記　号　処　理　56－3
計算機アーキテクチャ　84－3
（1990．　9．　28）

並列処理計算機ＭＤＦＭでの
データフローノードマネージメントについて

寺田 秀文
静岡大学 工業短期大学部

多数のデータ駆動型処理要素を相互に結合することで、超多重処理の実現を計る並列処理計算機がＭＤＦＭである。ＭＤＦＭは機能化されたノード（アクタ）に対し集合化処理を施すような粗粒度実行モデルの採用やデータフローグラフへの履歴依存性の導入などで特徴付けられる。さて粗粒度実行モデルの効果を得るためには各処理要素内で局所的に処理可能なノード集合を効率的に実行できなければならない。本報告ではノード実行時においてコストが比較的大きいトークンマッチングを可能な限り直接的に行い、処理要素内の実行を効率化するノードマネージメントについて、スケルトン記憶と具現記憶に重点を置き検討する。

A Dataflow Node Management on A Parallel Processing Computer MDFM

Hidefumi Terada

College of Engineering, Shizuoka University
3-5-1, Johoku, Hamamatsu, Shizuoka 432, Japan

A parallel processing computer MDFM implements a ultra-multi processing by many interconnected data-driven processing elements. The MDFM is characterized by a coarse-grain execution model that a grouping procedure applies to functioned dataflow nodes(actors), a introduction of history sensitivity to the dataflow graph and so on. To gain a effect of coarse-grain execution model, each processing elements must equip a efficient execution mechanism for locally executable dataflow nodes. Because of one of the costly procedure in execution time is a token matching, we'd like to perform this matching directly. This paper reports a dataflow node management scheme that supports direct token matching on actor skeleton storage and its instantiation storage of the MDFM.

1　はじめに

コントロール駆動方式（ＳＩＭＤ・ＭＩＭＤ方式，演算パイプライン方式，ＶＬＩＷ方式など）やデータ駆動・要求駆動方式などの方式に基づいた多彩な並列処理計算機が広く研究されている。さらに近年ではシステムを構成する処理要素数を比較的大きく想定した超並列処理計算機の研究が半導体集積技術などの計算機製造技術を背景に盛んに行われるようになった。これら計算機の狙いは単一プロセッサの処理速度限界をその並列化で越えるのはもちろんのこと、新しい処理様式の模索でもあり、その重要性は広く認められている。現在までにも数多くの並列処理計算機が商用化されたが、並列処理計算機に対して指摘された幾つかの問題がそれらの上で十分に解決されているとは言い難い。さらに多数の処理要素で構成する場合には処理要素（処理速度，局所記憶量など）の検討だけでなく、システム構成（相互結合網の形態や通信路容量，入出力機構など），プログラミング言語とその処理系，応用などの検討をしなくてはならず、より多くの課題を抱えることとなる。

並列処理において命令レベルでの非同期並列処理が自然に制御可能，多数の処理要素上への処理分散が命令レベルで容易，動的モデルでの多重プロセス環境の提供などの優れた特性がデータフローモデルにはあり、超多重処理のような処理様式の基礎モデルと考えられている。このモデルは非常に簡潔であるが、ハードウェア化には克服しなくてはならない問題点が幾つか存在し、現在までにも多種のシステム[1][2][6][7][8][9][13]が試作・研究されている。データフロー計算機を実現する際、その達成度を左右する重要な要因にデータフローグラフの実行モデルの設定がある。基本的にデータフロー計算機はデータフローグラフを直接実行する細粒度（命令レベル）実行モデルであるが、小さな実行単位が生成する多量のトークンが処理要素間ネットワークを介して通信されれば通信路がボトルネックとなり、最悪時には計算機が縮退する危険性に陥る。これに対して局所的に処理可能なノードの集合を単一処理要素に割付け、処理要素間通信を抑制するマクロアクタモデルなどの粗粒度実行モデルが幾つか提案されている。

