マイクロコンピュータとワークステーション 64－3
（1990．11．30）

フォルトトレラント・データフローマシンの実現

木村 寛治　　小柳 洋一　　当麻 喜弘

東京工業大学 情報工学科

データフローマシンは，プログラムに内在する並列性を引き出すために多数のプロセッサとそれらの相互接続を要素とする大規模なハードウェアを必要とする．その結果，システムの信頼性は低下してしまい，データフローマシンの耐故障化が必要になる．そこで，本論文は，パケット通信により処理を進めていくデータフローマシンの耐故障化の方式について論じ，通常動作時にはトークンを二重化し，2つのトークンの比較により誤りが検出された時のみ，三重化して多数決をとり誤り回復を行なう方式を提案する．そして，この方式を用いたフォルトトレラント・データフローマシンを，複数の MC68000 ボードで実現し，その評価を行なった．

Implementation of a Fault-Tolerant Data-Flow Machine

Kanji KIMURA　Youichi KOYANAGI　Yoshihiro TOHMA

Department of Computer Science, Tokyo Institute of Technology

2-12-1, Ookayama, Meguro-ku, Tokyo, 152, Japan

This paper describes the experimental prototype for implementing the fault-tolerant data-flow machine. Since a data-flow machine needs a large amount of processors, it is one of serious problems to maintain the dependability high. Therefore, some method for fault tolerance should be incorporated in such a data-flow machine. We propose the token duplication-comparison method for detecting an error and the token triplication-voting method for the recovery. The experimenatal prototype of a fault-tolerant data-flow machine is constructed by single boards of MC68000. Using this machine, the effect of a fault on the performance is evaluated.

1 まえがき

近年，計算機に要求される複雑・高度化した処理に応えるため，従来のノイマン型コンピュータに代わり，非手続き概念に基づくデータ駆動型の高性能計算機である，データフローマシンの開発が行なわれている．[1),2),3),4)]

データフローマシンは，プログラムに内在する並列処理性を効率良く引き出すために多数のプロセッサをネットワークで結合した分散型構成をとるのが一般的である．このためにシステムは巨大なものとなり，システム自身の信頼性を低下させることになる．そこでフォルトトレラント設計が必要不可欠となる．[5),6)]

そこで本研究では，データフローマシンが制御機構と演算機構のパケット通信により実行を進めていくことに着目し，各パケットを多重化して通信することによりシステム中の単一ユニット故障をマスクすることを目的とする．一方，システムの信頼度を上げるために冗長なパケットを用いて処理を行なうことはシステムとしての効率を低下させることになる．そこで無故障時のオーバヘッドを小さくするために，故障ユニットが検出されるまではパケットの冗長度を故障検出を行なえる程度に押え，故障が検出された時にはパケットを再送することにより，故障をマスクするという方式を提案する．

本研究はこのフォルトトレラント化の方式の実現について検討し，実際に MC68000 ボードを使用して実験機を製作し実験評価を行なった．以下 2 では，フォルトトレラント化しようとするデータフローマシンのモデルの概要を説明する．3 では，故障モデルを定義し，誤り検出，誤り回復の実現方法について述べる．そして 4 では，実験の結果を示す．

2 基本アーキテクチャ

2.1 計算モデル

データフロー計算モデルにおける計算は図 1 のようなデータフローグラフにより表現される．データフローグラフで，ノードは演算を表し，アークはデータの入(出)力線を示す．データ駆動計算機におけるプログラムの実行はグラフ上にトークンを流すことで行なわれる．計算を実行する規則は，入力データ (トークン) が全て到着した時にそのノードは発火し，ノードは発火すると入力からトークンを取り除き，演算結果を示すトークンを出力する．データフロー計算のノードの演算順序，結果は入力データのみに依存する．すなわち各ノードは互いに独立に並列実行することができる．

そのような並列処理環境下では関数が複数個所で実行可

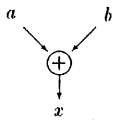

図 1: データフローグラフ

能となり，並列実行される．従ってその関数の実行を並列に動作させるためには論理的に関数本体のコピーを処理環境ごとに作る必要がある．また再帰関数の実行においても再帰のネスト間での並行処理を実現するためにはコピーが必要となる．関数本体のコピーという概念を物理的にどう実現するか，その解決法として

