# 2進木構造並列処理システムCoralのプロトタイプとそのOS

Prototype of Binary Tree Multiprocessor CORAL and Operating System

若林直樹 Naoki WAKABAYASHI
信友義弘 Yoshihiro NOBUTOMO
近藤喜久馬 Kikuma KONDO
高橋義造 Yoshizo TAKAHASHI

徳島大学 工学部

Faculty of Engineering,University of TOKUSHIMA

## 1. はじめに

数百、数千のプロセサをもちいて超高性能な高次並列処理計算機システムを構成しようとすると、従来から用いられた共通メモリ方式や共通バス方式では、その実現が困難である。

高次並列処理システムのためのアーキテクチャは、すべてのプロセサを結合したシステムが理想であるが、これはハードウェア的に実現不可能である。そこで考えられるものは、繰返し構造をもつ並列処理システムである。繰返し構造をもつシステムとして星状結合，鎖状結合，環状結合，格子状結合，木状結合などの構造が考えられる。

鎖状結合構成としてTOPSTAR[1]があるが，共有メモリのポート数が有限であるとすると，プロセサ台数が非常に多い場合，プロセサ間の平均距離は大きくなりプロセサ間の通信に不利になる。

環状結合構成として環状結合形超多重プロセサシステム[2]があるが，このシステムは先のTOPSTARを環状に結合したもので、やはりプロセサ台数が非常に多い場合，プロセサ間の平均距離は大きくなり，プロセサ間通信は不利である。

木状結合構成として，CM*[3], SMS-201[4], DDM1[5] snowflake[6]が発表されている。CM*はクラスタによって結合されているが，プロセサ台数を非常に多く用いることはできない。SMS-201は、1レベルの木構造とみなされ星状結合と同様であり，プロセサ台数が非常に多くなった場合の結合は、ハードウェア的に困難である。DDM1は、インストラクションレベルのdata-flowシステムであり，分散処理向きでない。snow-flakeは，広義の木構造をしており信頼性も高いため，実現されれば優れたシステムになると思われる。

木構造のアーキテクチャの特徴として

(1)リカーシブ構造
(2)階層構造
(3)インターフェイスの規模が小さい
(4)各プロセサは小規模構成

などがあげられる。これらの特徴により，単純なハードウェアモジュールの繰返しによって木構造が構成できる。

次に，木構造としてどのような構造が優れているかについて示す。

木の枝が一つのノードからK本出ているK進木について考えてみると，K進木構造のプロセサ台数(N)は，(1.1)式によって示すことができる。

$$N=\frac{K^{\ell+1}-1}{K-1} \qquad \ell:\text{木のレベル} \qquad (1\cdot1)$$

また，K進木構造のプロセサ間の平均距離(MPL)は、(1.2)式によって示すことができる。

$$MPL=\frac{2\ell K^{\ell}(K^{\ell+1}-1)}{(K^{\ell+1}-1)(K^{\ell}-1)}-\frac{4K^{\ell+1}}{(K^{\ell+1}-1)(K-1)} \qquad (1.2)$$

プロセサ台数と平均距離についてグラフにしてみると，図1となる。このグラフより多進木ほど平均距離は短いと分かる。ところが多進木になると，各ノードでのアクセス競合が多くなり，性能の劣化がおこると考えられる。以上のこと

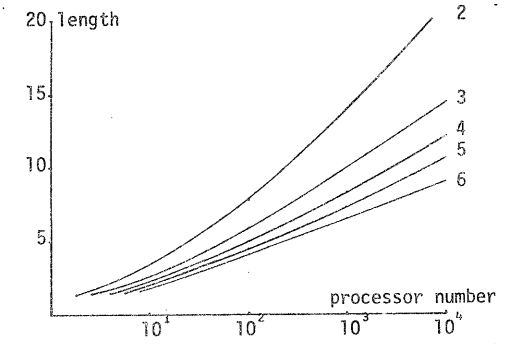



図1．多進木のプロセサ間の距離

より木構造として，2～5進木が最適であると考えられる。

そこでデータの分配，回収やプロセサ間距離そしてハードウェアについて検討すると，2進木構造が，分散型高次並列処理システムとして望ましい諸性質をもつことが分かった。さらに木構造は，信頼性に問題があると言われるが，信頼性におよぼす影響の大きいプロセサの数は，その分だけ少いので，一概に信頼性が劣ると断定するわけにはいかない。

2進木構造はVLSI化にも適応すると考えられ，図2のように幾何学的に単純なパターンとして構成される。

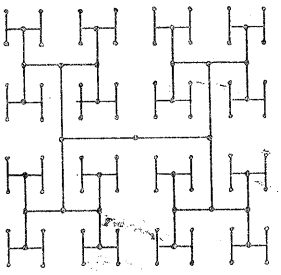


図2．2進木構造

2進木構造の計算機システムは，他にも提案されているが，我々は2進木構造をもつ高次並列処理計算機システムをCoralと名づけた。[7]

Coralは，図3のように構成される。各ノードは，エレメントプロセサであり，プロセサとメモリより構成される。また，各ブランチは，双方向のデータバスで構成される。ここでRPはRoot Processor，NPはNode Processor，LPはLeaf Processorとする。