筆者もデータ駆動型の並列処理計算機ＭＤＦＭを研究中であり、データフローグラフの開発支援システムであるＤＦＧ／ＤＤの開発を終えている[16]。ＭＤＦＭは多数のデータ駆動型処理要素の相互結合で構成し、これに採用する粗粒度実行モデルを考慮したアクタ集合やデータフローグラフへの履歴依存性の導入などで特徴付けられる並列計算機である[17]。本報告は処理要素内でのトークンマッチングを可能な限り直接的に行い、実行時オーバーヘッドを減少させるようなデータフローノードマネージメントについての検討を、記憶領域の有効利用と実現コストの考慮の下でスケルトン記憶と具現記憶を中心に行う。

以降ＭＤＦＭのシステムと処理要素のアーキテクチャを実行モデルやアクタ集合などから説明した後、プログラムの登録，呼出し，実行におけるデータフローノードマネージメントについての検討を行い、最後に今後の課題について述べる。

2　並列処理計算機ＭＤＦＭのアーキテクチャ

2.1 ＭＤＦＭの構成

ＭＤＦＭの処理要素はプロセッサ，局所記憶，通信機構で構成されるデータ駆動型の処理要素であり、大域的な資源で発生する資源競合を回避するためにすべての資源は処理要素上に分散する。図-1のＩＰは外部との操作を支援するインターフェースプロセッサであり、利用者インターフェース，ＭＤＦＭへの静的なアクタ割付け，それを支援するアクタ分散状況データベースの管理が主な役割りとなる。処理要素間の結合網としては静的な結合網（ハイパートーラスやｒ進ｎキューブなど）や動的な多段スイッチ網など、さらにそれらへの大域網付加などの形態が考えられるが、現在は単純化のためにトーラス型の隣接結合網として検討を行っている。また入出力機構も重要であり、処理要素間の結合網はこれに対する考慮も含めて決定しなければならない。

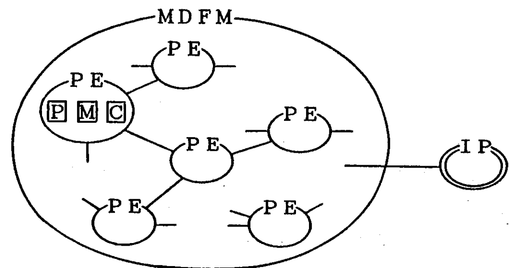

図-1　ＭＤＦＭシステム構成

2.2 データフローグラフの実行モデル

データフローグラフ[3]を直接実行する細粒度実行モデルの実行時並列度はデータフローグラフに表現されたものと同一になるが、資源競合やトークン通信などの問題から実機ではその十分な達成が困難である。これに対して、Gaudiotらのマクロアクタモデル[4][5]、坂井らの強連結枝モデル[12]などの粗粒度実行モデルが提案・実現されている。マクロアクタモデルは効率的なプログラム実行を現実的な計算機上で行うために処理の局所性に基づいて集合化されたアクタ（マクロアクタ）を単一処理要素に割付けて実行させるモデルであり、マクロアクタの実行時には処理要素間ネットワークを介して通信されるトークン数が減少され、かつ各処理要素の能力が引出せることになる。また強連結枝モデルはデータフローグラフに強連結枝ブロックと呼ばれる不可分ブロックを指定し、これを同一プロセッサ内のレジスタファイル上でパケットを作らずに実行させるものであり、これにより巡回パイプラインを介さない実行を実現し処理効率の向上を計るものである。強連結枝モデルはそのブロック内部の処理をコントロール駆動として実機EM-4[13]に実現されている。

またデータフローグラフ実行時に高い頻度で出現することが報告[15]されているフロー制御のためのノード（スイッチやゲートなど）はプログラム実行に本質的でなく実行オーバーヘッドであると共にトークン数を増加させる要因でもある。従ってＭＤＦＭの実行モデルではこれらをオペレータなどのノードに発火条件として内含させ、算術論理演算などの実行と独立に発火条件検査を行うことで粗粒度実行モデルの効果をさらに促進する。またデータフローグラフに適用される処理の局所性に基づいたノード集合化処理が粗粒度実行モデルでは特に重要であるが、ノード集合化処理が計算機システムに依存したプログラムの最適化処理であるために汎用アルゴリズムの開発は困難である。ＭＤＦＭでは処理要素ハードウェアが直接実行する実行単位であるノード集合を機能化し、それらに対して集合化処理を適用する。これは集合化処理の対照となるノードの水準を高めることで、この処理の負担が軽減されることを期待するものでもある。また集合化処理は主にプログラム翻訳時に行われる、プログラムの静特性による最適化であるが動特性をも含めた最適化がより望ましい。ＭＤＦＭでは動特性による再最適化をプログラム実行後に行う検討も進めている。