- 関数が呼ばれるたびに関数本体のコードをコピーするコードコピー方式
- 呼び出す関数ごとに色を与え，これを個々のトークンに付して処理環境を区別し，物理的にはコードを共有させる色つきトークン方式

がある．後者の色つきトークン方式では同一の色のトークンが入力アークに揃った時ノードが発火する．色つきトークン方式はコードコピー方式に比べてメモリ容量，アクセス時間の点で有利であることから，本研究では色つきトークン方式のアーキテクチャを採用する．

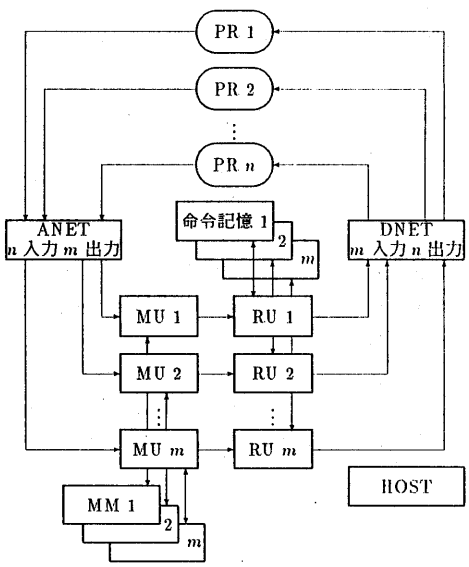

図 2: データフローマシンの構成

2.2 構成

本データフローマシンは図2に示すように，複数のマッチングユニット(MU)，読み込み更新ユニット(RU)，プロセッサ(PR)及びネットワーク(ANET，DNET)から構成される．MUには命令を実行するのに必要なトークンの待ち合わせを行なうためのマッチング記憶(MM)，RUにはデータフローグラフの各ノードの命令を格納する命令記憶(IM)がある．以下に各構成要素の動作概要を述べる．ここで，MUとRUの組を実行制御ユニット(CU)と表現する．

2.3 実行制御

データフローグラフの各ノードは最大2入力，2出力までとし，各ノードを識別するために関数番号(fn)とコード番号(cd)を割り当てる．各ノードの命令は全てのRUのIMに格納され，そのIMの形式を図3に示す．

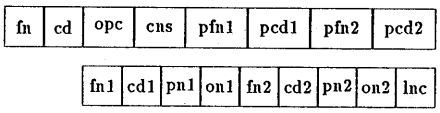

図3: 命令記憶の形式

IMは，fn，cd，演算コード(opc)，定数部(cns)，左側出力先のノードの関数番号(fn1)，コード番号(cd1)，入力ポート番号(pn1)，命令を実行するのに必要な入力数(on1)，右側出力先のノードの関数番号(fn2)，コード番号(cd2)，入力ポート番号(pn2)，命令を実行するのに必要な入力数(on2)のフィールドからなる．(pfn1，pcd1，pfn2，pcd2，lncは後述．)

次に，各ユニットの間を流れるパケットの形式を，図4に示す．

MUは結果パケット(RP)が送られてくると，ノードに必要な全ての入力トークンが到着したかどうかチェックする．RPには，起動番号(ist)，ループカウント(lc)，演算結果(res)，次のノードの情報(fn ~ on)のフィールドがある．1オペランド命令の場合(on = 1)はRPから直ちにオペランドパケット(OP)を組み立て，これをRUへ送出する．2オペランド命令の場合(on = 2)は，オペランドが揃ったか否か，すなわち次のノードの同じタグ(ist，lc，fn，cdが一致すること)をもつ対のトークンがすでにMMに到着しているか調べる．オペランドが揃わなければ，待ち合わせのためRPをMMに格納する．揃うとRP及びMM内のオペランド値(dat1，dat2)からOPを作成する．RUはOPが送られてくると，それに対する(fn，cdが一致する)命令