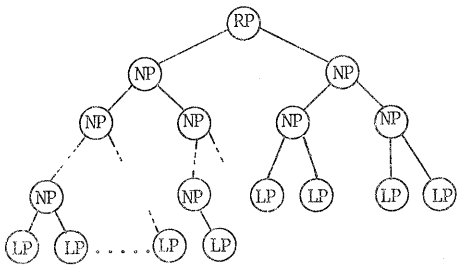



図3．Coralシステム

今回，Coralプロトタイプシステムを開発したので報告する。

プロトタイプは，次のような目的に使用するために開発したものである。

(1) 並列プログラムの開発
(2) 並列処理の効率の測定
(3) プロセサ間通信方式の実験
(4) 分散型OSの開発
(5) 信頼性のデータをとる
(6) 現在稼動中の並列処理システムとの実用性の比較
(7) 各種のアプリケーションの開発

## 2. Coralプロトタイプのハードウェア

Coralプロトタイプを開発する基本方針として，安く，早く，動くものを作ることであったため，cpuボードとしてワンボードマイコンを用い，各種インターフェイスにLSIを用いた。これらを用いたことによる性能低下は予測でき，システムのパラメータとして扱うことができるので問題はないと思われる。

次に，Coralプロトタイプの構成について述べる。

### 2.1 結合方法

#### 2.1.1 プロセサ間結合

各エレメントプロセサを結合する方法として次の3種類が考えられる。

(1) I/Oインターフェイス方式
(2) 共有メモリ方式
(3) DMA方式

これらの方式の特徴を考えると，共有メモリ方式やDMA方式は，転送速度が速く，特にDMA方

式は分散型システムに適していると考えられる。一方、I/Oインターフェイス方式は、低速であるが，ハードウェアが単純である。

プロトタイプでの結合は，ハードウェアの簡単さと柔軟性のために，プロセサ間の結合に，I/Oインターフェイスを用いることにし，インテルの8080ファミリチップの一つである i8255 (Programmable Peripheral Interface：PPI) を使用した。

このPPIは，3つのI/Oポートを持っており，プログラムによってそのポートの入出力を決定する。入出力動作は，プログラムモードでポートを介して行なう。

PPIを用いたプロセサ間結合を図4に示す。ポートAと他方のポートBを結線し，ポートC上位4bitと他方のポートC下位4bitを結線している。

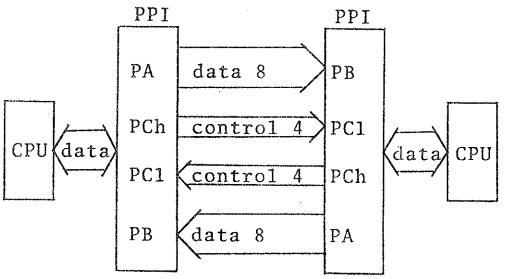



図4．PPIによる結合

2.1.2 データ転送

次に，PPIを用いたデータ転送の方法を述べる。

PPIのプログラムは，ポートAとポートC上位を出力，ポートBとポートC下位を入力と設定している。このように設定したPPIのポートAとポートBの間でデータを転送し，ポートC上位と下位の間で制御信号を転送する。制御信号には，次のものがある。

standby信号 ＝ データの出力完了の信号
acknowrage信号 ＝ データの入力完了の信号

この制御信号を用いたデータ転送のタイムチャートを図5に示す。

また，このデータ転送のアルゴリズムを次に示す。

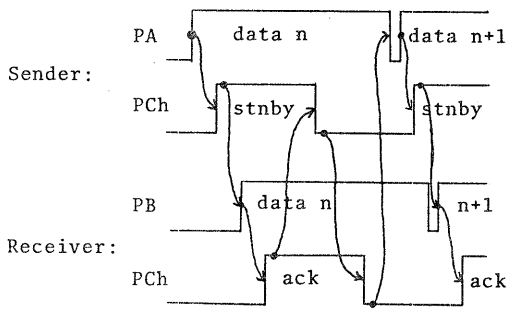



図5．データ転送のタイムチャート

```
/*Data Transmission*/
parbegin
   sender:  begin
                  put Data;
                  stnby:=true;
                  repeat get ack
                  until  ack=true;
                  stnby:=false
            end
   receiver:begin
                  repeat get stnby
                  until  stnby=true;
                  get Data;
                  ack:=true;
                  repeat get stnby
                  until stnby=false;
                  ack:=false
            end
parend.
```

この方法によって，プログラムモードでのデータ転送が可能となる。また，結線の対称性によって，双方向のデータ転送が可能となる。

次に，プロセサ間で通信を行なう場合，ルーティング機能が必要となる。そこで，各エレメントプロセサに図6に示すように番地をつけ，RPから1，NPを2,3…と割付ける。

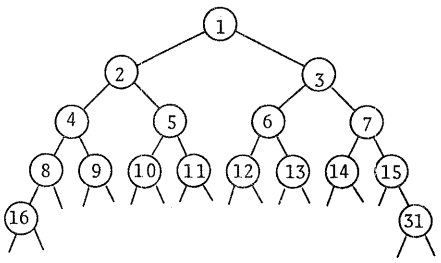



図6．番地付け

この番地付けには、アドレスiのleft-sonは、2iであり，right-sonは，2i+1である関係

がある。この関係を利用したルーティングアルゴリズムを次に示す。

```
/*Routing*/
begin
    a:=destination-address;
    s:=own-address;
    left:=s*2; right:=left+1;
    T:=false; L:=false; R:=false; S:=false;
    if a=s then s:=true;
    repeat
       case a of
          left : L:=true;
          right: R:=true;
          else if a  s
             then T:=true
             else a:=a div 2
       end
    until T or L or R or S=true
end.
```