2.3 アクタ集合

ＭＤＦＭの実行単位は基本演算（basic operation），蓄積演算（accumulative operation），マルチキャスト（multi-cast），メモリ（memory）の４群のアクタである。これらのアクタはデータフローノードに発火条件を付加するなどで機能化したものである。従ってすべてのアクタはフロー制御用ノードのゲートなどを含むことが可能となる。ここでマルチキャストとメモリは文献[16][17]の転送とデータのアクタにそれぞれ対応する。

基本演算アクタと蓄積演算アクタは算術論理演算のオペレータであり、基本演算アクタは１または２入力オペレータで、蓄積演算アクタはデータ蓄積領域とカウンタを持つｎ入力オペレータである。また蓄積演算には、①ｎ個の可変なオペランド領域を持たせる必要がない，②各トークン到着が発火を招くために発火制御が容易に行える，③プロセッサ内のレジスタが有効に利用できる，④ｎオペランド入力のオペレータを実現することでトークン数の減少が期待できるなどの特徴がある。

マルチキャストアクタはｍ個の行先アドレスを持つことが可能なｍ出力アクタであり、さら各行先アドレスに付加される転送条件により選択的なトークン転送が可能である。従ってマルチキャストアクタはフロー制御用ノードである複写や分配を含むことが可能となる。幾つかの同一データを処理要素間で転送するような場合は行先処理要素内のマルチキャストアクタにデータを１度だけ転送すればよく、その結果として処理要素間の通信量を減少できる。また次に述べるメモリアクタとリンクすることで動的なデータ（メモリアクタ）参照も要求再送手順を必要としない効率的な実現が期待できる。マルチキャストアクタは複数個の行先アドレス持ち得る可変長アクタであるが、他アクタの記憶占有量との統一性のために行先アドレス数は固定数とし、それを越える場合には新たなアクタをリンクする方法で現在検討を行っている。

ＭＤＦＭでは共有・構造データも他アクタと同様にメモリアクタとして各処理要素へ分散する。メモリアクタはデータを保持する領域と構造表現のための領域を持ち、発火後に消滅するタイプだけではなくプログラマの明示的な指示で発火後もシステムに滞在可能なアクタである。これによりデータフローグラフに履歴依存性が導入されることになる。基本的にデータフロ

ーグラフへの履歴依存性導入はデータフローモデルの基礎となる関数性を破るが、モデルの範疇に留めるような履歴依存性を許したデータフローグラフの報告も既に成されている[10]。

3 処理要素の構成

3.1 プロセッサ

プロセッサはプログラム実行時のアクタ具現を保つ具現記憶（レジスタを含む），命令パイプライン方式の演算回路，各アクタ群に対する実行循環系，それらの制御回路などから構成される（図-2）。

具現記憶ＡＩＳ（Actor Instantiation Storage）はプログラム実行時の記憶割付け操作を簡略化するためにブロック化され、さらに各ブロックは命令領域とデータ領域に分割される。同一アクタの異なる具現において命令部分は命令領域で共有され、データ部分はデータ領域でそれぞれが保持される（命令コード共有方式）。記憶の効率的な利用を考えれば、データ領域は全領域上での動的管理が望ましいが実行コストの面で不利と考えられるため、ＭＤＦＭでは各命令領域に幾つかの具現を保てるだけのデータ領域を割当てている。

演算回路ＦＵは具現記憶から送られる発火可能な演算（基本演算と蓄積演算）アクタを実行する。実行結果は基本演算ならば行先アドレスに従い即座に転送され、蓄積演算ならば演算が終了するまで具現記憶（レジスタ）に戻される。また各アクタに付加された発火条件とマルチキャストアクタに付加された転送条件の検査回路は演算回路と独立に設けられる。