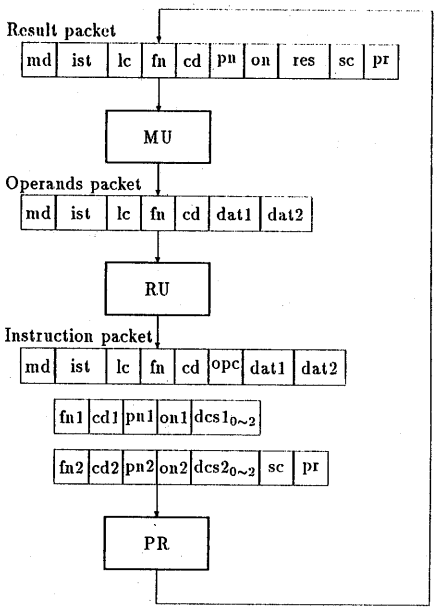

図4: パケットの形式

(opc)，次の命令の情報(fn1 ~ on1，fn2 ~ on2)を取り出し，OPと組み合わせて命令パケット(IP)をPRへ送出する．PRはIPのdat1，dat2に対してopcで指示される演算を実行し，その結果(res)をfn1 ~ on1，fn2 ~ on2と組み合わせそれぞれをMUに送る．

3 フォルトトレラント化の方式

次に，このようなデータフローマシンにおける，フォルトトレラント化の方法について示す．

3.1 故障モデル

まず，フォルトトレラント・システムに仮定する故障のモデルを規定しておく．

1. 故障箇所

   MU，RU，PR，及びそれらを接続する全てのリンク

2. 故障の種類

   単一ユニット及び単一リンクにおける永久故障，トランジェント故障，間欠故障などのあらゆる故障

3.2 誤り検出

単一故障を検出しその故障をマスクするためには，データフローマシンの並列性を利用して，三重化したCUからそれぞれ3個ずつトークンを出力し，そのトークンを受信

したCUがトークンの多数決を取り処理を進めて行けば良い．しかし，これではデータフローグラフにおける1つのアークに対応して生成されるパケットは9個となって，通常動作時のオーバーヘッドは大きくなりすぎ，現実的でない．このような冗長化に伴うオーバーヘッドを軽減させるために，通常動作時は，トークンの冗長度を二重化に抑えて誤り検出を行ない，誤りが検出されたときのみ，第3のトークンを要求して多数決を行なうという，二重化比較，三重化多数決を採用する．この場合，通常時はパケットの通信量は，1つのアークに付き4個に抑えることができる．

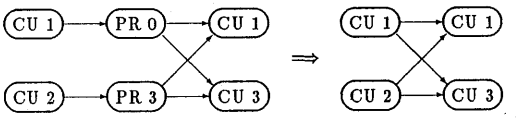

図5: 二重化されたトークン転送の様子と簡略表記

図5に，通常動作時のトークン転送の様子およびCUのみに着目した簡略表記をを示す．この図で示したように，実行可能となった命令は2つのCUに保持されている．この2つのCUから出力されたIPは2つのPRに送られ，実行される．PRでは処理が完了した後に，RPを2個複製し，それらを次の二重化したCUに出力する．このときのCUに入力する2つのトークンが通過したパスは互いに独立でなければばらない．図4におけるIP，RPの，送り元のCUを示すscと，演算を行なったPRを示すprは，通過したパスを調べることに利用される．

従って，トークンが故障ユニットを通過してきたとしても，2つのうち一方は必ず正常なトークンであるので，トークンの比較によって誤りを検出することが可能となる．

3.2.1 実行制御ユニットの選択

このとき，二重化されたRPは同一のCUに送出されなければ，トークンの比較をとることはできない．よって，これを実現するために，命令毎に，処理するCUの組をあらかじめ割り付ける方法を採用する．

RUは，左右出力先の次の命令をIMから読み出し，その命令に割り当てられている論理CU番号(lnc)を読み出す．lncは，処理するCU番号の組を表している(表1)．RUはこの表より，送り先CUの番号をIPのdcs1，dcs2に格納する．ここで，独立したCU番号とは誤り回復のとき用いる第3のCU番号のことである．

lnc	CU番号の組	独立したCU番号
0	0，1	2
1	0，2	3
2	0，3	1
3	1，2	3
4	1，3	0
5	2，3	1

表1: lncに対するCUの割り当て

3.2.2 プロセッサの割り当て

また，二重化されたトークンが独立のパスを通る必要性からCUは，独立したPRを選択しなければならない．選択は，実行しようとしている命令のcdによって行なう．

RUは，表2の(cd mod 12)に対応する，出力すべきPRの番号の組を知る．一方，自身の命令のlncで表1を引くと，CU番号の組のどちらかは自分のCU番号である．よって，自分と同じ命令を処理しているCUの番号と自分のCU番号の大小を比較することができ，これより表2のPRの組のどちらに出力すべきかを決定できる．表の独立したCU番号とは，誤り回復のとき用いる第3のPR番号のことである．

cd mod 12	小さい番号のCUが出力するPR番号	大きい番号のCUが出力するPR番号	独立したPR番号
0	0	1	2
1	0	2	3
2	0	3	1
3	1	2	3
4	1	3	0
5	1	0	2
6	2	0	3
7	2	1	0
8	2	3	1
9	3	0	2
10	3	1	0
11	3	2	1

