計算機アーキテクチャ59−12
オペレーティング・システム29−12
(1985・10・25)

パイプライン処理システムにおける関数型言語の実行方式

A Design Approach for Pipelined Processing System Using Function Level Language

三科 雄介　坂井 賢一　中村 維男　重井 芳治
Yuusuke MISHINA　Ken-ichi SAKAI　Tadao NAKAMURA　Yoshiharu SHIGEI
東北大学工学部
Tohoku Univ.

1．はじめに

現在の汎用計算機は、ペンシルヴァニア大学の von Neumann が提案した'プログラム内蔵方式'に、その基礎をおいている。こうしたノイマン型計算機の性能向上を支えてきたものは、高速論理素子の開発、および実装技術の改善である。しかし、これらの手法にも限界が見え始めている。

ノイマン型計算機における構成上の問題点として、単一プログラム・カウンタによる逐次制御方式、CPU-メモリ間におけるバス・ボトルネックが挙げられる。これらの問題点を解消、あるいは緩和するため、マルチプロセッサによる並列処理計算機や、データフロー計算機が研究されている。

一方、計算機システムの複雑化に伴い、既存のプログラミング言語によるソフトウェア保守の困難性が表面化している。このため、数学的計算モデルに基づく関数型言語が注目されている。関数型言語の特徴として以下の点を挙げることができる。プログラムの記述性が高く、アルゴリズムに密着したレベルでプログラムが書ける。実行順序という非本質的な概念がなく、簡潔な記述のうちにアルゴリズムに内在する並列性を自然な形で表現できる。このため関数型言語を指向した並列処理計算機のアーキテクチャの研究が盛んである。

計算機システムの構築を行う場合、処理の対象となるデータの構造及び処理を記述する言語の仕様は、システムの構成に大きく影響する。

特に関数型言語は従来の計算機システム上では、高速に実行できないため、関数型言語の性質を意識した新しい原理によるマシンアーキテクチャの研究が必要となる。

関数型言語とよばれるものは様々だが、基本となっているのはλ式と考えられる。λ式を評価実行する場合、その方法は2つに大別できる。第一に変数と値の結合、いわゆる環境を作成して簡約を進めていく方法がある[1]。第二の方法ではλ式を結合子に変換し、これを簡約する[2]。後者は前者と比べて簡約に要する項のサイズが増大するが、β簡約における変数名の衝突がなく、簡約化がより簡単な操作で実現できる。

言語ＦＰ[3]は、結合子の研究に関係して生まれた言語である。ＦＰは、J.Backus により提案された言語で、対象とするデータ構造を列（オブジェクト）として扱い、この列に対する演算としての関数、さらに関数の組み合わせを表現する構成子(ＰＦＯ)を用意している。

ＦＰの処理系を考える場合、システムを高速化するために要求される事項として以下を考えることができる。

1．ＦＰのプログラムは並列性を持つ関数列の集まりとして表現される。このため関数の処理の多重性を実現する必要がある。
2．リスト構造を対象として操作を行う場合、リストをたどることによる逐次性が存在する。このデータ量に比例するアクセス遅延を短縮する必要がある。

これらはＦＰによらず、関数型言語でリスト処理を行う際に共通する問題点でもある。その解決を目指したものとして、文献[4]の研究があげられる。

そこで著者らは、リスト構造をたどることにより生ずる逐次的な時間のずれを、縦続接続されたプロセッサ上における二次元の時間空間図で、空間方向へ処理の流れを展開することで多重性を引き出す方法について検討している[5]。

本システムは非同期のパイプライン構成をとり、リスト構造で表現されたデータ構造に対し、関数型言語ＦＰで記述されるプログラムの処理を行う。本システムは、パイプラインにおける各セグメントの機能を可変なものとすることで、パイプライン処理における汎用性を追及している[6]。このことにより、システムに与えられる種々の問題に応じて、動的にパイプライン処理を行うことが可能となる。