3.2 局所記憶

局所記憶ＡＳＳ（Actor Skeleton Storage）にはプログラムから翻訳されたアクタスケルトンが格納される。これをスケルトン記憶と呼んでいる。実行されるプログラムのすべて（関数や構造データなど）はどこかの処理要素のスケルトン記憶に予め存在しなくてはならない。またスケルトン記憶もブロック化されるが具現記憶とは異なり各命令領域には初期データを保持するだけのデータ領域が割付けられる。

3.3 通信機構

通信機構は処理要素間，処理要素とＩＰ間のパケット通信を行う。ポート数，通信路容量，結合形態は処理要素の能力などに依存するが、多数の処理要素を想定しているために高速な通信網が必須である。前述したように現在はトーラス型結合で検討をしている。

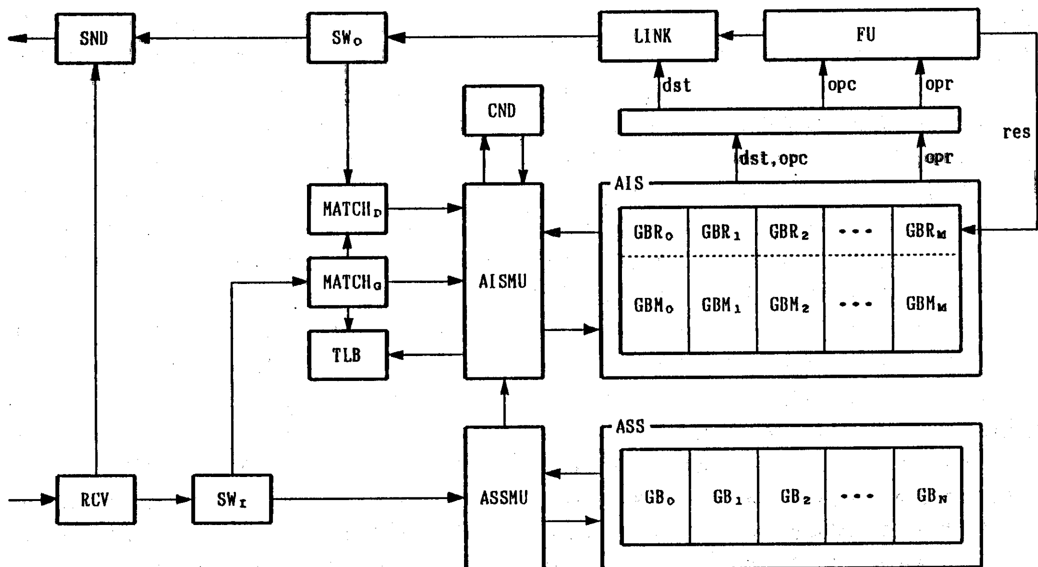

図-2 ＭＤＦＭ処理要素（ＰＥ）の構成

4　データフローノードマネージメント

4.1 グラフブロック

プログラム呼出し時の具現記憶割付けのために記憶はブロック化されており、この単位をグラフブロック（graph block）と呼ぶ。またプログラムは幾つかのグラフブロックに分割され、プログラムを表現するグラフブロックの集合をグラフブロック集合と呼ぶ。スケルトン記憶と具現記憶のグラフブロックは図-3のようにリンク部（LF），ノード部（NF），データ部（DF）から構成される。リンク部にはグラフブロック間リンク情報（<pe.as>または<pe.ai>，処理要素番号とスケルトン記憶または具現記憶におけるグラフブロックのベースアドレス）とグラフマネージャであるかなどのタグ情報が保持される。ノード部にはアクタ識別子，命令コード，オペランドアドレス，行先アドレスが保持される。ここでグラフブロック内の各ノードが保持する行先アドレスは<l.n.p>（lはグラフブロックのリンク部上のアドレス，nはグラフブロックのノード部上のアドレス，pはオペランド位置）で表現されている。