表2: cdに対するPRの割り当て

3.2.3 マッチング記憶の構造

ここで，MMの構造について述べる．フォルトトレラント化を考えない場合には，2オペランド命令で相手が見つ

からない場合のみMMにRPが格納される．しかし，以上のようにRPの多数決を行なう場合には

- 到着したRPと一致がとれない状態(UNFIX)
- 一致がとれ，オペランドが揃った状態(FIX)
- 2オペランドの場合，一致はとれているがオペランドが揃っていない状態(UNPAIR)

が存在する．そこでMMの各トークンに一致状態を示すフラグを付加し，RPの一部をキーにMMから取り出し，そのフラグによって処理を行なっている．

また，データフローマシンの効率化を図るため，MMにハッシュ機構を適用し，連想アクセスを行なっている．連想アクセス用のキーとしては(ist, lc, fn, cd)を使用する．詳細はここで述べないが，挿入，削除操作が多数回発生するので効率を考慮して連鎖ハッシュを適用する．

関連してIMにも効率の点からハッシュ機構を適用し，命令のアクセス時間を短縮している．実際，IMは誤り回復の際にも頻繁にアクセスされる．IMはMMと異なり静的であるので，構造の簡単なオープンハッシュを適用する．

3.2.4 タイムアウト

以上のように誤り検出を行なうとき，ユニットの故障によって比較をとるべき2つのトークンの一方が消失した場合，またたとえ2つのトークンが到着しても，タグがある故障により互いに異なることが起きた場合には，二重化トークンの比較を行なうことができない．MUは二重化トークンの対が遅く到着することを期待してさらに待つことになり，やがて処理の続行ができなくなる．この状態を回避するために，MMにタイムアウト機構を設ける．二重化されたトークンの最初の1つがMUに到着した時にタイマーを起動させ，それは予め決めておいた時間制限内に対のトークンが到着しないとき(タイムアウトの発生)，誤りが生じたものとみなす．

3.3 誤り回復

二重化したトークンの比較あるいはタイムアウトによって誤りが検出されると，CUは実行を失敗した命令に対して誤り回復動作を行なう．誤りを検出したCUはデータフローグラフに対して2つ前のノードのトークンを保持しているCUにトークン再送要求を送出することによって，三番目のトークンを得て多数決をとり正しい結果を得ることができる．

図6は，'−'演算ノードにおいて結果の不一致が起きたときの誤り回復動作を示したものである．CU1で，二重化された'−'演算の結果に不一致が検出されると，CU1は'−'

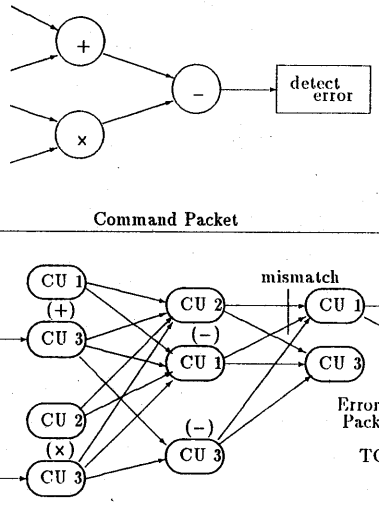

図6: 誤り回復動作の例

の1つ前のノード('+'および'×')のトークンを保持しているCUに対してその命令を再実行するよう指示するコマンドパケットを送出する．'+'および'×'のトークンを保持しているCUも二重化されているため，CU1は，'−'演算を行なったCUの組(CU1, CU2)とは独立したCUを表1より選択して('+', '×'共にCU3)コマンドを送出し，故障の可能性のあるCUに再送を要求することを防ぐ．コマンドを受け取ったCU3は，'+', '×'の実行結果をリカバリパケットとして，'−'を処理したCU1, CU2とは独立の第3のCU(CU3)へ送る．CU3はこれらのトークンに対して演算を実行し，CU1およびCU3へ送出する．CU1では，この第3のトークンによって多数決を取ることができる．CU3では，多数決は2つのトークンによって既に取れているので，この第3のトークンは捨てられる．このコマンドパケットからリカバリーパケットを生成し，多数決を採る過程において，PRも'−'演算を行なったPRの組以外のPRを使用する必要がある．これは，表2の独立したPR番号を参照することにより実現できる．

