計算機アーキテクチャ 79－15
（１９８９．１１．２１）

手続きフロー型並列計算機の制御機構と並列記述言語

冨澤眞樹　須崎有康　五十嵐智　阿刀田央一　斎藤延男
東京農工大学工学部

筆者らは、多重の手続きフローを記述するための並列制御機構を提案している。本稿では、計算機モデルと制御モデルを示し、その並列制御機構の概要について述べる。手続きフロー型並列計算機は、機械語レベルで並列制御命令を持つプロセッサ、バススイッチ及びメモリユニットを積木細工のように構成した計算機である。本稿では、この手続きフロー型並列計算機とその基底言語である並列記述言語C//について述べる。本研究の並列計算機のアーキテクチャは、オーソドックスなものである。本研究の目的は、ハードウェア、制御方式、言語、応用問題のアルゴリズムの総合的な整合性をはかることであり、その全部を結ぶ接点として、制御方式および制御機構を出発点としている。

現在すでに、制御機構、制御命令に関しては実証、報告ずみであり、これに加えてすでに例外処理部の制御機構が設計を終えている。この制御機構を組み込んだハードウェアは、応用研究に耐える第３次のものの設計作業が進行している。並行して、制御機構におけるデータの受渡し機構、アドレッシングモード、領域取得法などの見直しを進めているが、これはCPU自体の改変が必要なため、現在実装の予定はない。言語についてはプリプロセッサを製作して使用しており、コンパイラを作成中である。現在、各種並列アルゴリズムの記述、第２次のハード上での実行の実験によって、アルゴリズムとの整合性を調べている。また、将来の課題として、バス構造とアルゴリズムにおける変数の局在性との整合問題がある。

The Control Mechanism and Parallel Programing Language for Procedure Flow Multiprocessors

Masaki Tomisawa　Kuniyasu Suzaki　Satoshi Igarashi　Oichi Atoda　Nobuo Saito

Faculty of Technology Tokyo University of Agriculture and Technology

2-24-16 Naka-machi, Koganei-shi, 184 Japan

We have proposed a parallel control mechanism based on multipleprocedure flow. In the begining, we summarize the parallel control mechanism. The procedure flow multiprocessor consists of processor elements that have parallel control instructions in hardware level, bus switches, and memory units as building blocks. In this paper, we describe the multiprocessor and parallel programing language called C//. One of our purposes is to provide parallel programing environment for studying parallel algorithm.

1. はじめに

ノイマン型プロセッサから構成される並列計算機では、手続き呼出しをプロセス生成に対応付けるのが一つの方策である。このことは、提案されている並列アルゴリズムや並列処理言語がプロセスを粒度としているものが多いことからも妥当なことと考えられよう。その場合、逐次型計算機の手続き呼出しがCALL命令に対応するように、並列計算機の場合にもそのような機械レベルでの制御機構を用意し、プロセッサ要素間で自由な手続き呼出しを許すことが逐次型計算機の素直な拡張であろう。しかしながら、この機構をOSに委ね、集中管理していたのでは実行効率の向上は望めず、プロセッサ単位での分散管理が必要となる。さらに、自由で一般性のあるプログラミングを許すためには、一つの手続きを多重に呼び出したり、あるいは一つの手続きが複数の手続きから多重に呼び出されるようなプログラミング構造にも対応できなければならない。

筆者らは、上記要件に対応する制御機構を提案し、試作機による検証を行った。[1] 本稿では、制御機構の概要、ハードウェア及び制御機構に基づく並列記述言語について述べる。

2. 分散型並列制御機構

並列実行のための制御命令を組み込んだプロセッサによって、並列計算機をビルディングブロック化することが、分散型並列制御機構の目的である。提案している制御機構は、ごく一般的な制御モデルに基づくものであり、また特別なハードウェアを前提としない。本章では、計算機モデルと制御モデルを示し、制御機構が持つべき機能について述べる。