このアルゴリズムによって、データを起点ノードから目的ノードまで転送することが可能となる。

## 2.2 プロセサ間の同期

Coralのような分散型システムにおいては、各エレメントプロセサは、非同期に動作している。このため、プロセサ間通信を行なう場合、エレメントプロセサ間で同期をとる必要が出てくる。そこで、同期をとるために割込みを用いた。

プロトタイプでは、割込制御をハードウェアの簡単化と柔軟性のために、i8259 (Programmable Interrupt Controller ; PIC)を用いている。

PICの機能は以下のようになっている。外部割込要求信号が8つ入力でき、各入力信号に対して優先順位があり、割込み競合はおこらずCPUに割込む。割込んだ後、割込処理ルーチンに処理を移すためのベクトル・アドレスを自動的にCallする。このベクトル・アドレスは、PICに対する初期化によって設定される。

プロトタイプでのPICの構成を図7に示す。

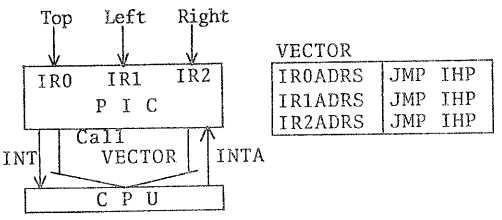



図7. PICの構成

一つのエレメントプロセサに対して、Top方向 Right方向、Left方向の3方向からの割込要求信号があり、それぞれIR2～IR0の順に優先度をつけて入力される。各方向からの割込要求に対して割込処理プログラムへのベクトル・アドレスが用意されている。このベクトル・アドレスに従って割込処理ルーチンに処理が移る。本論文で述べるOSにおいては、このベクトル・アドレスにIHP(Interrupt Handling Program)のアドレスがはいっている。

## 2.3 システム構成

以上述べてきたPPIとPICを用いることで、非同期なエレメントプロセサ間で同期をとってデータ転送を行なうことを可能とした。

次に、各エレメントプロセサ間の結合を図8に示す。

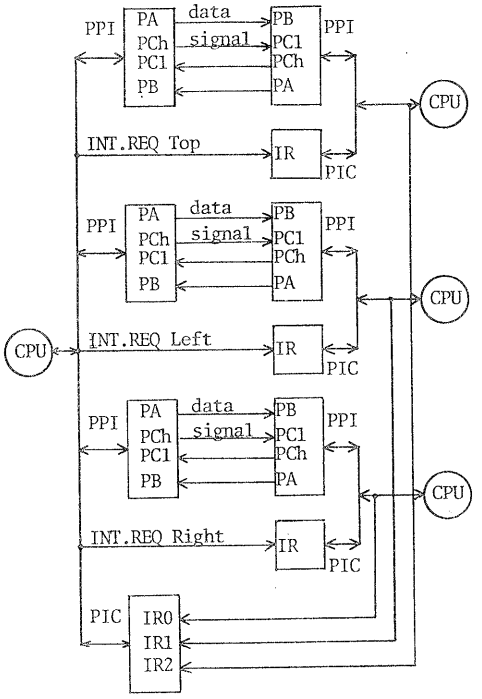



図8. エレメントプロセサ間の結合

そして、Coralプロトタイプのシステム構成図を図9に示す。RPは、SORD社のM223IIを用いている。これは、Z-80 CPU, 64KBメモリ, 700 KBのフロッピーディスク, デジタルカセット, プ

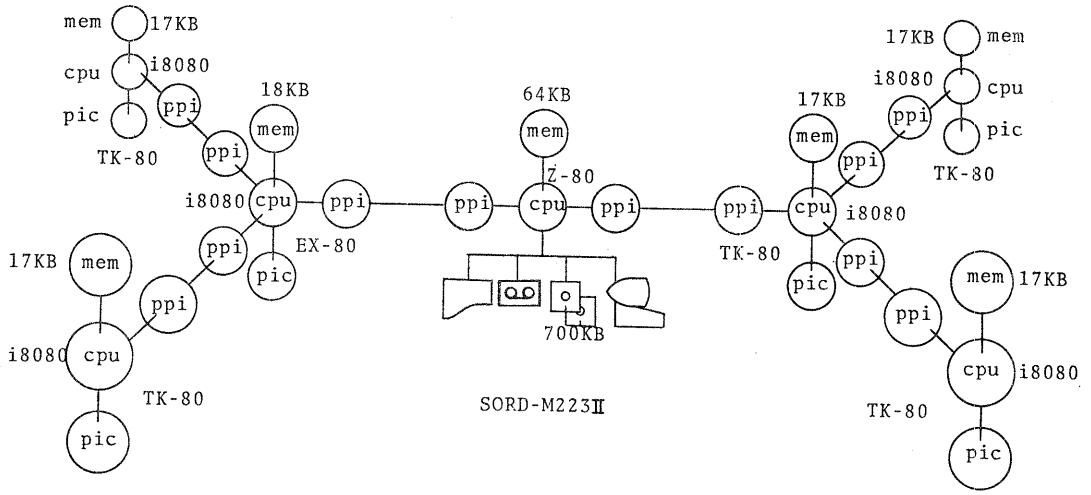



図9．Coral プロトタイプのシステム構成図

リンタ，CRT，キーボードより構成されている。NPは、東芝のEX-80，他のノードはNECのTK-80を用いている。それぞれi8080 cpu，TK-80は17KB，EX-80は18KBのメモリ，LED，テンキーから構成されている。

