情報システム 53－5
（1995．3．22）

# 分散データ処理の新しいプログラミングパラダイムについて

住吉 政英　　石井 勉　　伊吹 公夫

東京工科大学

スター網の頂点に統括プロセッサを設けて、共通データを各プロセッサにコピーする方法で、分散処理の排他制御を簡単にできるが、プロセッサ数の増加に伴う負担を軽減する工夫が必要である。筆者らはデータ転送の局所性に注目し、この負担を軽減する階層網構成と、共通変数の網階層に対応させるＵｎｉｖｅｒｓａｌ　Ｔｒｅｅ技術を考案した。また、コピー完了待ちをチェックするＩＦ文の氾濫で、応用プログラム構造が複雑になるのを避けるため、Ａｃｔｉｖｅ指向という新しいプログラミングパラダイムを用いて、コピー完了の機能を分散ＯＳに集中することに成功した。その実現法を述べるとともに、３種の試作応用例で、これらの技術の有効性を確認したことを報告する。

# A new programming paradigm for distributed data processing

Masahide Sumiyoshi,　　Tsutomu Ishii,　　Kimio Ibuki

Tokyo Engineering University
1404-1 Katakura, Hachioji-shi, Tokyo 192, Japan

In a star type distributed processing system, with main processor which copies shared data in each sub-processors, increase of data transfer would make trouble through the expansion of the network size. Preventing from the trouble, we propose two technologies, named 'Universal Tree' and 'ACTive oriented programming-paradigm', cooperating with distributed operating system. A universal tree is composed of hierarchical universal variables corresponding to the network hierarchy levels of the distributed programs. On the distributed operating system, the ACTive oriented programming-paradigm would solve IF Statement hazard caused by data write complete check.

## １．はじめに

各プロセッサが対等に結合する無階層網の並列処理では、プロセッサの台数が増えると、共通データの同期・排他制御が複雑になる[1]。本論文で提案する手法を用い、スター網の中心に据えたプロセッサで共通データを整理統括すると、一貫した原理が使え、制御が簡単になる。スター網では、共通データが多くなると転送時間の増加で実時間性を損なう心配があるが、データ転送の局所性を満たすような配慮によりこの問題を解決できる。

同期や排他制御を行うデータはプロセッサ間で共通にアクセスするデータに限定され、付録例のようにデータ転送の局所性を満たすようなシステムも多い。また、スター型有階層の情報システムでは、データ転送が局所化できるようにプロセッサ相互を適切にまとめて網を構成し、この局所性を満たせる場合が多い。

共通データの扱いには、集中した共通メモリを用いる方法と、分散した共通部分を一斉に書き換える方法[2]とがある。本論文では、後者の方法を用いるスター型分散システムで、分散処理の管理から応用プログラマの負担を軽減する手法を述べる。このため、

１）Ｕｎｉｖｅｒｓａｌ　Ｔｒｅｅ

２）Ａｃｔｉｖｅ指向[3]

という２つの考えを織り込んだ分散プログラム環境と、これに対応した分散ＯＳの導入を提案している。

一つのプロセッサ内のプログラムを複数モジュールに分けて作成する際、ローカルとグローバルに変数を階層化しているが、ここで述べるＵｎｉｖｅｒｓａｌ　Ｔｒｅｅとは、この関係をスター型分散システムの各網階層に対応させて拡張したもので、その詳細を３章に述べる。

同期排他制御はスター網により比較的簡単になるものの、待ち合わせのＩＦ文処理が応用プログラマの負担として残る。筆者らの一人がＡｃｔｉｖｅ指向と名付けて発表した新しいプログラムパラダイムを利用し、これに対応した分散ＯＳを組み合わせると、この負担を軽減することができる。Ａｃｔｉｖｅ指向の詳細は文献３）に譲るが、４章で簡単に触れた上、５章で、これと組み合わせた分散ＯＳの説明をする。

また、本論文で述べる技術の有効性や、実時間性が満たされることを確認するため、４台のプロセッサからなるスター網を制御する分散通信ＯＳ「Ｂｅｎｎｅｔ」や、簡易分散プログラム環境「Ｍ．Ｓ－Ｅｄｉｔｏｒ」[4]と、付録に示す３つの応用例を用意し、学園祭見学者を対象に利用者調査を行った。見学者全員から実時間性に問題ないという結果を得た。少なくとも試作システムの範囲では、データ転送に通常のＲＳ－２３２Ｃ手順を用いた１階層のスター網でも有効性を充分確かめることができ、その結果を６章で報告する。