2.1 計算機モデル

図1に示すように広いアドレス空間を分割して各プロセッサに割り当てる。プロセッサが原則として割り当てられた空間にあるプログラムを実行することにより、効率的な分散的制御が可能になる。また、プロセッサ数の増加によって生じるアクセス競合を回避できるように、計算機は局所参照性を利用できるような構成が望ましい。すなわち、PEと割り当てられるメモリ空間は近い距離に配置する。ページ#nにはPEnが低コストでアクセスできるように割り当て、他のPEはBSと結合網を通してアクセスできる。ページ#nの容量が少なければ、隣接するページ#n+1を使用し、PEn+1は使用しなければよい。このためPEに近いページへのアクセスコストは低く、遠いページへのアクセスコストは高くなるような結合網を想定する。

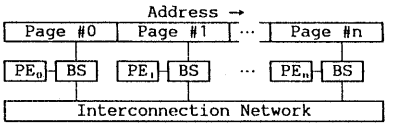


図1 計算機モデル

2.2 発注/納品モデル

プログラムは、並列実行の単位である手続きの集合からなる。プログラムの並列実行を記述するための基本制御命令として、DEMAND命令、ACCEPT命令、DELIVER命令の3つを用意した。これらの動作の概念を、図2に示す。

DEMAND命令は逐次計算機の手続き呼出しCALL命令に相当するが、制御がフォークするので呼出し側と呼ばれた側は並列に実行される。ここでDEMAND命令の動作を「発注」と考え、呼出し側を「発注側」、呼ばれた側を「受注側」と呼ぶ。受注側の手続きは

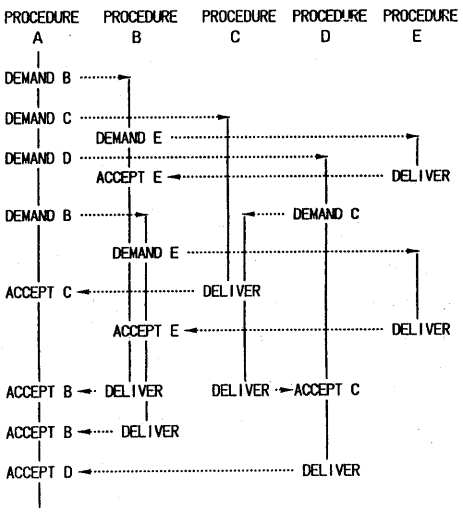


図2 発注／納品モデル

DELIVER 命令で終了することにより、発注側に対して「納品」する。発注側のACCEPT命令はDELIVER命令と対で使用され、発注した仕事を「受領」するもので、これによって制御がジョインする。このように3つの基本制御命令によって計算がすすめられる制御モデルを、発注／納品モデルと呼んでいる。発注／納品モデルでは次のような場合が一般的に起こる。

(1)一つの手続きが、時間的に隔離されず、複数の箇所から発注される。（多重受注）

(2)一つの手続きが、受注しない内に同一の手続きに発注する。（多重発注）

(1)については手続きはすべて再入可能とし、発注毎にプロセスを動的に生成することが必要である。この多重性を強調するために、手続きをテンプレート、それから生成されたプロセスをインスタンスと呼ぶ。(2)については、生成されたインスタンスを識別することが必要である。

2.3 制御機構の持つべき機能

プロセッサが持つべき並列制御機構は次のことを満足しなければならない。

(1)単純で一般的な制御モデルに基づき、必要十分な制御命令を持つこと。

(2)多数のプロセッサを利用できるように分散型であること。

(3)参照局所性を利用できること。

(4)ハードウェアレベルで組み込むために制御機構の規模が小さいこと。

発注／納品モデルを制御モデルとするとさらに、多重発注及び多重受注に対応するため次のことが必要になる。

(5)高速なプロセス生成、実行管理機能を持つこと。

(6)インスタンス間の引数の受渡し機構を持つこと。

3. 並列制御命令と制御構造

文献[1]で示した制御命令及び制御構造の概要について述べる。説明を簡単にするため、一つのプロセッサが一つのテンプレートに対応するものとする。実際には、一つのプロセッサに複数テンプレートを割り当てる。同一テンプレートを複数のプロセッサに割り当てることもある。これらの場合には、