このようにプロトタイプのハードウェアは、heterogeneous であるが，システムの開発のため，RPに十分の機能を持たせている。

Coralプロトタイプの概観を図10に示す。NP，LPは，LEDやテンキーを扱いやすくするためにボードが垂直に取付けられている。

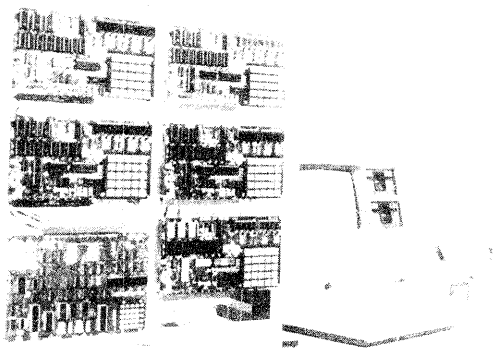


図10．Coralプロトタイプの概観

## 3. プロトタイプの動作試験

### 3.1 データ転送試験

プロトタイプは、異種（3種類）のプロセサより構成されている。そこで，任意のプロセサ間において割込みを用いたデータ転送試験を行なった。

255B，143B，15Bの3通りのデータ転送を255回繰返すのに要する時間を計り，最小2乗法を用いて結果の検討を行なった。

(1) 隣接プロセサ間データ転送

転送時間とデータ数の関係を1回当りに換算して図11に示す。

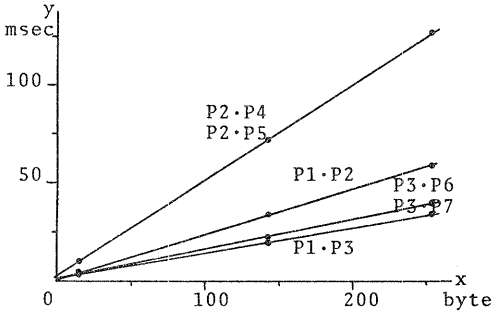



図11．隣接プロセサ間データ転送時間

この測定した3点に対して最小2乗法を用いて次の関係式を得た。

i) P1・P2間　　$y = 0.23x + 0.86$

ii) P1・P3間　　$y = 0.13x + 1.12$

iii) P2・P4間，P2・P5間　　$y = 0.49x + 1.67$

iv) P3-P6間，P3-P7間　$y = 0.15x + 0.88$

但し，$y$：転送時間（msec）
$x$：転送データのバイト数

なお，転送データ数を1Bで示すので，一度に転送できるデータ数は，最大255Bである。

上式においてxの係数は，1Bのデータ転送に要する時間である。i8080Aでは，sendおよびreceiveルーチンのループなしの1データ当りの実行時間は，0.06msecであり，send側とreceive側がポートCのコントロール信号を介してループによるwaitをはさみながら交互に動作しているので，$0.06 \times 2 + \alpha$の期待値が得られ，この試験値の妥当性がわかる。P2(EX-80)は，単位時間あたりのホールド要求回回数がハードウェア設計上(TVインターフェイス)多くなっているので，単位時間あたりのDMA回数は多くなり，転送時間は遅いと考えられる。P1(Sord)は，クロックが遅いので転送時間も遠い。一方，定数部分は割込みによるレジスタの退避，PICのコマンドセット，転送データアドレスや転送データ数のセットに要する時間である。以上の結果を図12に示す。

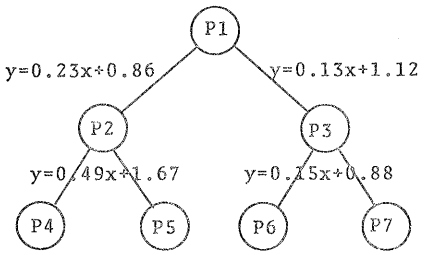



図12、隣接プロセサ間データ転送時間

(2) LP間往復データ転送

プロトタイプ上で重要なプロセスを実行するLP間のデータ転送を行なった。(1)と同様にして次の関係式を得た。

i) P6→P3→P7→P3→P6
$y = 0.58x + 1.09$　(msec)

ii) P5→P2→P1→P3→P7→P3→P1→P2→P5
$y = 1.98x + 6.08$　(msec)

まず，(1)の結果よりこの往復転送時間を求めるとi) $y = 0.60x + 3.52$，ii) $y = 2.00x + 9.06$となる。

xの係数は，(1)の結果とほぼ一致する。一方，定数部分は，(1)の結果より小さい。これは中継を行なうプロセサではreceiveとsendを連続して行なうためであり，ルーティングを行なうと，この定数部分がかなり大きくなると考えられる。

次に，データ転送をDMA方式で行なう場合について考えてみると

RP(2.5MHz)では，$y = 1.6x + 116$ (μsec)
他のプロセサ(2MHz)では，$y = 2x + 145$ (μsec)

となると思われる。

以上のことより，DMA転送がI/Oインターフェイスによる転送より約80倍高速であるとわかった。

3.2　1次元熱伝導方程式の並列処理実験

1次元熱伝導方程式は次のように示される。

$$\frac{\partial U(X,t)}{\partial t} = C^2 \frac{\partial^2 U(X,t)}{\partial t^2} \qquad (3.1)$$

$(0 \leq X \leq a)$

$U(X,0) = f(X)$

$U(0,t) = f(0)$，$U(a,t) = f(a)$

ここで

$$r = \frac{KC^2}{h^2} = \frac{1}{6}, \quad h = \frac{a}{M} \qquad (3.2)$$

$$U_n(m) = U(mh, nk) \qquad (3.3)$$

とおくと，次の差分方程式が得られる。

$$U_{n+1}(m) = \frac{1}{6}[U_n(m-1) + 4U_n(m) + U_n(m+1)]$$
$(m = 1, 2, \cdots, M)$ (3.4)