本稿では、リスト処理を指向したパイプライン処理システムにおける関数型言語ＦＰの実行方式について報告する。まず研究の背景としてＦＰの特徴とその評価規則について述べる。次にシステムの構成および動作を示す。さらに、評価がグラフの書き換えにより実現されることを述べる。パイプラインセグメント上で書き換え規則を実行する際の割付けを明らかにする。最後に簡単な関数例を挙げ、その効率的な実行方法を考察する。

２．ＦＰの評価規則

ＦＰシステムを実現するには、プログラム言語であるＦＰに対し、そのプログラムを実行する計算機システムを構築する必要がある。ＦＦＰ(Formal System for Functional Programming)はＦＰの形式的な意味を与えるシステムである。言い換えると実際のシステムにおいてはＦＰはＦＦＰに変換され、これがＦＦＰシステムに入力され、実行へ移される。

こうして、ＦＦＰ表現に対する評価機構を示すことで、ＦＰの意味を操作の形で与えることができる。評価機構は文献[3]に基づき以下のように示される。f は任意のＦＦＰ表現であり、同様にＦＦＰ表現である x に対して作用するものとする。作用対 f:x は評価関数により評価される。μ関数は意味関数でありＦＦＰ表現の評価を行う。ρ関数は表現関数であり、関数定義をその定義本体と関係させる働きを持つ。

$$\mu[f:x] = \begin{cases} f\text{がシステムの組込関数（原始関数）であるとき：} \\ \quad \to f\ \text{を}\ x\ \text{に作用した結果} \\ f\ \text{が}\perp\text{以外のアトムのとき：} \\ \quad \to \mu[\rho(f):x] \\ x = \perp\text{のとき：} \\ \quad \to \perp \end{cases}$$

$$\rho(f) = \begin{cases} f\ \text{が定義関数のとき：} \\ \quad \to \text{定義本体の置き換え} \\ \text{それ以外：} \\ \quad \to \perp \end{cases}$$

関数形式における表現関数のメタ結合規則は、関数列 $f_2,\ldots,f_n$ の、対象 x に対する作用の方法を与える。

メタ結合規則：

$$\rho(\langle f_1, f_2, \ldots, f_n\rangle):x = \rho(f_1):\langle\langle f_1, f_2, \ldots, f_n\rangle, x\rangle$$

３．システムの構成および動作

3.1 システムの構成

3.1.1 ハードウェア

本システムの構成を図１に示す。以下、図に従い、各ユニットを説明する。

構造体メモリ(以下ＳＭとする）は、簡約の対象であるグラフ（ＦＦＰ表現）を格納する。さらにリスト構造に対する効率的な処理を行うため、原始的なメモリ操作及び管理をＳＭ内で実行する。

プロセッサセグメント(以下ＰＳとする）は各々独立したプロセッサであるが、ＰＳ間のパスにより、全体としては各セグメントの機能を可変とした非同期のパイプラインシステムを構成する（以下これを実行パイプラインと呼ぶ）。ＰＳは与えられたグラフに対して対応する書き換え規則を適用し、簡約を行う。書き換え規則は各ＰＳに格納されている。

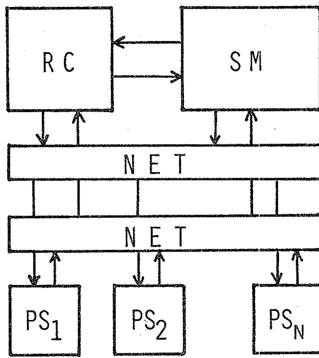

図1　本システムの構成

簡約コントローラ(以下RCとする)は、PSにおける関数形式(PFO)の簡約実行の結果として生成される作用対を管理する。作用対は、後述するようにグラフとして表現されるため、実際にはSMに存在する。RCには作用対の生成時に確保される作用ノードが登録される。作用ノードは作用対を構成する2つのFFP表現へのポインタをその要素とする。RCは登録された作用ノードを通じてSMへアクセスを行い、簡約可能な項を検出して実行パイプラインへ送出する。

本システムの特徴を以下にまとめる。

1．FPのプログラムとデータはシステム内でグラフにより表現される。
2．FPの評価機構は縦続接続されたプロセッサ内で実現され、その実行は並列に行われる。
3．処理は作用対の簡約要求で起動され、グラフの書き換えで簡約が進行する。