ＭＤＦＭ内での完全な行先アドレスは<pe.as.n.p>で与えられるが、ＭＤＦＭが大規模なシステムを想定しているためにそのビット数は比較的長くなる。従ってグラフブロック内のすべてのノードが完全なアドレスを保持するのは記憶の効率的な利用の面から望ましくない。また完全アドレスを保持する方式を採用したとしてもプログラム実行時のトークンマッチングを直接的に行おうとした時、全アクタの行先アドレス書換えという非常に重い操作がプログラム呼出し時に必要となる。故にグラフブロック間リンク情報をリンク部に保持することで記憶の使用効率を良好に保ちながら直接マッチングを支援するための呼出し操作を軽減する。実行時における行先アドレス決定に２回の記憶参照（ノード部の参照で<l.n.p>、その<l>でリンク部を参照）を必要とするが、処理要素内の循環パイプラインには影響を与えないと考えられる。Papadopoulosらがマルチプロセッサ Monsoon上に実現しようとしているETS（明示的トークン記憶）[11]はブロック内のトークン記憶位置を翻訳時に決定し実行時にそのブロックを動的にかなり大きな記憶の上へ割付けるものであるが、ＭＤＦＭのグラフブロックはこの動的割付けを分割して管理する方法であるとも言える。

4.2 プログラム翻訳と登録

前述したようにＭＤＦＭでは実行されるプログラムはそれ以前にスケルトン記憶内に割付けられる。主にこの割付けはインターフェースプロセッサがプログラム翻訳時に抽出した処理の局所性に基づいて行う最適化処理である。従ってこの処理はプログラムの静特性をＭＤＦＭの状態（スケルトン記憶の使用状態，具現記憶量，ＦＵ能力など）に適合させる処理となる。

ある高級プログラミング言語で記述されたプログラムはデータフローグラフ，ＭＤＦＭのアクタグラフの順に翻訳され、静特性解析による処理の局所性やグラフブロック長などに基づきアクタ集合化処理が成される。さらにグラフブロック間の依存関係を考慮し、各グラフブロックを割付ける処理要素を決定する。ここでプログラミング言語とその処理系（ＭＤＦＭアクタグラフへの変換，処理の局所性解析，アクタ集合化処理，処理要素への割付けなどを含む）については今後の課題であるため、本報告ではプログラムが図-4のようにある処理要素上のスケルトン記憶に割付けられたものと仮定して以降の議論を進める。

LF	GB_0	GB_1	GB_2	…
NF	N_0	N_1	N_2	…
DF	v_0	v_1	v_2	…

a) ＡＳＳグラフブロック

LF	GB_0	GB_1	GB_2	…
NF	N_0	N_1	N_2	…
DF	i_0	v_0	v_1	…
	i_1	v_0	v_1	…
	⋮			

b) ＡＩＳグラフブロック

<c> <d [.d]> <l.n.p>

operands　dest.

c) ノード部（NF）内の表現

* LF:link field, NF:node field,
DF:data field,
GB:graph block address, N:node,
i:instance, v:value, c:OP-Code,
d/l/n:address on DF/LF/NF, p:port