(1)DEMAND命令でプロセッサとテンプレートの指定

(2)(1)に対応する受信点

(3)可変長の記憶管理

に対応するような制御構造をもたせる。

3.1 プロセッサヤード

各プロセッサに対して図3に示すようなプロセッサヤードと呼ぶ、データ構造が与えられる。テンプレート領域には、多重実行される再入可能なプログラムコードが置かれる。プロセッサヤードの先頭にあるホーム領域は、インスタンスの生成、実行、削除等を行う制御手続きによって使われる。ホーム領域内のデマンドキュー（Demand Queue、以下ＤＱと略記する）は、受注管理及び実行待ちや待機中のインスタンスを管理するために使われる。ＩＥＢ領域内で動的に取られたヒープは、ＩＥＢ(Instance Environment Block)と呼ぶインスタンスの実行環境として使われる。ＩＥＢは、その先頭アドレス（ＩＥＢアドレス）によって起動手続きが管理している。

3.2 受信点(Recipient:Rcp)

受信点は、発注側と受注側のインスタンスを動的に結合するものであり、図4に示すような同期用のフラグ類を含む構造である。発注側のインスタンスがDEMAND命令を実行すると、そのインスタンスに環境を与えているＩＥＢ内に、動的に受信点を設定する。そして、この受信点のアドレスを含む「要求レ

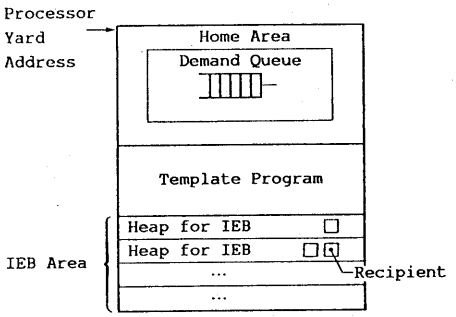
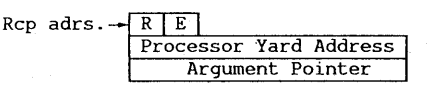


図3 プロセッサヤードの構造



Rcp:Recipient
R:Ready flag, E:Error flag

図4 受信点の構造

コード」を受注側のＤＱに積む。DEMAND命令は出力引数として、受信点アドレスを発注側のインスタンスに返す。受注側はインスタンスの実行を終了するとき、DELIVER命令によって受信点のＲフラグを立てる。発注側のACCEPT命令は、DEMAND命令との対応をとるため受信点アドレスを入力引数とし、Ｒフラグを見ることによって同期を取る。

３.３　基本制御命令

CALL命令がスタックによって実現されたように、並列制御命令は図５に示すようにデマンドキューと受信点によって実現される。発注側のDEMAND命令は、受注側のＤＱに要求レコードを積む。受注側のプロセッサの起動手続きは、ＤＱに要求があるとＩＥＢを割り当てインスタンスを生成し、テンプレートプログラムを実行する。テンプレートプログラムがDELIVER命令で実行を終了すると、DELIVER命令ではＤＱから要求レコードを取り除き、使用していたＩＥＢを解放する。つまり、DELIVER命令を実行したインスタンスは削除される。基本制御命令の動作を以下に示す。

(1)DEMAND命令

①ＩＥＢ内に、受信点を確保する。

②受信点のＲフラグ、Ｅフラグをクリヤし、ヤードアドレスと引数のポインタを設定する。

③受信点アドレスαを要求として、受注側のプロセッサ(図中ではプロセッサＢ)のＤＱに積む。

④ACCEPT命令と対応がとれるように、受信点アドレスαを返す。

(2)ACCEPT命令

受信点のアドレスαを引数として受注側のインスタンスとジョインする。応用プログラムからみたACCEPT命令は、単に同期をとるだけである。しかし、内部の手続きは同期待ちの状態になると、別のインスタンスを実行させようとする。ACCEPT手続きの動作は次のようになる。