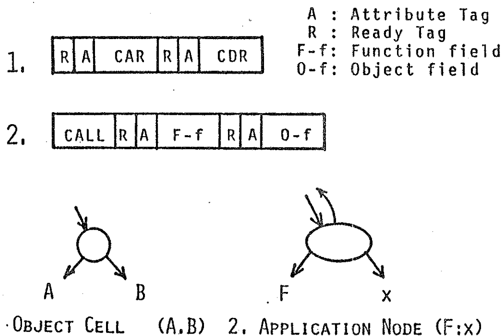

図2　セルの構成

3.1.2 FFPのグラフ表現

前述のように本システムでは、評価の対象であるFFP表現および作用対はグラフの形でシステム内に実現される。グラフの構成要素であるセルの構成を図2に示す。

第一に、FPの関数列とオブジェクト(列)は、通常の lisp と同様に通常のセル(以下オブジェクトセルと呼ぶ)からなる二進木の構造をもって表される。オブジェクトセルには、CAR・CDR 両フィールド共に値の到着を示す ready タグが付加される。 ready タグがオフの場合は、そのフィールドに対する参照は禁止され値が書きこまれるまで参照は待たされる。具体的にはSM内のウェイト・キューに参照要求が登録され、書き込みが行われた時点で参照が実行される。

次に、作用対を表現する作用ノードの構成について述べる。作用ノードは、3つのフィールドから構成されるセルで実現される。Fフィールドには、関数適用における関数表現を表すグラフ（FFP表現）へのポインタが格納される。同様に、Oフィールドには作用の対象となるオブジェクトへのポインタが格納される。CALLフィールドは、この作用ノードを指している親のセルを示している。

3.2 グラフの書き換え規則

3.2.1 書き換え規則と作用ノード

前述のように簡約化の操作とは、作用対 f:x に対する評価機構の実行である。本システムでは、簡約化の操作とは実行パイプラインにおける書き換え規則の実行を意味する。一方、本システムにおいて、作用対に相当するものは作用ノードである。つまり、作用ノードはシステム内での簡約項の存在を示すノードである。

システムに与えられた関数表現のグラフは静的なものである。関数表現は実行時に複数のオブジェクトに対して作用する可能性がある。言い換えると、関数表現は複数の作用ノードに共用される場合がある。従って簡約に伴う書き換え規則の実行で、関数表現にあたるグラフを直接書き換えることは出来ない。共用を可能とするため、本システムではグラフの形で与えられ

た関数表現をもとにして、実行時にオブジェクトと関数表現の対が展開される。この結果、新たな作用ノードを含むグラフが生成され評価が進められる。

一般に評価の対象となるＦＦＰ表現には複数の作用対が存在するが、簡約化は最内側の作用対に対して（もし複数存在する場合は並列に）実行される。

書き換え規則を大別すると、原始関数に対する規則とＰＦＯに対する規則に分類される。書き換え規則はシステムに用意された基本命令の組み合わせとして、ＰＳに内部記憶の形で蓄えられる。簡約項が送られてきた時点で、適切な規則が選ばれ、簡約が実行される。原始関数とＰＦＯの書き換え規則は簡約するという点では何等異なるところはないが、実現の操作においては次の点で異なる。

1. 原始関数の書き換え規則が作用対のＯフィールドのみを操作の対象とするのに対し、ＰＦＯを含む関数列の書き換え規則はＦ・Ｏ両フィールドのグラフが操作の対象となる。
2. ＰＦＯを含む関数列に対する書き換え規則の適用は、新たな作用ノードを生成する。