既知の初期値，境界値に対して式(3.4)の計算を4台のLPで並列に実行させる。すなわち，M点　点を4等分し，各LPに割りあて各反復において境界値を交換しながら計算を進める。またRPは初期値，境界値データの転送，ルーティング，計算の結果の収集を行ない，NPはルーティングだけ行なうとする。

## 4. Coralプロトタイプのソフトウェア

Coralのような分散型システムを実現する場合の問題点として次の事項があげられる。

(1) プロセス間結合

◦通信の方法．
◦プロセス間の処理の同期．
◦システムの拡張性
(2) 性能
◦並列処理するためにジョブを分解し処理後統合するためのオーバーヘッド
◦並列に処理されるプログラムが、一般に同型でないためによる各プロセサでの処理の待ち。
(3) 並列処理の方法
◦システムに適した処理方法
◦処理の記述

以上のような分散型システムの問題に対して、まずOSを設計する場合，次のような事を考慮しなければならない。

多数のプロセサの処理の実行制御の方法として，マスタ/スレイブ構成，各プロセサが夫々実行管理する構成，対称型構成の3つがある。本論文で述べるOSは、すべてのプロセサ上の実行制御は，同一構造となっているので対称型構成である。

通信には、同期と転送の機能が必要となる。また，通信の方式として1対1通信や放送がある。特に，放送は分散型システムでは、有効な通信方法となる。プロトタイプでは，1対1通信，放送の両方を使用する。

分散型システムでは、OSの作成と保守を容易にするために，OSのモジュール化が必要である。本論文で述べるOSは，各ノードごとに同一のモジュールからできている。

次に、分散型システムのためのプログラミング言語は、各ノードでのプロセスの定義，通信処理を記述しなければならない。それでRPに言語プロセサをもたせることによって，コンパイル，ロードすることが考えられる。

次に、分散型システムのユーティリティ・プログラムは，プログラムやデータを分散して配置する機能などが必要である。本論文では，その一つのIPLについて述べる。

最後に，分散型システムのアプリケーションとして適する問題は，大量のデータに対し同様な処理を繰返す問題と考えられる。これは，各ノードのプログラムを同型にするためである。また，プログラムはできるだけ通信の少ないアルゴリズムを用いる。本論文では、アプリケーションの例としてPDEとFFTをとりあげる。

## 4.1 CPOS: Coral Prototype Operating System

### 4.1.1 Coralでの処理

Coralで処理を行なう場合，RPから下部のプロセサへ，プログラムやデータを分配し，各ノードで処理を行ない，必要があればデータの交換なども行なって，最後に処理結果をRPに集める処理方法をとる。処理におけるプロセサ間通信は、メッセージを用いて行なう。

以上のような処理を実現するためのOSが，Coral Prototype Operating System (CPOS) である。この設計に、P.Brinch Hansenの提案したマルチプロセサネットワークシステムのためのプログラム[9]をCPOSの基本設計思想に取入れ、Coral用に改良し、次のように設計した。

### 4.1.2 CPOSの構成

CPOSは、各ノードとも同一のプログラム構成となっているので，各ノードごとにOSがモジュール化されている。

各ノード上でのCPOSは，図13に示すように構成している。

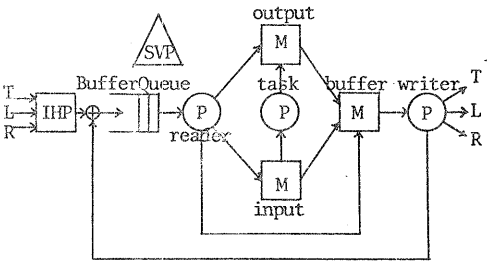



図13．各ノードのCPOSのデータフロー

1つのノード上のCPOSは、IHP(Interrupt Handling Program)，Supervisor，inputモニタ，outputモニタ，readerプロセス，writerプロセスから構成されている。そしてタスクプロセスが各ノードに1つづつ割りあてられている。

### 4.1.3 データの流れ

プロセサ間通信にパケットが用いられ，あるプロセスの実行中に，T,R,L方向から割込みがあると処理はIHPに移る。IHPは，1つのパケットを受取って，Buffer-Queueに入力する。その後元の実行処理に戻る。Buffer-Queueにパケ

ットが入力されたことでreaderプロセスが起動し，自分あてのパケットよりメッセージを再生し，その種類に従って各モニタにメッセージを渡す。その他のパケットは，他にルーティングするためにbufferモニタに渡す。

出力する場合，inputモニタやoutputモニタあるいはreaderプロセスからメッセージやパケットの出力をbufferモニタが依頼されるとwriterプロセスが起動され，writerプロセスがT,R,L方向に割込み転送する。相手が受取り不可能の場合は，自分のノードのBuffer-Queueに入力する。

### 4.1.4 処理の流れ

CPOS上でのジョブの実行は，タスクプロセス間で通信を行なって処理を進めていく。

タスクプロセス間の通信は，消費者タスクからrequestメッセージを生産者タスクへ送り，それに対する応答を生産者が処理し，結果をresponseメッセージによって返送する方法をとる。ここで消費者は，応答が返ってくるまで待ち状態となる。プロセス間通信は，タスクプロセスを1対1に結んだバーチャルチャネルを用いて行なわれる。