このようにして，不一致を検出したノードで第3のトークンを得るために，2ステップさかのぼってせいぜい2つのノードとそれらに対応するCUを探す必要がある．このためにはノードをさかのぼることが必要であるが，実現するために，IMに格納されている各命令にpfn1, pcd1, pfn2, pcd2のフィールドがある(図3 参照)．pfn1, pcd1は，左側の入力アークを1つ戻ったノードを示し，pfn2, pcd2は，右側の入力アークを1つ戻ったノードを示す．また，トー

クンの再送要求による命令の再実行をするためには，その命令を実行可能とするOPを保存しておく必要がある．そのために，RUに入力したOPを格納するためのオペランド記憶(OM)を設ける．

3.4 三重化多数決

上述のように2段前のノードを見つける方式において，関数の入口で誤りが検出された時に関数を呼び出したノードを求める必要がある．また，色の変化するノードの出力側で誤りが検出された時には，2段前のノードにおいて新しい色からその前の色がわからなければならない．このためには，トークン自身に関数の呼び出し側の情報を含まなければならない．これを実現するために U-interpreter[7]のような色の生成方式を用いることも考えられるが，実現困難である．よって，関数の入口や色の変化するノードの出力側のノードのようにトークンの再送により回復ができないノードにはトークンを三重化して送る．

3.5 保存トークンの破棄

命令の再実行をするためにはOPを保存しておく必要があることを述べたが，そのトークンを保存する側から見ればいつ再実行要求が来るか知ることはできず，そのため，すべてのトークンを保存しておかねばならない．しかし，OMは有限であるので，保存しているOPを破棄する必要がある．ここで，OPを捨てる時期について述べる．OPを捨てる時期は，そのトークンに対するノードにトークン再送の要求をすることはないと保証された時である．この保証を得るための方法として，トークンを受け取ったノードは2つ前のノードにトークンを受け取ったことを知らせる方法がある．この方法は命令を実行するごとに2つ前のノードのOPを破棄していくのでOMの領域が小さくて済むという利点があるが，特別なハードウェアを設けずにパケット通信により実現することはオーバーヘッドが大きくまた処理が複雑である．そこで，色の変わる関数およびループの出口では三重化されていることに着眼し，一度関数またはループを抜け出たトークンが関数内またはループ内のノードに対して誤り回復の要求をすることはないことから，関数およびループを抜け出る時に，関数またはループ内で使用したOPを捨てる方式を採用する．この方式は，関数またはループ内のすべてのOPを保存できなくてはならないので，OMの領域は大きくなるが一括してOMを破棄するため，処理が簡単でオーバーヘッドが小さい．また，ループ内のOP全てを保存しようとするとOPを格納する領域の大きさが静的には決定できず，非常に大きくなる可能性もあるので，ループカウントを更新するノードにおいても三重化し，その色を持つOPを破棄することにする．

また，RPの多数決を採り2オペランドの相手を探すことに使用されるMMのトークン破棄についても考える必要がある．RPの多数決が採れ命令実行に必要なオペランドが揃った場合にOPを出力し，すぐに破棄しても処理は継続できる．しかし，三重化の時や誤り回復により第三のトークンが到着する場合，すでに多数決が採れOPを出力しているにも関わらずタイムアウトが生じ不必要な回復を行なってしまう．よって，MMが有限であることから，最も影響が小さいものから破棄することにする．MMの一定の容量に達したら最も過去にOPを出力したトークンから捨てることにした．MUでは多数決が採られ2オペランドであれば相手が確定したトークンをキューに入れ，捨てる候補のトークンを管理する．

以上の保存トークンの破棄により，有限な記憶容量で規模の大きい問題に対処することが可能となる．

3.6 トークンの色について

関数を呼び出す時にはトークンのタグとして新たな起動番号を生成しなければならない．この時は多重化しても，一致や多数決をとることができない．そこでその起動番号の生成について論じる．その生成の一方式として図7にそのトークンの流れを示す．

