オペレーティング・システム 33－7
（１９８６．１２．１３）

マルチプロセッサシステムＰＡＲＫのＯＳについて

松田秀雄，増尾剛，金田悠紀夫，前川禎男（神戸大学）

本稿では，論理型言語を並列に実行することを目的として設計されたマルチプロセッサシステムＰＡＲＫ（Parallel Processing System of Kobe University）の上に実装するＯＳについて述べる．ＰＡＲＫは16ビットマイクロプロセッサMC68000を要素プロセッサに使用したバス結合型システムである．高速スタティックRAMを使用したアドレス変換回路と放送機能を持つ共有メモリが特徴である．ＰＡＲＫ上に実現する論理型言語としては，我々が提案したPARK-PrologとICOTの並列推論マシンの核言語として選ばれたFGHC（Flat Guarded Horn Clauses）の２つを考えている．この２つの言語は，並列実行の方式や記述の方法が異なっているが，両方の並列実行を支援するモニタを設計した．本稿でこの２つの言語の並列実行の実現とモニタの機能を中心に述べる．

Operating System on Multiprocessor System PARK

Hideo MATSUDA, Tsuyoshi MASUO, Yukio KANEDA and Sadao MAEKAWA
Department of Systems Engineering, Faculty of Engineering, Kobe University
1-1, Rokkodai, Nada, Kobe, 657 Japan

We present an Operating System on multiprocessor system PARK (Parallel Processing System of Kobe University). The processing elements of PARK is 16-bit microprocessor MC68000 (Motorola), and all the processors are connected with a common bus. We will implement two logic programming language PARK-Prolog and FGHC (Flat Guarded Horn Clauses) on PARK. These two languages are different on the parallel execution method. We propose a monitor system which supports the parallel execution of these two languages.

１．はじめに

近年，人工知能分野の研究が盛んに行われている．Prologを代表とする論理型言語は，人工知能の研究ではLispと並び盛んに使用されている．しかし，自然言語処理等の応用分野で実用的なシステムを開発するには，論理型言語の実行を飛躍的に高速化する必要があると言われており，そのための試みの１つとして並列計算機による実行の高速化がICOTを中心に進められている．

本稿では，論理型言語の並列実行を目的として設計されたマルチプロセッサシステムＰＡＲＫ（Parallel Processing System of Kobe University）上に実装するＯＳについて述べる．ＰＡＲＫは16ビットマイクロプロセッサMC68000（モトローラ社製）を要素プロセッサに使い，これをバスにより結合したシステムである．

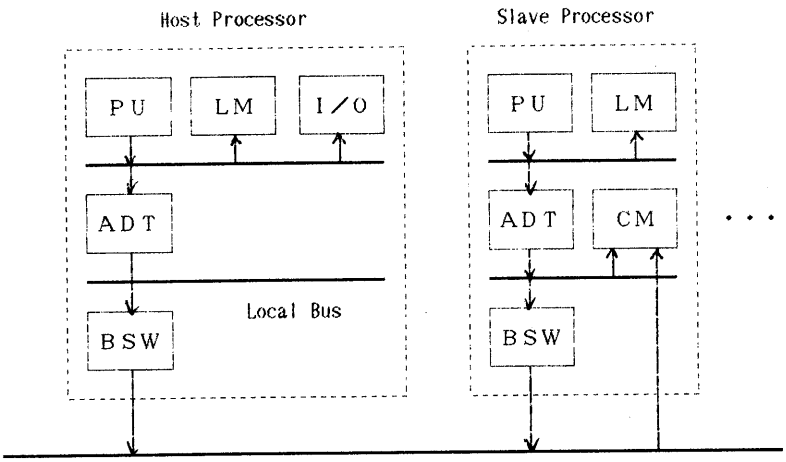
ＰＡＲＫ上で実現する論理型言語としては，我々の提案したPARK-Prolog[1]とICOTの並列推論マシンの核言語として選ばれたFGHC（Flat Guarded Horn Clauses）[2]の２つを考えている．この２つの言語は共に一階述語論理に基礎を置いた論理型言語という点では同じであるが，並列実行の方式や記述の方法が異なっている．