上の違いは、基本関数がオブジェクトをその引数とする普通の意味での関数であるのに比べて、ＰＦＯが関数（関数列）を引数とする汎関数であることから起こる。

3.2.2 作用ノードの生成と消滅

作用ノードについて、その生成、起動、実行、消滅の過程を以下に示す。

i）作用ノードの生成

簡約開始時に与えられたＦＦＰ表現は作用対であるから、単一の作用ノードを含むグラフで表現される。

これ以外の作用ノードは簡約の進行に伴い、ＰＦＯの評価における書き換え規則の実行により、作用ノードのプールからとられて動的に生成される。（規則は後述する。）

ii）作用ノードの起動

○簡約開始時に生成される作用ノード：

既にＦフィールド、Ｏフィールド共にready 状態にあり実行状態へ移行する。

○簡約途中に生成される作用ノード：

Ｆフィールド、Ｏフィールドが共にオブジェクトセルを指している場合には実行状態に移る。Ｆフィールド、Ｏフィールドの何れかが他の作用ノードを指している場合は、オブジェクトセルで構成されるグラフを指すまで、活性状態のまま待ちになる。

iii）作用ノードの実行

実行への条件が満たされた作用ノードは、起動コントローラによりパケット形式で実行パイプラインに送出される。パケットは、作用ノードに含まれる３つの要素（Ｆフィールド、Ｏフィールド、 ret アドレス）から構成される。ただし、retアドレスは親の作用ノードのアドレスである。

iv）作用ノードの消滅

実行パイプラインから新たに生成されたグラフ（簡約結果）のルートが返されると、作用ノードは返されたルートのアドレスを親のセルに書きこみ、その後解放される。解放された作用ノードのセルはフリーセルとなり、プールにもどされて次の作用ノード生成に備える。

3.3 書き換え規則による変換

3.3.1 原始関数

Backus の論文[3]において原始関数として挙げられているのは、構造データについての簡単な選択及び変換関数、算術関数、述語関数である。これらはシステムにおいて組み込み関数として実現される。すなわちシステムに用意されたＳＭに対する構造操作命令、ＰＳ内の演算命令の組み合わせによって原始関数は構成される。原始関数の組み込みは、関数の意味を損なうことがないよう行われなければならない。しかし原始関数が名前の通り primitive な操作であるため、個々の原始関数で高い並列性を見出すことは困難である。

このことから、ある程度高レベルの関数（例えば文献[3]の転置関数：transpose）については、直接に組み込み関数化することが有効と考えられる。特に定型的な操作を繰り返す場合にはこうした方法が有効と思われる。

3.3.2 ＰＦＯの変換

代表的なＰＦＯとして、composition（関数合成）と condition（条件判断）、そして apply-to-all（関数の並列作用）について、グラフの変換方法を示す。

i) composition

関数合成は次式で成立する関数である。

$(f \circ g):x = f:(g:x)$

$(f \circ g):x \leftrightarrow \langle\langle comp,f,g\rangle . x\rangle$

(FP)　　　(FFP)

実行パイプラインには作用対を構成するフィールドのポインタ値と結果を返す作用ノードのアドレスが送られる。ＰＳはこれらの情報から、メタ結合規則を適用する形でグラフを書き換える。書き換え規則は複数のＰＳにわたって分散実行される。具体的操作は各セグメントに内部記憶の形で蓄えられており、隣接するセグメントからの情報をもとに適切なシーケンスが実行される。

具体的な操作の内容を composition について考える。まず composition の意味は、次のように与えられる。

$apply \circ [2 \circ 1, apply \circ [3 \circ 1,2]]$

ここで apply:<f,x> は f:x を意味する。composition の意味記述をＦＰで与えることは超循環的であるが、システムの基本命令への展開が操作として明らかになる。すなわち apply:<f,x> は作用ノードの登録命令に相当し、選択関数 2,1,3 はＳＭ命令にあたる。図３にグラフの変換を示す。

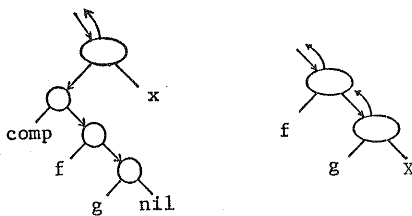

図3　composition でのグラフの変換

ii) condition

条件判断

if p(x) then f(x) else g(x)

に相当するＦＰ表現は次のように定義される。

$$(p \rightarrow f;g):x = (p:x)=T \rightarrow f:x;\ (p:x)=F \rightarrow g:x;\ \perp$$

$(p \rightarrow f;g):x \leftrightarrow \langle\langle cond,p,f,g\rangle . x\rangle$