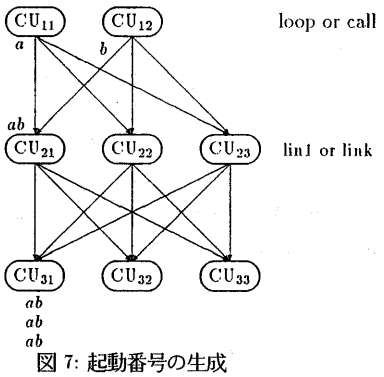

図7: 起動番号の生成

まず，起動番号のためのカウントを作り出す命令(loop, callノード)を実行するPRは，その演算として自分のプロセッサ番号と，カウンタより読み出した整数の組(pcnt)をデータ(a, b)とするトークンを出力する．次のノードは三重化されていて，そのノードのCUは2つのトークンが到着するのを待ち，2組のpcntをまとめて(ab)次のノードに出力する．次のノードでは，2組のpcntが一致する2つのトークンが存在する，そうでないときは，1組のpcntが一致する3つのトークンが存在する時にトークンの多数決

がとれたものとし，その一致のとれた pcnt によりシステムで唯一の色を生成することができる．

すなわち，起動番号を図 8 のように定義する．

PR 番号	カウント	PR 番号	カウント

図 8: 起動番号の形式

ここで，あらゆる単一故障に対処するには，pcnt をデータとするトークンが確実に消失したことを示すタイムアウトを設けること，起動番号を生成する 5 つの CU，PR がそれぞれ独立であることという制限が必要で，検討の余地が残されている．

4 実験

4.1 実験システムの構成

実験に使用する各ユニットは，マイクロプロセッサ MC68000 を用いた汎用の処理ボードに OS9 オペレーティングシステムを搭載したもので，各ユニットの機能は，C 言語によって記述されたプログラムにより実現する．ユニット間の通信は，8 ビット幅のパラレルポートを介して行なう．その通信速度は 45kbyte/sec である．本実験のための概念的な構成は，MU，RU，PR それぞれ 4 つで，ANET，DNET は完全グラフとする．

4.2 実験結果

まず基本的なプロトタイプ回路を作成し，4 つの MU，4 つの RU，4 つの PR の各グループを 1 枚ずつ，計 3 枚のボードで実現し，実験評価を行う．

実験を行なったデータフローグラフは，再帰関数による $5! \times 100$ の計算[8]で，次のような条件の下で実行させる．

1. 故障なし．
2. 単一の PR での，データ誤りのトランジェント故障．PR2 が '−' の演算を行なったとき，誤った演算結果を送出する．
3. 単一の PR での，パケット消失のトランジェント故障．PR2 が '−' の演算を行なったとき，演算結果を送出しない．
4. 単一の CU の永久故障．MU3 が RU3 へパケットを送出せず，CU3 の機能が永久に停止する．

実際に動作を行なった結果，全ての条件の下で正しい演算結果が出力された．それぞれの場合について，各パケット数と処理時間を表 3 に示す．

動作条件	OP 数	IP 数	RP 数	処理時間 (秒)
1. 無故障	217	197	668	6.60
2. 演算誤り	245	225	700	7.22
3. RP 消失	237	217	684	7.87
4. 永久故障	330	313	713	16.75

表 3: 誤り回復時の処理時間 ($5! \times 100$)

条件 2. の演算誤りは，誤りが速やかに検出され回復を行なうことができるので，パケット数もあまり増加せず実行時間についてもそれほど遅くならない．条件 3. のパケット消失の場合は，タイムアウト値により実行時間が左右される．このときの最適なタイムアウト値は，約 250msec である．条件 4. ではやはり多くのタイムアウト検出により処理するパケット数がかなり増え，実行時間もかなり長くなっている．

現在，MU に 4 枚のボード，RU に 2 枚のボード，PR に 2 枚のボード，ホストに 1 枚のボードを用いたシステムを実現している．そのシステムでの，3 行 3 列の行列の n 乗 ($n = 2, 4, 6, 8$) の計算の実行結果について述べる．

実験に使用した 3 行 3 列の行列の n 乗の計算のデータフローグラフを図 9 に示す．図 9 で [matrix mult] の部分は行列の乗算をする部分で，積と和のノードからなっている．また，アークの近くに 9 と書かれているところは，実際にはアークが 9 つ存在し，その入出力のノードも 9 個存在する．この図ではトークン破棄のためのノードは省略されていて，データフローグラフ全体では 500 ノード以上の規模になっている．なお，この数値演算は全て浮動小数点演算を行なっている．

