その結果、その段階での部分的スループット時間が最小となるノードへそのActorを割付ける。これをすべてのActorについて終了するまでくりかえす。 (6)

このアルゴリズムに従って求めた割付関数を$m_{practical}(\cdot)$と表わす。

3-3 冗長プログラムの作成

シミュレーションで用いた冗長プログラムは、第12図の原プログラムを基に、2-3節で述べた方法に従い作成した記憶検証Actorを用いたマクロ冗長プログラムである。ただし簡単化のため検証Actorへの Link遅延, 及び検証Actorの処理時間は無視した。この冗長プログラムは非同期検証モードであるから、プログラム出力と、検証出力との間には時間差がある。プログラム出力のスループットに基づく等価多重度を、U_p, 検証出力のをU_vと表わす。又、冗長プログラムの割付関数は、通常Actorに対し(6)のアルゴリズムで求め、冗長Actorについては、ループ上を、ずらしたものを用いた。

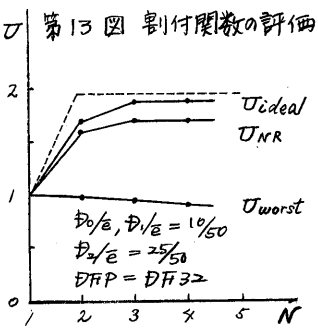
冗長Actorの割付関数: $m_r(a_i') \overset{\text{def}}{\equiv} \left(m_n(a_i) + \left[\frac{N}{2}\right]_{\text{切捨}}\right) \bmod N$ (7)
(a_i'はa_iと対応する冗長Actor, $m_n(=m_{practical})$は通常Actorの割付関数)

なお、以下のシミュレーションでは、冗長Actorは、通常Actorより低い優先度で実行される。

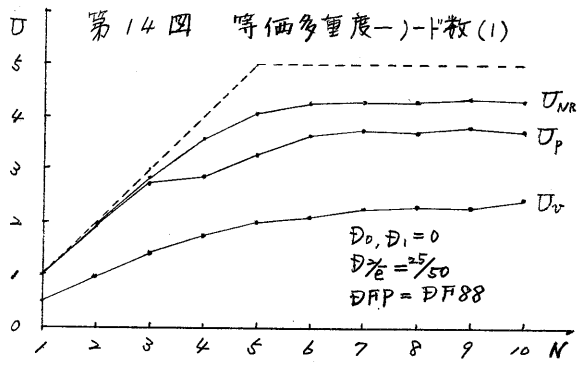
3-4 シミュレーション結果

第13図～第16図がシミュレーション結果を表わす。(図中の記号の意味については、第1表参照のこと。)第12図のプログラムで、すべてのリンク遅延を、(4)式で取り得る最小値に仮定し、入力から出力に至る経路に関するリンク遅延とActor処理時間の和の最大のもの(臨界経路)をTmとすると、これがプログラムのスループットの理論的下限を与える。これに対する等価多重度((6)式のTをTmとする)をUmとすると、Umは等価多重度の上限を与える。他方、明らかに等価多重度はノード数をこえることはできない。この二つの上限を示すものが、第13図～第14図の破線である。なおActor処理時間eiは0～最大値の一様乱数を用いて発生し、結果は、すべて10種の処理時間パターンについての平均値である。

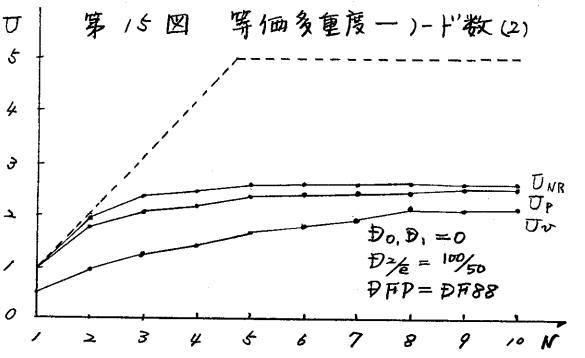
第13図は、(6)のアルゴリズムの評価を行なった結果であり、このアルゴリズムが、最適なものに近いことが例証されている。第14図第15図は、(4)式のリンク遅延時間を定めるパラメタD0, D1, D2の2種の例について、非冗長プログラムの多重度U_{NR}とU_p, U_vをプロットしたものであり、いずれもUm程度のノード数においてほぼ飽和しているのが示されている。これは、プログラムに固有な値であるDmが、ノード数を定めるための設計基準を与えることを意味すると考えられる。第16図は、ノード間リンク遅延と平



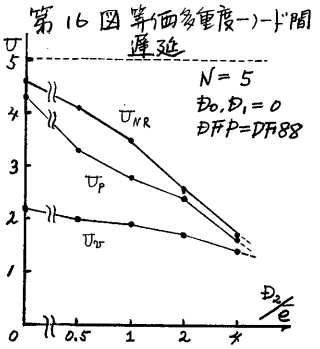
第13図 割付関数の評価



第14図 等価多重度ーノード数(1)