(FP)　　　(FFP)

グラフの変換は図４に示されるように行われる。図で示されるように、p:x の結果が求まった時点で f , g いずれかの関数が選択されて、ルートの作用ノードに与えられる。cond' は組み込み関数で、引数がＴならば選択関数 1 を返し、Ｆならば選択関数 2 を返す。

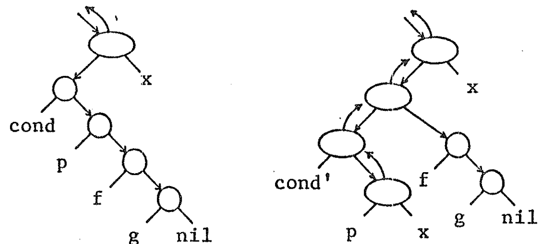

図4　condition でのグラフの変換

iii) apply_to_all

並列の関数作用を行う apply_to_all（以下 α とする）は下のように定義される。

$$(\alpha\ f):x = x=\phi \rightarrow \perp;\ x=\langle x_1,...,x_n\rangle \rightarrow \langle f:x_1,...,f:_n\rangle;\ \perp$$

$(\alpha\ f):x \leftrightarrow \langle\langle \alpha\ f\rangle . x\rangle$

(FP)　　　(FFP)

$x = \langle x_1,x_2,x_3\rangle$ についてグラフの変換を行ったものを図５に示す。

さらにパイプライン上での実現の概要を図６に示す。図６で用いられる英小文字は、図５のイタリック体で示されるセルのアドレスである。書き換え規則の実行がパイプラインセグメント上に展開され、作用対の生成がパイプライン的に行われることがわかる。

α の変換における特徴として、グラフのルートが作用ノードセルではなく、通常のオブジェクトセルになることがあげられる。生成された作用対 $f:x_1, f:x_2, f:x_3$ は、システムの資源が許すかぎり並列に実行される。作用対（α f）: x の結果を必要とする作用ノードはオブジェク

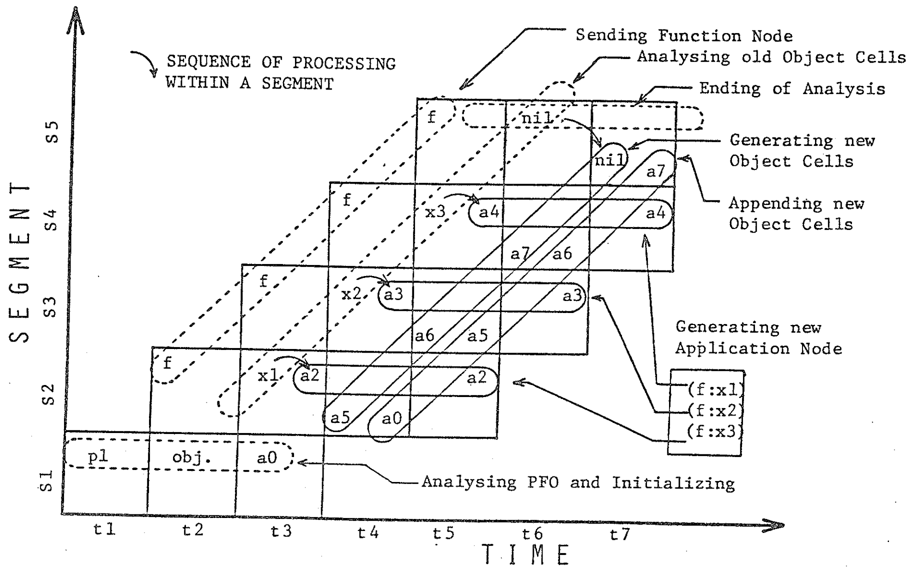

図6　apply-to-all の書き替え規則の実現

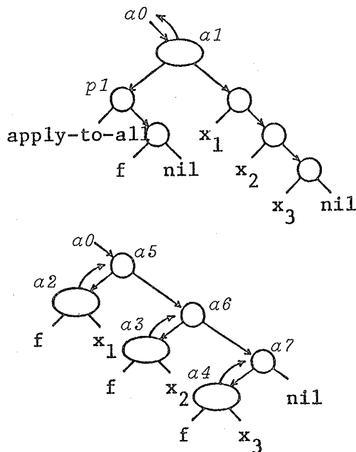

図5　apply-to-all でのグラフの変換

トセルの ready タグを参照して、生成された順に結果のリストをアクセスすることが可能である。この結果として関数の呼び出し関係において、関数の先行制御が可能となる。