本稿で述べるＯＳは，これら２種類の論理型言語の実現に必要な機能を提供するために設計されている．

２．ハードウェア構成

ＰＡＲＫのハードウェア構成を図１に示す．図に示されるようにＰＡＲＫはバス結合型のマルチプロセッサシステムである．入出力を専門に行うホストプロセッサと並列実行用のスレーブプロセッサの２種類のプロセッサから構成されている．プロセッサ台数は，現在の所，ホスト１台，スレーブ３台の構成であるが，設計上はホスト，スレーブ合わせて１６台まで，単に共通バスに接続するだけで台数を増やせる．スレーブプロセッサは，他のプロセッサから参照できる共有メモリを持つ．

バス結合型システムは，全てのプロセッサがバスに対して対等なため，プロセッサ間の距離を意識せず通信が行えるなどの利点がある．しかし，その反面，台数が増え，バスの使用頻度が高まると，競合が生じ調停による待ち時間が増大するという欠点を持っている．バス競合を軽減するため，バスとメモリの階層化が成



PU：Processing Unit, MC68000 8MHz
LM：Local Memory, CM：Common Memory
ADT：Address Translation Unit
BSW：Bus Switch, Bus Controler/Arbiter

図1　ＰＡＲＫのハードウェア構成

されている（共通バス／ローカルバスと共有メモリ／ローカルメモリ）．

その他，特徴的な機能にアドレス変換と共有メモリへの放送があげられる．

ＰＵから共有メモリへ出力されるアドレスは，アドレス変換回路により変換される．変換はページ単位で行われる（１ページ8Kバイト）．読出しと書込みで別のアドレスに変換できる．

変換する先をメモリが存在しないアドレスに設定しておくと，そのアドレスへの参照があった時にはＰＵであるMC68000にバスエラー割り込みがかかるようになっている．この機能を利用してメモリの境界を越えた参照や，特定のアドレスに対する保護などが容易に実現できる．

アドレス変換は高速スタティックRAMをテーブルとして使うことにより実現されており，１CPUクロック（125ns）以内に変換できる．

各々のスレーブプロッセの共有メモリ（容量はそれぞれ512Kバイト）はアドレスが異なっており，アドレスでどのプロセッサのメモリか指定できるようになっている．また，特定のアドレスに対して書込みを行うと，放送（全てのプロセッサの共有メモリへ同一のデータを一斉書込み）が行われる．全てのプロセッサで共有され，しかも頻繁に読出しが起こる様なデータについては，放送により全プロセッサにデータを分配することにより，バス競合を減らすことができる．

３．並列論理型言語

３．１　PARK-Prolog

PARK-Prologは逐次型Prologにプロセスの生成，プロセス間の同期・通信の機能を付加し，並列実行が記述できるように拡張したものである[1]．

PARK-Prologでのプロセスとは，ゴールの逐次的な実行を指す．プロセス間で共有されるのは，プログラムに相当する定義節と以降で述べるチャネルと呼ばれる特殊な領域だけであり，変数の束縛情報を保持する環境は共有されない（多環境方式）．プロセスの生成やプロセス間通信など，プロセス間で変数の値が受け渡されるときには，その値がコピーされて受け渡される．

PARK-Prologでは，逐次実行部分の記述は DEC-10 Prologと同様の記法で行う．すなわち，逐次ＡＮＤ‘，’，逐次ＯＲ‘；’の２つの２項演算子で記述する．並列実行のための記法としては，次のようなものが用意されている．

（１）並列ＡＮＤ

記法：　ゴール１　／／　ゴール２

逐次ＡＮＤ‘，’と同様，ゴールとゴールとを結ぶ２項演算子として使い，ゴール１，ゴール２のそれぞれを実行するプロセスが生成される．

構文の形式上，並列ＡＮＤと呼んでいるが，これで結ばれた２つのゴールは，次に示すように論理的なＡＮＤ関係にはならない．

(a) ゴールの引数の値は全てコピーされて子プロセスに渡される．未定義変数を含んでいる場合は，その辺数の値を保持する領域が新たに作られる．すなわち，２つのゴールの間に共有変数があっても，プロセスの生成の後には独立した変数となる．