## ２．システム構成

図１にてＵｎｉｖｅｒｓａｌ　ＴｒｅｅとＡｃｔｉｖｅ指向を実現するためのシステム構成の説明をする。

ソースリストは試作言語として開発したＡＣＴ言語で記述し、実行はジェネレータによって中間コードに変換された後、Ａｃｔインタープリタにて動的に行われる。ここで、Ａｃｔインタープリタ開発の際、複数プロセッサに向けてのデータ転送の処理の作成の負担を軽減するために、通信部分を分けた。試作システムとしては、統合環境として、ソースリスト作成、中間コード変換（ジェネレータ）、Ａｃｔインタープリタを含めたＭ．Ｓ－Ｅｄｉｔｏｒを開発し、通信処理部分をＢｅｎｎｅｔとして開発した。

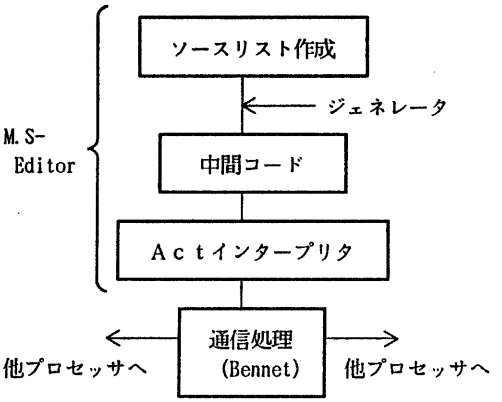



図１　システム構成

## ３．Ｕｎｉｖｅｒｓａｌ　Ｔｒｅｅ

同期や排他制御を行うデータはプロセッサ間で共通にアクセスするデータに限定されている。応用プログラマの、そのためのデータ通信の配慮を軽減するためにＵｎｉｖｅｒｓａｌ変数を提案する。この際、スター網の中心に据えたプロセッサで共通データを統括しているので、データ転送量が多くなると実時間性を損なう可能性がある。これを解決するためには、データ転送の局所性が有効に利用できるように、Ｕｎｉｖｅｒｓａｌ変数を階層化するとよい。このような、変数の階層の概念であるＵｎｉｖｅｒｓａｌ　Ｔｒｅｅを提案する。

### 3．1　Ｕｎｉｖｅｒｓａｌ変数の概念

従来の変数の概念には、図２のように１台のプロセッサ全体に有効な「Ｇｌｏｂａｌ変数」、１台のプロセッサ内で部分的に有効な「Ｌｏｃａｌ変数」というものがある。

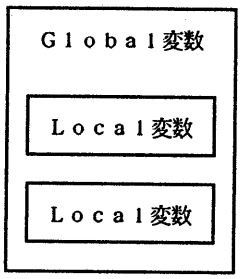



図２　Ｇｌｏｂａｌ変数とＬｏｃａｌ変数の関係

ここで図３のようにＧｌｏｂａｌ変数の１つ上の階層にあたる変数を提案する。その変数をここでＵｎｉｖｅｒｓａｌ変数と名づけ、全てのプロセッサに共通な変数を取り入れる事によって応用プログラマの負担を軽減する。ここで、共有変数の概念は単に複数プロセッサに共通な変数であるのに対して、Ｕｎｉｖｅｒｓａｌ変数が変数の階層化であることに注意されたい。後述するが、このことは、さらなる階層化を行うと、メインプロセッサの負担の問題の解決につながってくる。

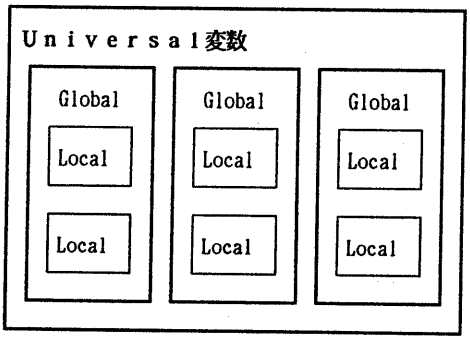



図３　Ｕｎｉｖｅｒｓａｌ変数

### 3．2　Ｕｎｉｖｅｒｓａｌ変数の実現

データ転送の局所性を満たすためには、共通データを分離し共通メモリに記憶する方法や、分散データの共通部分を一斉に書き換える方法などがある。本研究での手法は後者に該当する。図４のように、それぞれのプロセッサがＵｎｉｖｅｒｓａｌ変数をもち、変数の書き換え要求があったなら、メインプロセッサを介してすべての値をそろえるようにする。このようにすることにより、同時に発生した書換要求も順番に処理できるので、共通データの書換にともなう同期排他制御の問題が比較的簡単になる。