以上のような処理を実現するために，inputモニタとoutputモニタが使われる。

### 4.1.5 プロセス実行管理

プロセスの実行管理は，Supervisorによって行なう。Supervisorの機能は，プロセスの生成，事象の同期である。

CPOSで扱うプロセスの状態は，ready状態，run状態，wait状態，halt状態の4つである。これらの各状態ごとにSupervisorは，各種データベースを用いてプロセスを管理する。次に，各状態の関係を図14に示す。

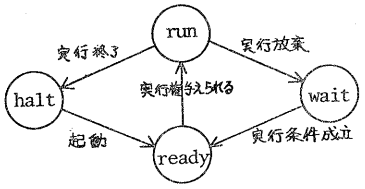



図14．プロセスの状態

各状態のプロセス管理を以下のように行なう。

(1) Run 状態プロセスの管理

run状態のプロセスの管理は，Running PCB Addressによって行なわれる。この制御域には，run状態にあるプロセスのPCBアドレスが格納されている。

(2) Ready 状態プロセスの管理

ready状態のプロセス管理は，Ready-Queneによって行なわれる。この制御域は，他のプロセスから起動要求があった場合，及びwait状態プロセスの待ちの原因が解決した場合など実行準備が整ったプロセスのPCBアドレスが格納されている。Supervisorはready状態のプロセスに優先順に実行権を与える。

(3) Wait 状態プロセスの管理

wait状態のプロセス管理は，ECB(Event Control Block)によって行なわれる。wait状態となる原因としてCPOSでは，5種類ある。

1. bufferモニタあき待ち
2. bufferモニタ入り待ち
3. Buffer-Queue入り待ち
4. 要求待ち
5. 応答待ち

### 4.1.6 プロセス管理のためのデータベース

(1) PCB (Process Control Block)

各プロセスの情報が書込まれていて，1つのプロセスのためのPCBは，32Bである。

(2) ECB (Event Control Block)

wait状態のプロセスが，どの事象について待っているか示す。各事象に対して，待っているプロセスの存在を示す部分と事象の完了を示す部分が用意されている。

(3) Ready Queue

Supervisorが，ready状態のプロセスの存在を知り，どのプロセスに実行権を与えるかを選択するために使用する。writer，reader，タスクの3つのプロセスのために用意されていて，各々についてreadyか否かを示している。

### 4.1.7 IHP (Interrupt Handling Program)

CPOSでプロセス間通信を行なう場合，送出し側は，受取り側に割込んでパケットを転送する。

割込まれた受取り側で割込処理を行なうプログラムがIHPである。次にIHPの機能を示す。

◦ House keeping
◦ 1個のパケットを入力する。
◦ Buffer Queue にパケットを入れる。
◦ 元の処理に戻す

次にIHPのアルゴリズムを示す。

```
/*IHP*/
begin
    (Interrupt Disable)
    Save Registers to Stack;
    if not Buffer Queue-full
       then
          begin
             Continue:=true;
             Input one message;
             ENQueue
          end;
       else    Disable:=true;
    Restore Registers;
    Interrupt Enable;
    Return
end.
```

### 4.1.8 プロセス間通信

(1) パケットによる通信

プロセス間通信を行なうためにCPOSでは、タスクプロセス間を1対1に結ぶバーチャルチャネルを用いて、メッセージを転送する。

メッセージを転送する場合CPOSでは、パケットを用いて、メッセージをいくつかのパケットに分割して転送する。

各々のバーチャルチャネルには、チャネル番号を付けてあるが、夫々のチャネルがどのタスクプロセスを結んでいるかを知るために、次のように行なっている。

まずタスクプロセスは、各ノードに1つづつ割りあてられているので、タスクプロセス番号を各ノード番号を用いて表わす。

各ノードを1対1に結ぶために、次のように考える。

| S \ R | 1 | 2 | 3 ⋯ | ⋯ n |
|---|---|---|---|---|
| 1 | 1 | 2 | 3 ⋯ | ⋯ n |
| 2 | n+1 | n+2 | n+3 ⋯ | ⋯2n |
| 3 | 2n+1 | 2n+2 | 2n+3 ⋯ | ⋯3n |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| n | ⋯ | ⋯ | ⋯ | ⋯ $n^2$ |

ノードの数をn，チャネル番号をCとすると，

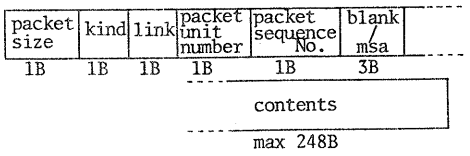
$$C=(S-1)\cdot n+R$$

$$\begin{cases} S=\text{差出し番地} \\ R=\text{受取り番地} \end{cases}$$

となる。またチャネル番号を作るときも，この関係を用いる。

(2) パケットフォーマット

プロセス間通信で用いるパケットのフォーマットを図15に示す。




| Kind | Link |
|---|---|
| request | Channel-No. |
| response | Channel-No. |
| broadcast | Node-No. |
| define | Node-No. |

図15．パケットフォーマット

パケットの最大転送量は，headerを含めて，256Bである。また，packet-unit数とpacket-sequence No.は、一つのパケットにメッセージが入り切らない場合，複数個のパケットを用いて転送するために設けている。次に，パケットがdefineとbroadcastの場合，linkはノード番号で示される。これはdefineの場合は，このパケットを用いて各ノードのタスクプロセスを定義し，broadcastの場合は，このパケットをそのノードのファミリすべてに放送するために用いている。

(3) Buffer Queue の構成

IHPによって入力されたパケットは，Buffer Queueに入力される。これは待ち行列として構成されており，Queue Control Blockによって管理される。ここでは、各スロットのstatusとして，empty，in-use，fullの3つを用いている。