①受信点のＲフラグがセットされているならば、②へ。さもなければ、そのインスタンスを待機状態にし、③へ。

②受信点を解放し、出力引数として引数ポインタの値を返す。そして、そのインスタンスの実行を続けさせる。

③起動手続きに制御を渡す。起動手続きは実行可能な別のインスタンスを探す。

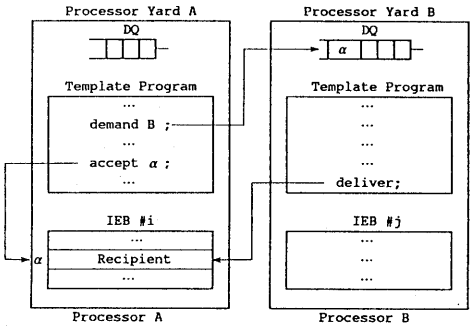


図５　基本制御命令の動作

(3)DELIVER命令

ＤＱの要求レコードから発注側の受信点アドレスαが分かるので、その受信点のＲフラグをセットする。デマンドしたままアクセプトしていないものがあるときは、Ｅフラグをセットし、これをリトラクト(3.5参照)する。ＤＱに積まれている要求レコードを取り除き、使用していたＩＥＢを解放する。そして、起動手続きに制御を渡す。

３.４　起動手続き

ＤＱ内の要求レコードに対して、インスタンスの生成、消滅、実行管理を行うのが起動手続きである。前述のように発注側の要求は、ＤＱを介して受け取られる。発注側の一つのDEMAND命令に対応するＤＱ内の要求レコードは、受信点アドレス、ＩＥＢアドレスとからなる。前者は発注側のDEMAND手続きによって与えられるが、後者はその時点では空である。受注側の起動手続きがこの要求に対してＩＥＢを与えたとき、すなわちインスタンスを生成したとき、起動手続きが管理情報としてＩＥＢアドレスを書き込む。ＤＱ内の要求レコードが除去されるのは、対応するインスタンスのデリヴァ時なので、ACCEPT命令よって実行を中断されたインスタンスに対応するものもＤＱに残っている。

起動手続きがＤＱから要求を選び出し、インスタンスを生成、実行する手順は次のようになる。

①ＤＱに要求レコードが積まれるまで待つ。積まれていれば②へ。

②ＤＱの先頭から要求レコードを調べる。このとき、もし、

(1)この要求にＩＥＢが割り当てられていなければ、ＩＥＢを割り当てインスタンスを生成、実行する。

(2)この要求に対するインスタンスがACCEPT命令の待機状態で、対応するＲフラグがまだクリアされていれば、読み飛ばす。セットされているものがあれば、そのインスタンスを実行する。

このように要求は、基本的にＤＱに積まれた順でインスタンスは生成、実行される。しかし、再帰呼出しや効率を考え、実行可能なものを選出し実行する。

３.５　並列制御命令

３.３で述べたDEMAND命令、ACCEPT命令、DELIVER命令は、並列手続きフローを記述するだけの基本制御命令であった。制御構造に対して応用プログラムが適切な操作をするための命令や並列処理に必要な相互排除及び非決定的選択のための命令も必要である。

(1)RETRACT命令

RETRACT命令は、DEMAND命令によって生成された受注側インスタンスを強制的に削除するもので、受信点アドレスを引数として、削除すべきインスタンスを指定する。削除されるインスタンスがDEMAND命令によってさらに発注を行っている場合、リトラクトは孫請けのインスタンスへと伝搬する。この伝搬には、ハードウェア割込みが使用される。

(2)SELECT命令

SELECT命令は生成されているインスタンスの受信点アドレスのリストの先頭アドレスとその受信点数を入力引数とし、デリヴァされた受信点アドレスのリストでの位置を出力引数とする。どれもデリヴァされていないときは、他のインスタンスに実行を切り替える。

(3)TASS(Test & Set/Switch)命令

入力引数は通常のテストアンドセット命令と変わらないが、その動作は常に成功する。すなわち、失敗した場合は他のインスタンスの実行に切り替え、起動手続きによって再び実行された時に再試行する。

３.６　引数の受渡し

発注側のプログラムと受注側のプログラムとの引数の受渡しは、DEMAND命令の引数ポインタによる参照渡しで行われる。受注側の起動手続きがインスタンスを生成したとき、このポインタは、受注側のＩＥＢの定位置にコピーされる。受注側のインスタンスはこのポインタをもとに、発注側からの引数を直接アクセスする。

４.　手続きフロー型並列計算機

標準化された構成要素により、使用目的に適合した計算機を利用者が構成することができる。３.で述べた制御命令を持つプロセッサからなる並列計算機を、手続きフロー型並列計算機と呼ぶ。