(b) どちらか一方のゴールの実行が失敗しても，他方のゴールの実行に基本的には影響しない．

これらの論理的なＡＮＤから削られた機能は，プロセス間通信を用いてプログラム中で陽に記述することにより，補うことができる．

（２）チャネル生成

記法：　mkchan（［チャネルリスト］）

プロセス間通信のためのチャネルを生成する組込み述語である．チャネルリスト中にある変数の個数だけチャネルが生成され，チャネルへのポインタが各々の変数に代入される．

（３）fork述語

記法：　fork（ゴール）

与えられたゴールを実行するプロセスを生成する組込み述語である．並列ＡＮＤとは，生成されるプロセスが２つから１つに変わった点を除けば同じ様に機能する．すなわち，並列ＡＮＤ　Ｐ／／Ｑは，forkを使

って，次のように表すこともできる．

P // Q :- fork(P), fork(Q).

（4）チャネルへの送信

記法：　チャネル！項

mkchanにより作られたチャネルに対して，項を送信（send）する組込み述語である．次の？（チャネルからの受信）と同期を取っており，？による受信より先に実行されると受信動作が行われるまで実行が一時中断（suspend）する．

データの受渡しは！と？の後の項を統一化（unification）することにより行われる．送信される項は送信に先だってコピーがとられ，コピーが送られる．すなわち，受信側から送信側へ値を受け渡すことはできない．統一化が失敗した場合は，？が再度実行されるまで実行が一時中断される．

（5）チャネルからの受信

記法：　チャネル？項

mkchanにより作られたチャネルに対して，項を受信（receive）する組込み述語である．！と同期を取っており，！による送信より先に実行されると送信動作が行われるまで実行が一時中断される．通信の際に行われる統一化が失敗すると，？の実行は失敗する．

以下に，PARK-Prologのプログラムの例を示す．以下では，ゴールの実行によりチャネルCとゴールp(C)を実行するプロセス（プロセスpと呼ぶ）とゴールq(C)を実行するプロセス（プロセスqと呼ぶ）の2つのプロセスが生成される．

プロセスpはチャネルCにアトムaを送信し，プロセスqはチャネルCから変数Xに受信する（結果的にXにaが代入される）．

```
ゴール：　　:- mkchan([C]),(p(C) // q(C)).

定義節：　　p(C) :- C!a.
　　　　　　q(C) :- C?X.
```

3．2　FGHC

GHC（Guarded Horn Clauses）については，文献[3]を初めとして数多くの文献で紹介がされている．従って，ここではその概略を述べるにとどめる．

GHCでは，その名の通り，ホーン節にガードのついたガード付きホーン節をプログラムの基本単位とする．ガード付きホーン節は以下のような形をしている．

H :- G1,‥,Gm | B1,‥,Bn.

ここで，H，G1,‥,Gm，|，B1,‥,Bnはそれぞれヘッド，ガード，コミットオペレータ，ボディである．ヘッドとガードを合わせて受動部（Passive Part），ボディを能動部（Active Part）と呼ぶ．

GHCでは全てのゴールがプロセスであり，並列に実行することが可能である．変数の束縛環境はただ1つ（単一環境）であり，プロセス間で1つの環境を共有する．

受動部と能動部とで変数の取り扱いが違う．受動部では，その中にある未定義変数への代入は許されるがそれ以外の変数を具体化することは禁止される．能動部の実行では，どの変数も自由に値を代入できる．プロセス間の同期・通信は，PARK-Prologのように組込み述語により行うのではなく，この変数への代入規則によって実現される．

FGHCは，GHC（Guarded Horn Clauses）のサブセットであり，GHCに対して以下のような制限を加えたものである[2]．

(a) ガードにはユーザ定義の述語を置かない．
(b) ゴールから呼び出される候補節のヘッドの統一化（unification）とガードの実行は逐次的に行う．

4．並列ＯＳ

4．1　構成

PARK-PrologおよびFGHCの実行はインタプリタにより行われるのではなく，コンパイラによりいったんオブジェクトコードにコンパイルしてから行われる．全ての定義節が実行に先だってコンパイルされている必要がある．