Buffer Queue を操作するために，パケットを1個入力するENQueueとパケットを1個取出すDEQueueの2つのオペレーションがある。

各ノード上のBuffer Queueのスロット数は，入力方向が自分を含めて4つであるので，それの1.5倍として6と決めた。

### 4.1.9 モニタ

各モニタは，ノード上でのプロセス間通信に携わっている。

まずinputモニタは，receiveとresponseの2つのオペレーションより構成されている。re-

ceiveが要求を出し，responseが応答を受取る働きを行なう。inputモニタのアルゴリズムを次に示す。

```
/*Input Monitor*/
type input =
monitor(buf:buffer);
var this:message;
procedure entry receive(c:channel;var v:item);
begin
  with this do
   begin kind:=a_request; link:=c end;
  buf.send(this); delay(receiver);
  v:=this.contents
end;
procedure entry response(m:message);
begin
  this:=m; continue(receiver)
end;
```

また，NPとLPではinputモニタの機能として，タスクプロセスの定義も行なう。

outputモニタは，sendとrequestの2つのオペレーションより構成されている。requestが要求を受取り，タスクプロセスに処理の依頼をし，処理されたデータをsendによって転送する働きを行なう。outputモニタのアルゴリズムを次に示す。

```
/*Output Monitor*/
type output =
monitor(buf:buffer);
var list:array[channel] of
        record ready:boolean end;
      c:channel; this:message;
procedure entry send(c:channel; v:item)
begin
  with list[c] do
  if not-ready then delay(sender);
  with this do
   begin
     kind:=a_response; link:=c;
     contents:=v
   end;
  buf.send(this);
  list[c].ready:=false
end;
procedure entry request(m:message);
begin
  with list[m.link] do
   begin ready:=true; continue(sender) end
end;
```

また，RPのみoutputモニタに，各ノードに対してタスクプロセスの定義と放送の機能を持つ。

bufferモニタは，sendとreceiveの2つのオペレーションより構成されている。sendは，メッセージの転送をwriterプロセスに依頼しメッセージをパケットに分解する。receiveは，writerプロセスにbufferよりパケットを渡す働きを行なう。次にbufferモニタのアルゴリズムを示す。

```
/*Buffer Monitor*/
type buffer =
monitor
var buf:unit of packet; P:packet;
proceduer entry send(m:message);
begin
  if m=message
    then begin
           repeat pack m into P;
            if buf.full then delay(sender);
            buf.put(P);
            continue(receiver);
           until decompose-end
         end;
    else begin
           if buf.full then delay(sender);
           buf.put(P);
           continue(receiver)
         end;
end;
procedure entry receive(var P:packet);
begin
  if buf.empty then delay(receiver);
  buf.get(P);
  continue(sender)
end;
```

### 4.1.10 Readerプロセス

readerプロセスは，Buffer-Queueからパケットを取込み，それが自分あてのパケットならば受取り，その他ならばbufferモニタに渡す。いくつかのパケットを受取って，それを1つのメッセージに再構成する。このメッセージがrequestメッセージならばinputモニタに。responseメッセージならばoutputモニタに，defineメッセージならばinputモニタに渡す。以上のような機能をもつreaderプロセスのアルゴリズムを次に示す。

```
/*Reader Process*/
type readerprocess =
process(inpset,outset:channelset;
        nnum:node-No.;
        inp;input;out:output;buf:buffer);
var P;packet; m:message;
begin
    cycle
     repeat
      deq;
      if P-arrived=false
        then delay(readerprocess);
        else with P do
               if not(link or nnum)
                 then buf.send(P)
                 else compose m
     until compose-end;
```

```
      case m of
       (kind=a_response):inp.response(m)
       (kind=a_request) :out.request(m)
       (kind=define)    :inp.define
      end
     end
end;
```

次に転送されてきたパケットからメッセージを再構成する方法を示す。まず，タスクプロセスがメッセージを置く番地としてMSA(Message Start Address)が与えられているとする。そして各パケットのテキストがロードされる番地をLSA(Load Start Address)とすると，

$$LSA = 248 \times (P.seg.No. - 1) + MSA$$

但し，P.seg.No.はパケットのseg.No.で，
248はテキストの最大長である。

となり，この番地からパケットサイズのテキストをロードする。この方法をBuffer Queue上の同種のパケットすべてについて行なえばメッセージが再構成される。

### 4.1.11 Writerプロセス

writerプロセスは，bufferモニタからパケットを1個受取り，それをT,R,Lのいずれかのバスリンクに出力するか，自分のBuffer Queueに入力する。

writerプロセスは，他のノードに割込んで同期をとり，1個のパケットを転送する。この際ノードとノード間で同時に割込み合わないように，ノード間の位置関係で，高位が優先的に割込みを行なう。

次に，writerプロセスがどこにパケットを転送するかは，2.1.2で述べたルーティングを用いる。次に，writerプロセスのアルゴリズムを示す。

```
/*Writer Process*/

type writerprocess =
process(buf:buffer);

var P:packet;

begin
    cycle
     buf.receive(P);
     with P do
     interrupt-request;
     if continue=true
       then output-to-buslink(P)
       else enq(P)
    end
end;
```