各計算について，3 枚のシステムと比較した全処理時間を表 4 に示す．

処理内容 3 × 3 行列	RP 数	処理時間 (秒) 3 枚	9 枚
2 乗	1464	15.2	3.8
4 乗	4208	42.9	10.4
6 乗	6952	77.3	17.4
8 乗	9696	115.0	24.5

表 4: 処理時間の比較

MU，RU，PR のボード割り当ての比率が異なることから，単純に数値の比較はできないが，ボードを増やしたことによる並列性の効果は確認することができる．

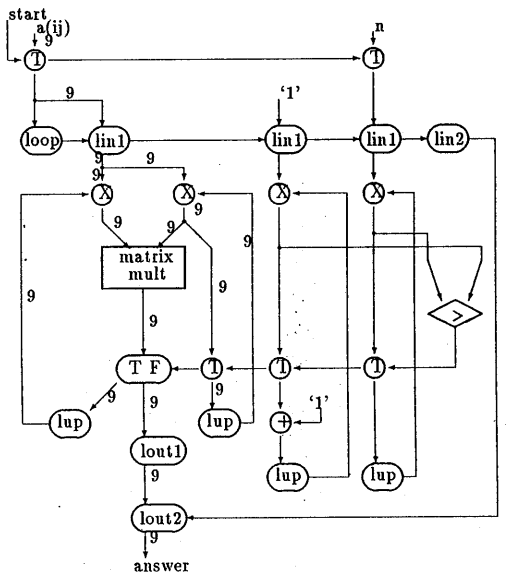

図 9: 3行3列の行列の n 乗の計算

5 まとめ

本研究では，フォルトトレラント・データフローマシンを実現し，その評価を行なうことを目標とした．まず，データフローマシンのモデルを提示し，そのシステム中の単一ユニットの故障を許容することができるフォルトトレラント・データフローマシンの実現法について述べた．実際にMC68000ボードを用いた実験機を製作し，トークンの二重化比較三重化多数決方式の有効性を確認できた．

今後の課題として，故障診断，システム再構成の問題がある．実験結果からも明らかなように永久故障の場合にシステムの実行効率がかなり落ちる．このことから耐故障能力を向上させるためには，エラーパケットの集合から故障ユニットを見つける診断アルゴリズムを決定すること，故障ユニットが検出された時に，そのユニットをシステムから切り離す方法を検討しなければならない．

また，ネットワークについては実験システムでは完全グラフとして構築したが，データフローマシンのフォルトトレランスを考える場合，特にシステム再構成においてネットワーク構成がきわめて重要な要素の1つであると思われる．この点からも，実際に稼働しているデータフローアーキテクチャに近い新実験システムを構築し，フォルトトレラント・システムの実現手法について，今後さらに研究が必要である．

参考文献

1) M. Amamiya, M. Takesue, R. Hasegawa, and H. Mikami: "Implementation and evaluation of a list-processing oriented data flow machine architecture", In Proc. 13th, Ann. Int. Symp. on Comput. Arch., pp. 10–13 (1986).

2) J. Gurd, C. C. Kirkham, and I. Watson: "The manchester prototype dataflow computer", Comm. ACM, **21**, 1, pp. 34–52 (1985).

3) 雨宮 真人: "関数型言語とリスト処理向きデータフロー・マシン", 情報処理学会誌, **26**, 7, pp. 765–779 (July 1985).

4) 島田 俊夫: "数値計算向きデータフロー計算機", 情報処理学会誌, **26**, 7, pp. 780–786 (July 1985).

5) D. P. Misunas: "Error detection and recovery in a data-flow computer", In Proc. 1976 International Conference on Parallel Processing, pp. 117–122 (Aug. 1976).

6) J. L. A. Hughes: "Error detection and correction techniques for dataflow systems", In FTCS, pp. 318–321 (June 1983).

7) Arvind and K.P.Gostelow: "The U-interpreter", Computer, **15**, 2, pp. 42–49 (Feb. 1982).

8) Y. Tohma, M. Chinuki, and K. Kimura: "Experiment of recovery mechanism of data-flow machines", 信学技報, **FTS 89-13**, pp. 17–24 (1989).