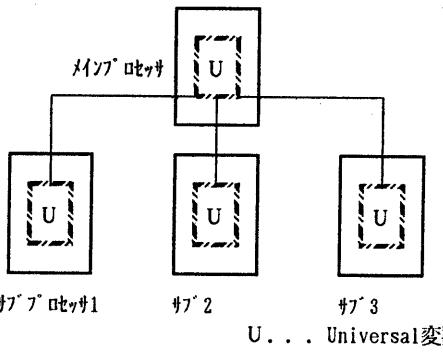



図４　Ｕｎｉｖｅｒｓａｌ変数の実現

### 3．3　Ｕｎｉｖｅｒｓａｌ　Ｔｒｅｅ

メインプロセッサを介して、全プロセッサのＵｎｉｖｅｒｓａｌ変数を書き換える手法は、プロセッサの台数が増えるとメインプロセッサの負担が大きくなってくる。これでは、実用的なシステムだとはいえない。そこで、実際の情報システムではプロセッサ相互を適切にまとめれば、データの局所性を満たすことが多い点に注意する。そのため、Ｕｎｉｖｅｒｓａｌ変数は単に複数プロセッサに共通な概念として捕らえず、データの局所性を利用し、スター型分散システムの各網階層に対応する階層化されたＵｎｉｖｅｒｓａｌ　Ｔｒｅｅとすることが有効だと考えられる。

### 3．4　Ｕｎｉｖｅｒｓａｌ変数の実行

Ｕｎｉｖｅｒｓａｌ変数において、ＡＣＴ言語ソースリストが中間コードに変換され、実行される過程を説明する。

字句解析、構文解析の後に、中間コード生成におい

て、Ｕｎｉｖｅｒｓａｌ変数の取扱いの準備をしなくてはならない。

（１）　中間コード生成

中間コード生成の段階において、Ｕｎｉｖｅｒｓａｌ変数は、定義された順に０，１，２，３・・・と番号を付ける。そして、番号が渡されたならその格納場所、変数の型がチェックできるようにテーブルを作成しておく。

Ｕｎｉｖｅｒｓａｌ変数の書換要求は、この変数番号と、関連数値、その他の情報を転送することによって、伝達される。受信したプロセッサは、これらの情報を判別し、テーブルを参照して、それぞれの処理を行うことになる。

（２）　実行（インタープリタ）

実行に際しては、いくつかの状態を考慮しなくてはならない。その状態を以下に記す。このとき、複数のプロセッサは図４のようにスター状に接続されており、その中心をメインプロセッサ、その他のプロセッサをサブプロセッサと呼ぶことにする。

１）メインプロセッサ上でUniversal変数書換要求
２）メインプロセッサ上でUniversal変数書換受信
３）サブプロセッサ上でUniversal変数書換要求
４）サブプロセッサ上でUniversal変数書換受信

ここで、要求とは、Ｕｎｉｖｅｒｓａｌ変数への書換命令実行を意味し、受信とは、他プロセッサから書換要求が来たことを意味する。それぞれの状態における要求、受信に対して、以下のような処理を行う。

１）値を書き換え、書換情報を送信
　　（全てのサブプロセッサに転送）
２）情報を受信し、値を書き換え、書換情報を送信
　　（同上）
３）書換情報を送信。（メインプロセッサに転送）
４）情報を受信し、値を書き換える。

つまり、Ｕｎｉｖｅｒｓａｌ変数書き換えは、必ずメインプロセッサを通してサブプロセッサへ要求される。こうすることによって、同時に発生した書換要求は処理の順番を決めることができ、従来の手法で問題であった排他制御の問題を比較的簡単にすることができる。

## ４．Ａｃｔｉｖｅ指向

### ４．１　分散処理における同期の問題

分散処理は、複数台のプロセッサで構成され、各プロセッサ内に処理を持ち、それらの処理が互いに絡み合い、１つの処理になってゆく。この様な分散プログラミングを従来の逐次型言語を用いて記述すると、同期を行うためのＩＦ文が多用され、構造が複雑になる。これを解決するためには、Ａｃｔｉｖｅ指向が有効である。