第15図 等価多重度ーノード数(2)



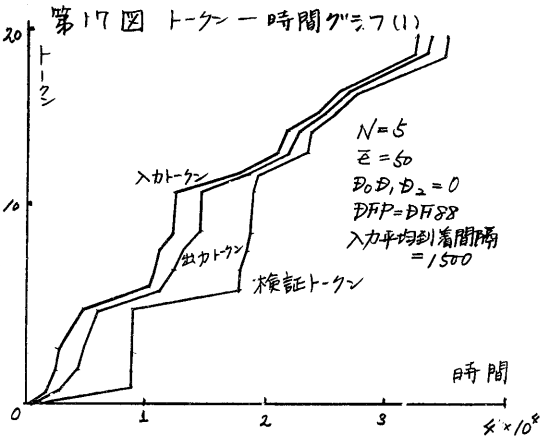
第16図 等価多重度ーノード間遅延

表1 記号表 N ---ノード数
- U ---等価多重度
- ē ---平均Actor処理遅延
- D0 D1 D2 ---(4)式参照,
- Uideal ---最適割付におけるU
- Uworst ---最悪割付におけるU
- U_{NR} ---アルゴリズム(6)の割付におけるU
- U_p, U_v ---3-3節参照,

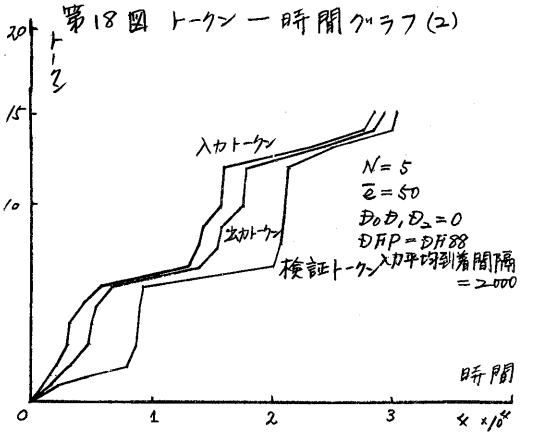
均Actor処理時間との比に対するU_{NR}, U_P, U_Uをプロットしたものである。比が1以上で各等価多重度の低下が著しく、ループネットワークの速度は、この比が1以下になるよう設計する必要があることを示している。第11図の設計に当たり、ループの速度とActorの処理時間を、それぞれ8Mbps, 数百～数千μ秒としたのはこの結論に従ったものである。

第17図, 第18図は、上述シミュレーションとは別に、第12図のプログラムに入力トークンがポアッソン到着するとして、冗長プログラムにおける、入力、出力、検証出力の3者の時間関係をシミュレートしたものである。これは、実時間信号処理等にデータフロー計算機を用いることを想定したものであるが、先のものと同様の結果を得ている。

第17図 トークン－時間グラフ(1)



第18図 トークン－時間グラフ(2)



むすび

本論文では、DFCが本来有している冗長性を生かした、自律的故障検出手法について提案した。これは、従来用いられてきた二重化等の固定的冗長ハードウェア付加方式に対し、可変的冗長ハードウェア付加方式とも言えるもので、柔軟で効率的な冗長付加方式である。又、本方式を適用するに際し、必要な「プログラムの対応する冗長部と通常部が別のハードウェア要素で実行される」という付加的条件は従来のDFCでは完全には満足されないため、新たに分散型DFCのモデルを提案し、合わせて、シミュレーションによる評価を行ない、その有効性を示した。なお、本方式に従う、自律的故障検出機能をもつ、中レベルActor（数百～数千μ秒）を対象とした分散形DFCを、現在製作中である。

[1] Dennis,J.B.,"Programming Generality,Parallelism and Computer Architecture", Information Processing 68

[2] Dennis,J.B.,"First Version of Data Flow Language",Lecture Notes in Computer Science,Vol-19,Springer-Verlag,1974

[3] Dennis,J.B.,Misunas,D .P.,"A Preliminary Architecture for a Basic Data-Flow Processor", 2nd Annual Symposium on Computer Architecture,1975

[4] Rumbaugh,J.,"A Data Flow Multiprocessor",IEEE Trans. on Computers,Vol C-26, No.2,1977

[5] Reames,C.C.,Liu,M.T.,"A Loop Network for Simultaneous Transmission of Variable-length Messages",2nd Annual Symposium on Computer Architecture,1975

[6] Liu,M.T.,"Distributed Loop Computer Networks",Advances in Computers,Vol 17, Academic Press,1978

[7] Misunas,D.P.,"Error Detection and recovery in Data-Flow Computer", Proc. of Int'l Conf. on Parallel Processing,1976