図-3　グラフブロックの構造とノード部内の表現

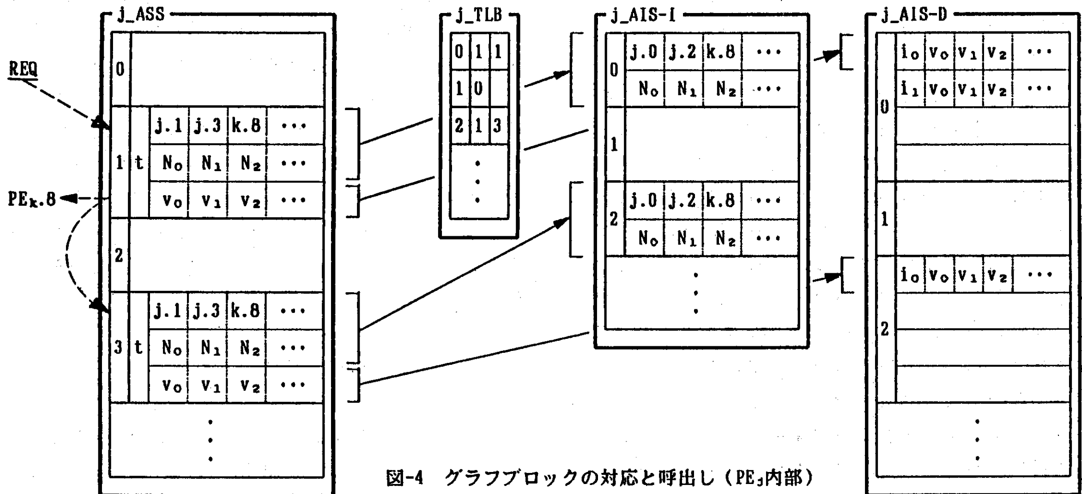

図-4　グラフブロックの対応と呼出し（PE_j内部）

4.3 プログラム呼出し

プログラム呼出し(invoking)はグラフブロック集合内の全グラフブロックの呼出しを管理するグラフマネージャに呼出し要求を送ることで起動される。呼出し要求を受け取ったグラフマネージャは自身を呼出すと共にこの呼出し要求を他のグラフブロックに伝える。スケルトン記憶内のグラフブロックは具現記憶へ以下のように呼出される。

呼出し要求を受け取ったスケルトン記憶管理機構（ＡＳＳＭＵ）は具現記憶管理機構（ＡＩＳＭＵ）にグラフブロック領域の割付け要求をし、スケルトン記憶内のグラフブロックを渡す。具現記憶管理機構がＴＬＢを参照することで空領域を見つけ、これにグラフブロックをロードする。この時、同一グラフブロック集合内のグラフブロックが同一処理要素内に幾つか存在する場合はそれらグラフブロックのリンク部（グラフブロック間リンク情報）をスケルトン記憶内のアドレスから具現記憶内のアドレスに書換える。これにより同一処理要素内のグラフブロックを渡るトークンのマッチングを直接的に行うことが可能となるが、処理要素を渡るグラフブロック間リンク情報の書換えはコスト大である故に現段階では考慮していない。具現記憶領域の確保，リンク部書換え，グラフブロックローディングと共にＴＬＢにグラフブロックのスケルトン記憶内でのグラフブロックアドレスを書込み、存在ビットをセットして呼出し操作が終了する。

図-4のように具現記憶は命令領域とデータ領域に分割されているため、命令領域にノード部とリンク部を、データ領域にデータ部（図では具現タグi_0のデータ部）をロードすることになる。

4.4 プログラム実行

プログラムは処理要素間と処理要素内で高度に分散されて実行される。処理要素間の実行分散は独立した処理要素の独立実行で行われ、その効率的な実現はアクタ集合化処理に依存する。処理要素内では２レベルの実行分散が行われる。第１に各アクタに付加された発火条件とその演算が独立実行可能であり、これを異なる機構に分散して実行する。発火条件の検査は各アクタの演算実行とは順序関係がなくトークン到着順で実行でき、さらに演算実行に先行して発火条件検査が行われることで演算実行が不必要となる場合もあり得る。このように発火条件検査と演算実行を分解するで発火条件の付加から派生するアクタの実行オーバーヘッドは無視できると考えられる。第２にＭＤＦＭの４アクタ群はそれぞれの実行内容が異なり、一過性の演算である基本演算アクタや滞在型演算である蓄積演算アクタはＦＵ付近で、メモリアクタは具現記憶付近で、マルチキャストアクタは具現記憶や通信機構付近で実行される。これらに対しＭＤＦＭでは具現記憶とＦＵ間の基本演算系，具現記憶（レジスタ）とＦＵ間の蓄積演算系，具現記憶内のメモリとマルチキャスト系，他処理要素との系のような実行系を設定しているが、これについては文献[17]を参照されたい。