### ４．２　Ａｃｔｉｖｅ指向

従来法である逐次型言語を用いて分散プログラミングを行うなら図５のように、処理Ａは「もし処理Ａ要求がきた」なら実行し、次に処理Ｃは「もし処理Ｂが終了した」なら実行する、などと順序構造を含んでプログラミングしていた。これでは、プロセスの数が増えれば、分散プログラミング構造が複雑なものになるのは避けられない事実である。つまり、分散プログラミングにおいての処理の起動、終了などに関するＩＦ文は氾濫しやすく、複雑になりやすい。

そこで、図６のように処理に実行条件を付け、条件がそろったなら活動的に動かすという視点からプログラミングを行うと、分散プログラミングを構造化することが出来る。この、処理を活動的に動かすという指向が、Ａｃｔｉｖｅ指向である。

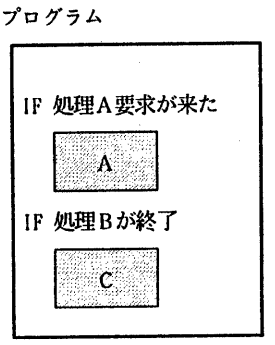



図５　逐次型言語による分散プログラミング

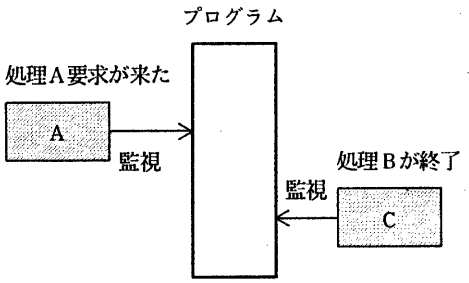



図6　Ａｃｔｉｖｅ指向による分散プログラミング

### ４．３　試作システムにおけるＡＣＴ命令の実現

Ａｃｔｉｖｅ指向による処理は、試作言語のプログラム上では図7のように記述する。実行条件が満たされると、４．４で説明するように、処理が活動的に動作される。これは、プログラム中のどこに記述してもよく、プログラムの最後にこのような処理をまとめておくと都合がよい。本誌試作システムでは、実行条件は変数（Ｕｎｉｖｅｒｓａｌ変数も含む）の値の条件の記述に限定してある。


図7　ＡＣＴ命令

### ４．４　ＡＣＴ命令の実行

ＡＣＴ命令が、ＡＣＴ言語ソースリストから中間コードに変換され、実行される過程を説明する。

（１）　中間コード生成

ソースリスト中に、ＡＣＴ命令があったなら、

・その後に記述されている条件
・その条件が成立したとき処理の開始場所

をテーブルに書き込む。

（２）　実行（インタープリタ）

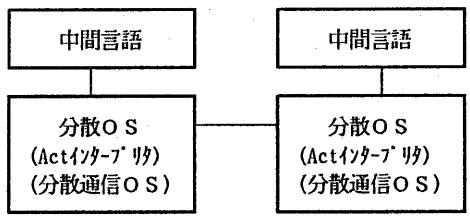
実行時、変数の値の書換があったなら、以下のような処理を行うようにインタープリタを作成しておく。変数はＵｎｉｖｅｒｓａｌ変数も含む。

１．中間コード作成時に表にした条件中に、書きかえた変数があるかチェック。
２．あったなら条件が成立したかチェック。
３．成立していたなら、現プログラムカウンタをスタックに積む。
４．中間コード作成時の表中の処理開始場所をプログラムカウンタに書き込む。
５．まだ、チェックしていない条件があるなら１．へ

注意すべき点は、変数の値の変更がある場所には、すべてにこの処理を行わせるようにする点である。

## ５．分散ＯＳ

Ｕｎｉｖｅｒｓａｌ変数により、共通データの書換にともなう排他制御の問題を簡単にした。しかし、データの書換を待つための同期の問題が残る。その問題を解決するためにＡｃｔｉｖｅ指向を利用した。つまり、Ｕｎｉｖｅｒｓａｌ変数の書換の完了に対して、処理を活動的に行わせることにより、同期排他制御の問題を簡単にした。本試作システムでは、図1のようにＡｃｔインタープリタと通信処理を分けて作成したが、Ｕｎｉｖｅｒｓａｌ変数とＡＣＴ命令を含めたＡｃｔインタープリタと、通信部分を含めて分散ＯＳとするのが今後の発展につながると考えられる。そのため、従来応用プログラマが負担していた排他制御と同期の処理を、図8のように分散ＯＳにまとめることが有効である。