コンパイルは，最終的にMC68000の機械語への展開の

ところまで行われるが，途中でWarrenの抽象マシン命令[4]に基づいた中間言語にいったん変換される．

処理系としては，このコンパイラと，コンパイラが生成したオブジェクトコードの実行を支援するモニタの２つから成っている．

コンパイラはPARK-PrologとFGHCとでそれぞれ用意されるが，モニタは１つとなっている．PARK-PrologとFGHCとでは，変数の束縛環境の表現，並列に実行されるプロセス間での同期・通信の方法などが異なっている．

モニタはこれら２つの論理型言語のいずれの実行にも対応できるようになっている．

モニタは次の５つの部分からなる．

①実行管理モジュール
②入出力管理モジュール
③プロセッサ管理モジュール
④プロセス管理モジュール
⑤メモリ管理モジュール

以下，これらについて説明する．

（１）　実行管理モジュール

UNIXでいえばshellに相当するもので，ユーザの入力したコマンドを解析し実行する．オブジェクトコードをロードし並列実行を開始するコマンドの他，実行中にスレーブプロセッサの状況（割り当てられたプロセスの数など）を表示するコマンドが用意される．

ホストプロセッサに置かれる．

（２）　入出力管理モジュール

コンソール入出力，ファイル入出力を行う．ホストプロセッサに置かれ，スレーブプロセッサが入出力を行う組込み述語を実行するとプロセッサ間通信により呼び出され，入出力処理を行う．

（３）　プロセッサ管理モジュール

プロセッサ間の通信を行う．全てのプロセッサに置かれ，次のような機能を実現する．

(a) ホストからスレーブの実行の起動・停止を制御する．
(b) オブジェクトコードをスレーブにロードする．
(c) スレーブからホストの入出力管理モジュールを呼び出す．

（４）　プロセス管理モジュール

プロセスの実行を制御する．スレーブプロセッサに置かれ，次のような機能を実現する．

(a) プロセスの生成・消滅
(b) プロセスのプロセッサへの割当ての決定
(c) プロセスのスケジューリング
(d) プロセス間の同期・通信

プロセスは次の４つの状態をとりうる．

・スケジュール（scheduled）
・実行可能（ready）
・実行（running）
・実行中断（suspend）

図２にプロセスの状態の遷移を示す．プロセスは最初に生成されたときスケジュール状態となり，実行に必要なメモリの割当が行われると実行可能状態になる．実行可能状態のプロセスのうち一つがスケジューラにより選択され，実行の制御が渡され実行状態にはいる．実行中のプロセスに同期待ちが生じると実行中断状態に入り，スケジューラに再び制御が戻る．同期待ちが終了すると，そのプロセスは実行中断から再び実行可能状態に戻る．

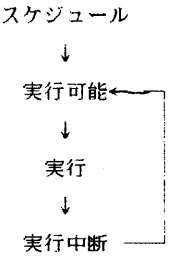


図２　プロセスの状態遷移

スケジュールおよび実行可能状態のプロセスは，そのPCB（プロセス制御ブロック）が，スケジュールリスト，実行可能リストの形でつながれている．実行実行中断状態のプロセスは，中断の原因となったところ（PARK-Prologではチャネル，FGHCでは共有変数）からリンクが張られ，実行中断リストを形成する．

プロセスのスケジューリングは，各スレーブプロセッサで独立して行われる．スケジューラ，スケジュールリスト，実行可能リストは各スレーブについてそれぞれ置かれる．

（5） メモリ管理モジュール

PARKのアドレス変換機構が持つ共有メモリのページング機能を利用したメモリ管理を行う．次のような機能を実現する．

(a) プロセスの生成時に必要なメモリの割当て

(b) プロセスの消滅時に不要なメモリの回収

(c) プロセスの実行時に出される領域拡張要求によるメモリの追加割当て

メモリの領域拡張要求は，プロセスの実行時に領域の使用状況を監視する必要はなく，あらかじめ与えられたメモリ領域からはみ出して参照が行われたときに発生するバスエラー割り込み時に行われる．アドレス変換ハードウェアによって支援されており，メモリ境界を越えた参照の検出，および未使用のメモリページの獲得・割当て等の実現を容易にしている．