４.１　計算機構成要素

図１の相互結合網を、２入力１出力の並列バススイッチ(BS)により構成する。二つの入力バスは、BS内でそれぞれアドレスデコードされ、設定されたアドレス範囲内ならば、競合回路を通り出力される。バスサイクルは非同期バスサイクルであり、二つの入力が競合する場合は、入力バスサイクルにウエイトサイクルが入る。

プロセッサ要素(PE)は、割込み回路とハードウェア的にプロセッサＩＤと呼ぶ識別番号を認識することが必要である。前者は、リトラクトや例外処理に必要とされる。後者は、立ち上げ時のプロセッサヤードアドレスなどの各プロセッサに固有な情報をアクセスするために使用される。

メモリ要素(MU)は、一般的な回路で構成される。しかし、その容量は、プロセッサに割り当てられるプロセッサヤードの大きさ、プロセッサ数、アドレス空間、使用目的などにより決定される。

これらのBS、PE、MUの３つの構成要素により図６に示すように、応用プログラムにあった構成を採ることができる。

４.２　例外処理

並列計算機の例外処理に関しては、カーネルプロセッサと呼ぶプロセッサが集中管理する。並列計算機のための例外処理として立ち上げ処理、凍結例外処理、解析例外処理の３つがある。

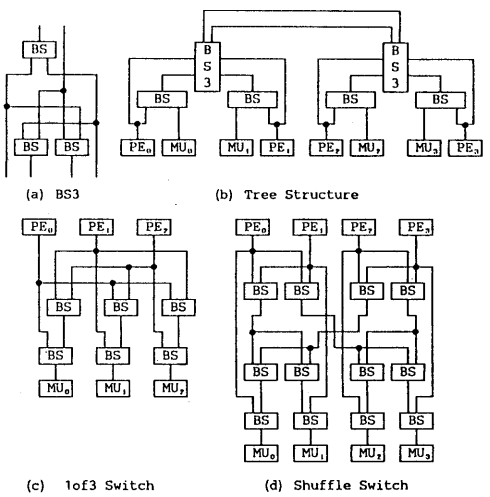


図６　構成例

(1)立ち上げ処理

カーネルプロセッサが複数のメンバプロセッサを管理する。プログラム実行は、カーネルプロセッサがプロセッサヤードをメモリにロードし、割込みを掛けることによって該当メンバプロセッサの起動手続きを実行させる。

(2)凍結例外処理

メンバプロセッサの例外発生は、すべてのプロセッサに割込みを起こさせる。他のメンバプロセッサは、実行状態を凍結し割込み待ちになる。カーネルプロセッサは、例外発生したメンバプロセッサのデマンドによりそのプロセッサを識別し、原因を調べる。また、この例外処理により、並列計算機でブレークポイントを使用したデバッグが可能になる。

(3)解析例外処理

逐次計算機のトレースに相当するような、デバッグのための例外処理も必要である。この例外処理は、特定の制御ワードの状態により、３．で述べた並列制御命令によって発生する。DEMAND命令やACCEPT命令では実行前に例外が発生することにより、トップダウンでデバッグするためのスタブテンプレートの作成が容易になる。TASS命令では実行後に発生し、ロックした状態で共有資源の操作ができる。

５．　並列記述言語C//

ここではＣ言語を基にし、関数呼び出しをインスタンスの生成に対応させた言語について述べる。以下、関数とはＣ言語での関数の意味で用いる。

５.１　前倒し呼び

関数のインスタンスは、呼出し時に生成され、結果を返すとともに消滅し、存在は一時的なので、動的な名前付け法が整合性がよい。本言語では、インスタンスタグと呼ぶ擬似的な変数によって、時間差と名前付けを同時に書く記述形式を用いる。このような形式を、ここでは「前倒し呼び(advance call)」と呼ぶ。本言語では、テンプレートに対応する並列関数と、並列関数内部で従来どおりに呼ばれる逐次関数がある。前者は、「template」と宣言して区別する。並列関数の呼出しと返り値の受取りは、以下のように書く。

```
template float pfunc();
tag t;
  …
t //= pfunc(x);
  …
y = //t;
```