分散ＯＳ．．．　Ｕｎｉｖｅｒｓａｌ　Ｔｒｅｅ、Ａｃｔｉｖｅ指向処理を用いて同期排他制御を行うオペレーティングシステム

図8　分散ＯＳ

## ６．試作例

Ｕｎｉｖｅｒｓａｌ　Ｔｒｅｅ、Ａｃｔｉｖｅ指向の有効性を確かめるために、試作言語としてＡＣＴ言語（ＡＣＴｉｖｅ指向言語）を開発し、その言語における分散プログラミング環境として、「Ｍ．Ｓ－Ｅｄｉｔｏｒ」を作成した。また、言語の作成において、複数プロセッサに向けてのデータ転送の処理の作成の負担を軽減するために、分散通信ＯＳ「Ｂｅｎｎｅｔ」を開発し、アプリケーションと分散通信処理部分との分離の有効性も確認した。

## ７．局所性

実時間性に影響を与えているかを調べるために、付録に述べる３つの応用サンプルプログラムを用い、学園祭見学者を対象に利用者調査のアンケートを行ったところ、実時間性に問題はないという結果を得、局所性を充分確かめることができた。

## ８．あとがき

複数プロセッサ間に共通な変数として、Ｕｎｉｖｅｒｓａｌ変数を提案し、それに伴う問題の解決としてＵｎｉｖｅｒｓａｌ　Ｔｒｅｅという概念を提案した。また、分散プログラミングにおけるＩＦ文の氾濫を指摘し、それを解決すべくＡｃｔｉｖｅ指向を提案した。本研究では、Ｕｎｉｖｅｒｓａｌ変数とＡｃｔｉｖｅ指向の有効性を確かめるために、試作システムとして、ＡＣＴ言語とそのプログラミングのための環境となるエディタ、ＡＣＴ言語作成を容易にするための分散通信ＯＳを開発した。

ここで、Ｕｎｉｖｅｒｓａｌ　Ｔｒｅｅの発想から、単一プログラム上で、Ｕｎｉｖｅｒｓａｌ　Ｔｒｅｅ、Ｇｌｏｂａｌ、Ｌｏｃａｌ変数定義を行い、それぞれのプロセッサ上の処理を記述するという手法にも発展することもできる。また、試作システムにおいて、ＡＣＴ命令に記述する実行条件は変数（Ｕｎｉｖｅｒｓａｌ変数も含む）の値の条件の記述しかできない。実用的なシステムにするためには、条件の記述を、より広い意味での条件にする必要がある。

今回提案した２つの概念は、このように様々な技術に発展する可能性を秘めたものである。現段階では、数多くの問題を含んでいるが、今後の研究により問題を解決してゆき、さらなる発展に導く必要がある。

## 謝辞

本論文に対して御討議頂いた工藤講師、越田助教授、千種助教授に、この場を借りて感謝の意を表します。

## 参考文献

1)宮村黙：並列処理言語，マグロウヒル
2)John L Hennessy & David A Patterson：Computer Architecture A Quantitative Approach, Morgan Kaufmann Publishers Inc. p. 467-469
3）住吉政英：Prologue Of ACT, I/O, 工学社, p. 44-53 (1993.10)
4）住吉政英：M.S-Editor, Computer Fan, 工学社 p. 48-57(1994.10)

## 付録

### １）Ｕｎｉｖｅｒｓａｌ変数の実現

Ｕｎｉｖｅｒｓａｌ変数の簡単なサンプル。
Ｕｎｉｖｅｒｓａｌ変数　Ｘ、Ｇｌｏｂａｌ変数Ｙの定義を行い、キーの入力があったなら、Ｘ，Ｙの値を＋１する。このとき、変数Ｙは各プロセッサ内のみ有効なので、他プロセッサに影響はない。しかし、変数Ｘは全プロセッサに有効なので、それに応じて変化する。

### ２）４台の画面を飛び跳ねるボール

通信、同期、の問題をほとんど考慮することなく、プログラミング出来る事を示したサンプル。４台並んだプロセッサの画面上をボールが駆け回る。

### ３）対戦！！！４台スロット「Ｔｅｎｎｅｔ」

最大４人のプレイヤーが、目標点を目指してスロットゲームを行う。他プレイヤーより先に目標点に達すれば勝ちで、その順位を競う対戦ゲーム。ゲーム中ある条件が満たされると、他プレイヤーのゲーム進行を妨害でき、この処理が実時間性に問題なく実行されたことからシステムの有効性が確認できた。