4．検討

4.1 再帰関数と効率

本項では再帰関数実現の効率化について検討を行う。再帰関数を実行する場合、関数によっては関数本体の実行に要する時間よりも、関数呼び出しを実現するために要する時間のオーバヘッドの方が大きいと思われる。そこで3.3.1で述べたように、こうした再帰関数定義を直接組み込み関数とすることで処理の効率化が可能である。

再帰関数を組み込み関数として、一次元のパイプライン上に展開する場合の基本的な考え方は次の通りである。

第一に再帰関数の関数本体の処理は一つのPSで実行する。第二に再帰呼び出しが起こったとき、新たな作用ノードをシステムに要求せずに呼び出し側のPSで個別に作用ノードを生成し、隣接したPSに呼び出し時のデータを送り、そのまま関数本体を実行させる。このとき隣接したPSも同一の処理操作を行う内部記憶を持つものとする。

以上により、再帰構造は一次元のパイプライン上をPSに展開される。再帰関数におけるデータの流れは通常双方向となる。しかし先行

consを用いることが可能な場合には、再帰構造の流れを呼び出し方向に限定できる。

4.2 再帰構造と関数例

関数の論理的な結合関係に着目した場合の並列リスト処理の効果に関する報告がなされている[7]。それによると処理構造を、線形評価型構造と木評価型構造に大別することができる。

木評価型は関数を並列に適用する場合に現れる。生成起動される関数は木構造をなす。これを一次元のパイプライン上(2次元の時間空間)にマッピングするには、動的なマッピング・アルゴリズムが必要となる。従って組み込み関数とするよりも、作用ノードを用いた方法が望ましいと思われる。

本項では線形評価型構造を持った再帰関数間の呼び出し関係に注目して、簡単な関数例を挙げて組み込み関数定義の有効性について検討する。

線形評価型構造は関数を直列に起動することにより生じる。さらにこの構造は、呼び側の関数と呼ばれる側の関数の関係から、データ依存のある場合とない場合に分かれる。

i)データ依存のない例:union

和集合を求める関数 union のプログラムを図7に示す。

再帰関数memberは空間方向へ再帰を展開することが可能であり、図9ではeq-distrと/eqにより構成される部分にあたる。

unionは再帰的に定義され、オブジェクトの第二要素の先頭から順番に、第一要素とのmemberをとる。unionの起動は、論理的には、このmemberの終了を待つことなく次々と行なわれる。

```
union = null○2→1;
        member○[1,1○2]→union○[1,tl○2];
        apendr○[union○[1,tl○2],1○2]
member = (/or)○eq_distr
eq_distr = null○1→[ ];
           apendl○[equal○[1○1,2],
                   eq_distr○[tl○1,2]]
```

図7 関数 union のプログラム

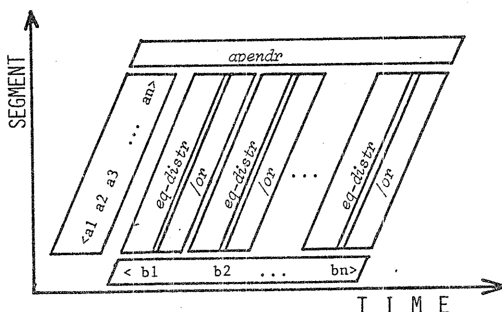

図8 関数 union のパイプライン上での実行

unionの再帰の各レベルについて作用の対象となるオブジェクトの第一要素は常に同一である。そこで、unionをパイプライン上で実行する際に図8のようにunionの1レベルの実行による履歴を利用する形でマッピングすることが可能である。