「tag t;」によってtが「インスタンスタグ」(構造体タグと混同するおそれはないので、以下単にタグと呼ぶ）の名前であることを宣言する。タグは値を明示的に利用しないので変数ではないが、形式的に「tag型変数」と呼ぶ。「t //= pfunc(x)」によって前倒し呼びが行われて、関数pfunc()のインスタンスが作られ、制御が渡される。主プログラム側も制御を保持し、以後の任意の時点で１回だけ、式「y =//t」によって、このインスタンスの返り値を変数yに受け取る。この表記法によれば、

```
t[0] //= pfunc(x[0]);
t[1] //= pfunc(x[1]);
```

のように同じ関数の複数のインスタンスを併存させても、tag 型変数の名前によって区別されるので、

```
y[0] = //t[0];
y[1] = //t[1];
```

によって、結果を受け取る変数との対応を明示的に書くことができる。

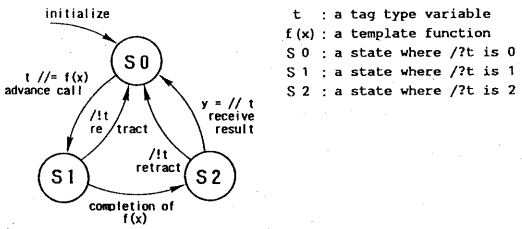


図7　tag型変数tの状態遷移

このように、文脈上、関数のインスタンスは作られると同時に動的に名前付けされたことになり、一方tag 型変数は、インスタンスの結果の受取証書のようにふるまう。手続きフロー型計算機の機械語レベルでは、インスタンスは「受信点アドレス」によって識別される。//= 演算子が実行されると、DEMAND命令（発注）の実行とともに、tag 型変数に受信点アドレスが暗黙に代入される。

前倒し呼びに伴う tag型変数の状態遷移を、図7に示す。tag型変数の状態は、/? 演算子によって陽に得られ、tag型変数tがそれぞれ図中のＳ０、Ｓ１、Ｓ２の状態にあるとき、式「/?t」は 0、1、2 の値をとる。Ｓ０はタグが使われていない状態、Ｓ１、Ｓ２は共に受取証書として有効な状態である。Ｓ２はインスタンス内での処理がすでに完了した状態（機械語レベルでデリヴァ済みの状態）であるが、言語上は /? 演算子による以外、Ｓ１と区別されない。Ｓ１とＳ２の区別は、機械の実行経過に依存したプログラムの記述上必要になる。受取証書は１回限り有効で、// 演算子によってインスタンスの結果を受取ると、タグはＳ０状態に戻る。また「/!t」を実行すると、tに対応するインスタンスはリトラクトされ、タグはＳ０状態に戻る。

５.２　コピー渡し

ユーザは、値渡しを使うのが原則である。値渡しは、逐次型言語と同様安全性が高い。アドレス渡しによる結果の受取りは、前倒し呼びでは副作用が起こる時刻を規定できないので、ユーザ側の注意を強いることになる。副作用の起こる時刻を論理的な一点に確定するために、コピーを仲介にする引数の渡し方を提供するのがよい。これにより、アドレス渡しを共有変数の参照以外、たとえば単に複数の値を返すだけの目的で使用する必要性は減少する。また、受注側で誤りが生じたとき発注側に及ぶのを避けることができる。

本言語では、Ｃ言語と同じ値渡しとアドレス渡しのほかに、並列関数に対して「コピー渡し」をサポートする。呼出し側で

```
t //= func(a,&b,$c);
  …
//t;
```

と書いたとき、aは値渡し、&bはアドレス渡し、$cはコピー渡しを意味する。コピー渡しでは、前倒し呼びをするときに呼出し側の環境内に一時的なコピーフレームが作られ、インスタンスはこの中に値を返し、//が実行されたときコピーがもとの変数に逆代入される。