## 4.2 タスクプロセスの一例

ここでは，プロトタイプ上でSandeのアルゴリズムによるFFTを並列処理する例について述べる。

データ数$M = 2^3$の場合のFFTの信号線図を例として図16に示す。

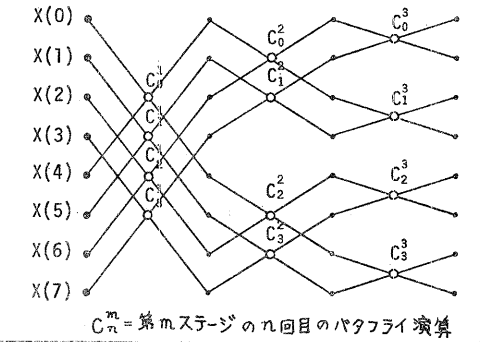



図16．データ数8のFFTの信号線図

この図でのバタフライ演算は，図17のように行なう。

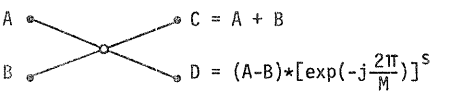



図17．バタフライ演算

図16からわかるように各ステージのバタフライ演算の回数は同じであるから，各ステージごとに並列処理させていく。

そこで，プロトタイプ上で各ステージのバタフライ演算を2分して処理する。すなわち，第1ステージを考えると，プロセサ2で$C_0^1$と$C_1^1$，プロセサ3で$C_2^1$と$C_3^1$を処理させる。さらにプロセサ2と3は，バタフライ演算の和のみ行い，複素掛算を下位のプロセサ4と5，6と7に行なわせて，バタフライ演算も並列処理させる。

## 5. 操作法

CPOSをプロトタイプ上にロードする方法とCPOSのもとでのオペレーションを次に述べる。

### 5.1 IPL (Initial Program Loader)

各エレメントプロセサは，IPLをもっている。

RPのIPLはファイルに，NPとLPのIPLは各プロセサのROMに用意してある。

IPLの機能は，インターフェイスと割込コントローラの初期化とRPの持っている1KBのデータをすべてのノードに放送することである。

操作は，NPとLPのIPLをスタート後，RPのIPLをスタートさせる。

### 5.2 CPOSLD(Cpos Loader)

5.1で述べたIPLを用いて，最初に放送する1KBがCPOSLDである。CPOSLDは，各ノードにCPOSをロードするプログラムである。NPとLPに共通にIPLによって放送される。このようにしてNPとLPに配置されたCPOSLDは，NPの場合そのノードへのOSを受取り，次にLPへのOSを中継した後，自分のOSをスタートさせる。また，LPの場合OSを受取りそれをスタートさせる。

### 5.3 CPOSでのオペレーション

各ノードのタスクプロセスの定義は，RPのoutputモニタのdefineによって行なわれる。これは，RPのファイルに持っている各タスクプロセスをdefineメッセージとして転送し，各ノードではそのメッセージをinputモニタのdefineによって受取られ定義される。

RPのタスクプロセスの例を次に示す。このタスクプロセスは，キーボードによって指定されたファイルを読込んで実行される。

```
/*Task1*/
type taskprocess =
process(inp:input; out:output);
var a,b:channel; x,y:message;
begin
  out.def(2,task2);
  out.def(3,task3);
      . . . .
  out.def(7,task7);
      . . . .
  receive(a,x);
      . . . .
  send(b,y);
      . . . .
end.
```

## 6. おわりに

高次並列処理計算機をめざした2進木並列処理システムCoralプロトタイプと，そのOSとして要求駆動型OSのCPOSについて述べた。

我々は，現在プロトタイプにもう1レベル加えて，15台のプロセサ構成に増設を計画している。この増設には，80系のi8085 CPUを用いたTK-85を使う予定である。また，プロトタイプのOSとしてCPOSの他にCORALOSを開発中である。

CORALOSは、Coralのアーキテクチャの特徴を生した階層構造をした分散型OSである。OSの各機能は，いくつかのOSのモジュールグループとして分散するように考えられている。各OSモジュールは，マスタ/スレイブ構成である。また，ジョブプロセスも，マスタ/スレイブ構成になっており，スレイブプロセスは，すべて同一に構成されていて処理結果をマスタジョブプロセスに返える。CORALOSでのジョブプロセスは、データ駆動で処理される。

今後，プロトタイプを15台に増設するとともに，ソフトウェアを拡充しCPOSまたはCORALOSを用いて，各種偏微分方程式について並列処理実験を行ないCoralの評価に役立てたいと考えている。

### 参考文献


1) 栗原,鈴木,元岡「High Level Data Flow Machine (TOPSTAR) のシステムプログラム」情報処理学会アーキテクチャ研究会37-16,1980

2) 金田「疎状結合型超多重プロセサシステムによる大次元連立一次方程式の並列計算」情報処理学会論文誌,Vol.21,No.5

3) S.H.FULLER:"Mul-Microprocessors:An Overview and Working Example"Proceeding of IEEE,Vol.66,No.2

4) R.KOBER:"SMS-201 A Powerfull Processor with 128 Microprocessors"EUROMICRO JOURNAL5,pp.48-52,'79

5) A.L.DAVIS:"The Architecture and System Method of DDM1:A Recursively Structured Data Driven Machine" Proc.Fifth Annual Symosium on C.A. pp.210-215,'78

6) P.A.FINKEL:"Processor Interconnection Strategies IEEE,Trans.Comput.Vol.C-29,No.5,pp.360-371, '80

7) 高橋,若林,信友"A Binary Tree Multiprocessor:CORAL" Journal of Information Processing,Vol.3,No.4,'80

8) P.BRINCH HANSEN:"NETWORK:A Multiprocessor Program IEEE Trans.Software Eng.Vol.SE-4,No.3,May,1978