図-5に基本演算アクタの演算実行を示す。図において、$\mathrm{MATCH_G}$は他処理要素から到着したトークンの行先アドレスの第２要素、スケルトン記憶内のグラフブロックアドレス<as>をＴＬＢ参照で具現記憶内のグラフブロックアドレス<ai>に書換え、トークン行先アドレスを処理要素内の表現<ai.n.p>に変換する。この行先アドレスは命令領域内のノード（命令コード，オペランドアドレス，行先アドレスを持つ）を直接指示しているため、$\mathrm{MATCH_D}$が行うトークンマッチングは具現タグ<i>を持つデータ部のマッチングとなる。対応するデータ部が存在すれば、データ部アドレス<df>と命令領域のノードが保持しているオペランドアドレス<d>で指示されたオペランド（アドレスは<df.d>）の到着による発火可能性の検査が行われ、さらに発火可能であれば対となるオペランドと共に到着したデータを演算回路に送る。またデータ部が存在しない、つまり到着したトークンが新たな具現タグを持つのならばデータ領域上にデータ部を確保し、具現タグとデータを書込む（動的な領域管理については別報告）。ここで発火可能性検査の終了時には命令コードや結果トークンの行先アドレスが既に確定しており、オペランド待ちの状態にある。従って$\mathrm{MATCH_D}$は検査終了時に待ち合わせ回路に信号を送り、これらとの同期をとる。図のTRNSは具現記憶の命令領域内に保持されている結果トークンの行先アドレスの第一要素、グラフブロックのリンク部上のアドレス<l>をグラフブロックのリンク部を参照することで、他処理要素へならば<pe.as>、処理要素内へならば<ai>に変換する。ここで<pe.as>に変換された行先アドレスを持つ結果トークンは処理要素間通信と行先処理要素でのグラフブロックアドレスの変換が必要であるが、<ai>に変換された行先アドレスは次のアクタ具現を直接指示するので効率的な循環が期待できる。最後にアクタの演算が演算回路ＦＵで実行され、LINKにより結果トークンがルーティングされる。

さて図からも容易に推測されるが、オペランドマッチング部はデータ部のマッチングや発火可能性検査の他、データ領域の動的な管理が必要であるために処理要素内循環パイプラインのボトルネックとなる。従って特に効率化を計らねばならない機構である。例えば、高い確率で同一の具現タグを持つトークンが連続することから、具現タグとデータ部アドレスの対を幾つか保持するようなバッファを$\mathrm{MATCH_D}$に設け、マッチングをバッファ内検索とデータ領域内検索とで並行して行う方法が考えられる。またアクタの演算が演算回路ＦＵで実行される以前に確定している結果トークンの行先アドレスをマッチング機構などに先送りするような先行機構をLINKに付加することも考えられる。このような機構で効率的な循環が期待できるのだが処理要素のハードウェア量に直接に影響するため、検討事項の１つである。さらにＭＤＦＭではプログラム実行時に得られる情報を利用したアクタの再最適化（プログラム実行後の再最適化）を計画している。これはプログラムの動特性を処理要素やグラフブロックへのアクタ割付け操作に反映されるのが目的であるが、プログラム実行時に抽出すべき情報の選択や再最適化アルゴリズム、それを支援する機構なども検討課題である。