呼び出される側では、例えば、

```
template int func(u,v,w)
int u,*v,w;
{ …
}
```

というように書かれる。この中でwの値を変更していれば、結果は「//t」と同時刻に、cに及ぶ。

５.３　共有変数のロック機構

この言語では、keyhole型変数と、この変数を用いるlock文を用意する。lock文は

```
lock (k; l) ＜文＞
```

のように書かれ、k、lはkeyhole型変数である。kは＜文＞の実行前にロックされ、lは＜文＞の実行後にアンロックされる。すでに kがロックされているときは、もちろん他からアンロックされるまで待つ。lock文は物理的には TASS命令に翻訳されるので、インスタンスの実行切り替えが起きる(3.5参照)。

keyhole型変数のスコープは通常の変数と同じである。keyhole型変数は、「アンロック状態」「ロック状態」の２状態をとり、前者は0とする。一方ロック状態は、これを判定するビットが機械によって異なるので、プログラム中関係式では!0、代入式では¯0とするのがよい。

５.４　関数の非決定的選択

いくつかのインスタンスの中から任意の一つを選ぶには、tag型変数をもって以下のように書く。

```
among (i; i < MAX; t[i]);
```

t[j]は添字つきのタグ型変数である。among文の中では、暗黙にiを0から1ずつMAX個スキャンし、タグの羅列を構成する。tag型変数は値を明示的に用いず、通常の左辺値としての代入もできない。しかし外部の写像（例えば配列 f[i]）を参照して、t[i]のかわりにt [f[i]] とすれば、インクリメントが1に固定されても、任意の羅列を構成できる。

上記の文例を実行すると、iに1つ選択された値を持って、文から制御が下に抜ける。among文は物理的には SELECT命令に翻訳されるため、選択できなきときはインスタンスの実行切り替えが起こる(3.5参照)。選択されたインスタンスの結果を受け取る時は、

```
//t[i] ;
```

と書けばよい。

5.5 記憶領域の性格と記憶クラス

発注／納品型並列計算機では、インスタンス毎にヒープが与えられ、それぞれこの中にスタックが作られる。すなわちこの計算機のメモリは、共用領域、テンプレートのコードに付随した領域、インスタンスのスタック上の領域の3通りの性格で使われる。本言語の自動変数はスタック上にとられるのを原則とし、逐次型言語の場合と変わらない。また外部静的変数はインスタンシエートせず、同一ファイルに属する関数（並列、逐次とも）間の共有変数として利用するのが自然である。また逐次計算機で手続に固定されていた内部静的変数は、論理的にテンプレートに付属すると考える。同一テンプレートから派生したインスタンスは、内部静的変数を共有する。初期化についてはC言語と同様に考えてよい。内部静的変数は、表や文字などの定数や、不要な再計算を避けるための並列関数の履歴として使用される。

5.6 ロード条件ファイル

3．で述べたように、各プロセッサにはプロセッサヤードが与えられる。一つのヤードには通常複数の関数テンプレートが置かれ、それらの関数のインスタンスが使うヒープが取られる。人海戦術処理のように、同一関数のインスタンスが物理的に並列実行される必要があるときは、幾つかのプロセッサヤード内に同一関数テンプレートのコピー（クローン）を分配する。プロセッサヤードへの関数テンプレートの配分などの機械依存部は、クローン数なども含めて、プログラム記述とは別の、ロード条件ファイルを設け、これに記述する。

並列関数、逐次関数ともに分割コンパイル可能であるが、別ヤードの並列関数から呼ばれる逐次関数はコピーして、ロード時にヤード毎にロードモジュールを再構成しなければならない。

6．おわりに

制御機構に関しては、複数テンプレートを一つのプロセッサに受け持たせる方式、インスタンスの実行環境管理方式などの定量的な評価が必要である。特にインスタンスの実行スケジューリング、環境管理、例外処理によるデバッグ方式などは、試作機上で実験評価されている。4．で述べた計算機の構成要素は、CPUに68000を使用した試作機の使用実績をもとに、現在新しいプロセッサボードを製作中である。並列記述言語C//はコンパイラを作成中であり、並列プログラム構造、並列関数と逐次関数のリンク方式などの検討をしている。プログラムの論理構造、参照の時空間的局所性、バス構成と、関数テンプレートの配置との関係が実行効率におよぼす影響は、現在明かではない。プログラムのソースファイルとバス構造を所与とし、ロード条件ファイルの最適化問題として研究する。

参考文献

[1]冨澤ほか：“手続きフロー並列計算機における分散組込み制御機構”，信学論(D)， J71-D，10，PP.501-538(昭63-10).