unionは時間軸方向に再帰を展開する形で実行される。unionの起動は作用ノードを用いて別々の空間で行うことも可能である。両者の優劣はunionの起動に要する時間とリスト<a1,a2,..,am>への参照が衝突することによるアクセスの遅延時間のトレードオフにより決定される。

ii)データ依存のある例:compact

関数 compact は与えられたオブジェクトに対し、removeを用いて重複する要素を削除して新しいオブジェクトを返す。 プログラムを図9に示す。

compactは再帰的に起動されるが、removeの生成するオブジェクトに対して作用する。compactをパイプライン上で実行した場合の時間空間図を図10に示す。大文字のイタリックで再帰関数の展開を示す。apendlは先行consと同様の機能をもつ。

```
compact = null→[ ];
              apendl○[1,compact○remove○[1,tl]]
remove = null○tl→[ ];
             equal○[1,1○2]→remove○[1,tl○2];
             apendl○[1○2,remove○[1,tl○2]]
```

図9 関数 compact のプログラム

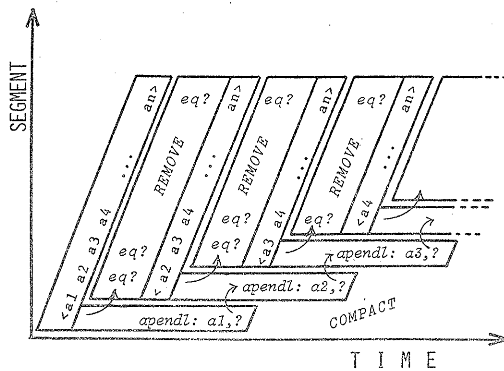

図１０　関数 compact のパイプライン上での実行

removeはセグメントに沿って再帰的に判断と削除を実行していく。unionとは異なりcompactの再帰もremove同様にセグメントの方向に展開される形となる。removeとcompactのあいだにデータの依存関係が存在するため、再帰的に起動されるcompactはremoveの生成するオブジェクトの系列を参照し実行を進める必要がある。

再帰関数の起動において、データの依存関係がある場合は、処理系がセグメントにオブジェクトを履歴の形で残し、これを引き続く再帰で利用する方法が有効である。この場合は、ＳＭへのアクセスが少ない分、組み込み関数定義が有効である。

５．まとめ

関数型言語によるリスト処理を指向したパイプライン処理システムの実行方式について報告した。まず関数型言語ＦＰの評価規則について説明した。そして、本システムではＦＰのプログラムとオブジェクトがリストの形で扱われることを示した。次に作用対に対し、並列評価を実現するための機構として作用ノードを導入し、評価がグラフの書き換えにより、行われることを示した。さらに評価機構が縦続接続されたプロセッサ上の書き換え規則によりパイプライン処理として実現されることを示した。最後に再帰関数についてその呼び出し関係に注目し、組み込み関数化の有効性について検討した。その結果、データの依存関係がある場合は組み込み関数化が効率的な実行手段として有効であることが明らかになった。

謝辞

日頃ご指導を頂く、重井研究室の遠藤昇氏に厚く感謝致します。

＜参考文献＞

(1) P.Henderson:"Functional Programming Application and Inplementation," Prentice-Hall International,1980.
(2) D.A.Turner:"A New Implementation Technique for Applicative Languages," Software practice and experience,Vol.9,No.1,pp.31-49(1979).
(3) J.Backus:"Can Programming be liberated from von Neumann style? A functional style and its algebra of programs",CACM,Vol.21,No.8,(1978),pp.613-641.
(4) 長谷川隆三、三上博英、雨宮真人："リスト処理向きデータフローマシンアーキテクチャの評価"，信学論(D),J67-D,9,pp.957-964.
(5) T.Nakamura,K.Sakai and Y.Mishina:"Function Level computing on the Brain Structured Computer," Proceedings of The IEEE Computer Society's Ninth International Computor Software & Application Conference 1985.
(6) 遠藤、中村、重井："汎用パイプライン処理における機能割付の最適化"，信学技報，EC82-38(1982-10).
(7) 長谷川隆三、雨宮真人："データフローマシンによる並列リスト処理"，信学論(D),J66-D,12,pp.1400-1407.