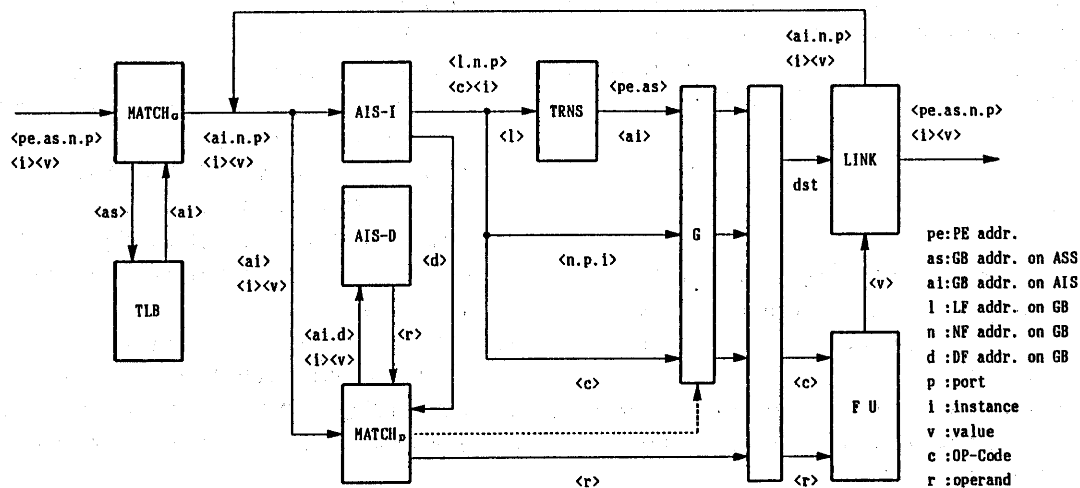

図-5　処理要素内でのアクタ実行

5 まとめ

多数のデータ駆動型処理要素の相互結合で構成される並列処理計算機ＭＤＦＭは超多重処理の実現が目的である。ＭＤＦＭの処理要素は比較的大きなスケルトン記憶（局所記憶）と具現記憶，命令パイプライン方式の演算回路，操作の異なる各アクタ群に対して設定される実行循環系，それらの制御機構で構成される。また現実的なデータ駆動型計算機上で効率的なプログラム実行が期待される粗粒度実行モデルの一実現例として実行単位となるアクタ（機能化されたデータフローノード）集合を定義し、それらに対して集合化処理を行うようなモデルを採用する。さらにメモリ概念のないデータフローモデルに履歴依存性をメモリアクタで導入し、一般化を進めるのと同時に新しい処理様式の模索を計る。

本報告ではＭＤＦＭにおけるデータフローノードマネージメントの検討を処理要素内のスケルトン記憶と具現記憶を中心に行った。このノードマネージメントにおける基本方針は処理要素内で局所的に伝搬されるトークンに対するマッチングを直接的に行い、トークンマッチングによる実行オーバーヘッドを減少することであり、記憶領域の効率的な利用やプログラム呼出し・実行における操作コストなどの面からグラフブロックでのマネージメント方法を提案した。

最後に各アクタの効率的な実行系の検討を単一処理要素でのシミュレーションで行うことが当面の課題であり、その後に相互結合網などと共に複数処理要素でのシステム達成度評価を行うこととなる。さらに文献[14]で解説されているプログラミング言語と処理系（最適集合化処理も含む）なども並行して検討すべき事項である。

【参考文献】

[1]Amamiya,M.. Datarol Processor : An Ultra-multi-processing Architecture for Massively Parallel Computing, Proc. Parallel Processing, 1988

[2]Arvind, Nikhil,R.S.. Executing a Program on the MIT Tagged-Token Dataflow Architecture, Proc. PARLE, 1987

[3]Davis,A.L. Keller,R.M.. Data Flow Program Graphs, IEEE Comp., Vol.15, No.2, 1982

[4]Gaudiot,J.-L.. Structure Handling in Data-flow Systems, IEEE Trans. Comp., Vol.C-35, No.6, 1986

[5]Gaudiot,J.-L., Najjar,W.. Macro-Actor Execution on Multilevel Data-Driven Architectures, Proc. Parallel Processing, 1988

[6]Gaudiod,J.-L., Bic,L.. Data-Flow : A Status Report, ACM Comp. Arch. News, Vol.17, No.6, 1989(ISCA '89)

[7]Gurd,J.R., Kirkham,C.C., Watson,I.. The Manchester Prototype Dataflow Computer, CACM, Vol.28, No.1, 1985

[8]Gurd,J.R., Kirkham,C.C., Bohn,W.. The Manchester Dataflow Computing System, Dongarra ed. Experimental Parallel Comp. Arch., North-Holland, 1987

[9]McGraw,J.R.. Dataflow Working Group Summary, Simon ed.. Instrumentation for Future Parallel Computing Systems, ACM Press, 1989

[10]西川，寺田，浅田．履歴依存性を許すデータ駆動図式，電信学論，Vol.J66-D，No.10，1983

[11]Papadopoulos,G.M., Culler,D.E.. Monsoon : an Explicit Token-Store Architecture, Proc. ISCA '90, 1990

[12]坂井，山口，平木，弓場．データ駆動型シングルチッププロセッサEMC-Rにおける強連結枝モデルの導入，電情信学，データフローワークショップ1987 予稿集，1987

[13]Sakai,S., Yamaguchi,Y., Hiraki,K.. An Architecture of a Dataflow Single Chip Processor, Proc. ISCA '89, 1989

[14]関口，山口．データ駆動計算機用の高級言語と処理系，情処学誌，Vol.31，No.6，1990

[15]山口，戸田，弓場．先行制御機構をもつデータ駆動計算機EM-3の評価，電情信学論，Vol.J72-D-1，No.3，1989

[16]寺田．データフローマルチプロセッサのためのシミュレータDFG/DD，38回情処学全大，1989

[17]寺田．並列処理計算機MDFMのアーキテクチャ，情処学計算機アーキテクチャ研報，79-16，1989