4．2 PARK-Prologの実行

4．2．1 プロセスの生成

プログラム中でゴールの並列実行の指定（並列ＡＮＤ，fork述語）があるとプロセスの生成が行われる．プロセスの生成は以下のような手順で行われる．

（1）生成プロセスを割り当てるプロセッサを決定する．

（2）割り当てられたプロセッサのPCB（プロセス制御ブロック）を１つ獲得する．

（3）ECB（環境コピーバッファ）を１つ獲得し，ゴールの引数のコピーを行う．

（4）PCB，ECBといったプロセス生成に必要な情報を割当て先プロセッサのスケジュールリストに登録する．

以上の動作によりスケジュールプロセスが１つ作られたことになる．

プロセスを割り当てるプロセッサとしては，実行時に動的に最も負荷の軽いプロセッサを選ぶ．ここでプロセッサの負荷とは受け持っているプロセスの個数と定義する．各プロセッサが受け持っているプロセスの個数は共有メモリの放送機能によりバス競合を起こすことなしに容易に読み出せるようになっている．

プロセスの生成時には，生成に先立って変数束縛環境の分離が行われる．ゴールの引数中の未定義変数およびリスト，構造体といったローカルスタック，グローバルスタックに値領域を持つ項がECBにコピーされる．これらの項を指していた引数も全てECB内を指すように変換される．ゴールが最初に呼び出すべき節のアドレスもこの時にECBへ書き込まれる．

4．2．2 プロセスの切替え

プロセス間通信により，現在実行中のプロセスが実行を中断するとプロセス切替えが生じる．モニタのプロセススケジューラが呼び出され，そこに制御が渡される．

スケジューラは，実行可能なプロセスのうち実行可能リストの先頭にあるものを選んでそこに制御を渡す．ラウンドロビンのスケジューリングを行っているため，実行可能リストの中で以前選ばれた所の次にあるプロセスが選ばれる．

実行可能なプロセスが１つもない時には，スケジュールリストからスケジュール状態のプロセスを選ぶ．スケジュール状態にあるプロセスには，実行に必要な各種の領域（ローカルスタック，グローバルスタック，トレイルスタックなど）のメモリ割当てを行った後，制御が渡される．割当てに必要な情報はECBに書き込まれており，割当ての後ECBは解放される．

4．2．3 プロセスの消滅

プロセスは実行すべきゴールがなくなると消滅する．ローカルスタック，グローバルスタックなどの環境領

域の解放を行った後，PCBを未使用のPCBを保持している自由（free）リストにつなぐ．制御はスケジューラに渡される．

４．２．４　プロセス間通信

（１）チャネルへの項の送信

ECB獲得後，そこへ送信データ（！の後の項）の書き込み（未定義変数，リスト，構造体はコピーされる）を行った後送信を行う．

現在の所，制御の方向と同時に，統一化の際の変数への値の代入方向を限定している（受信側の変数への代入しか許さない）．

（２）チャネルからの項の受信

送信プロセスとの同期が取られると送信データが書き込まれているECBが渡ってくる．送信データと受信パターン（？の後の項）との統一化が行われる．統一化が失敗すると受信プロセスがバックトラックする．受信後，ECBは解放される．

４．３　FGHCの実行

４．３．１　プロセスの生成

プログラム中で能動部のゴール実行にさしかかるとプロセスの生成が行われる．プロセスの生成はPARK-Prologと同様の手順で行われるが，FGHCでは変数束縛の環境が１つだけなので，（３）の所ではゴール引数の値を単にECBに書き込むだけでコピーは行われない．

４．３．２　プロセスの切替えおよび消滅

受動部の実行でゴール引数中の変数の具体化待ちになると，実行中断状態にはいる．後は，PARK-Prologと同様のプロセスの切替えが行われる．

プロセスの消滅については，PARK-Prologと全く同じである．

４．４　モニタ機能の詳細

以上述べてきたモニタの機能を利用するためには，以下にあげるモニタ呼び出し命令を使用する．これらは，コンパイルの際の中間言語命令として位置づけられる．

（１）プロセス管理

・割当て先プロセッサの決定

```
select_processor
```

・自由リスト，スケジュールリスト，実行可能リストからのプロセスの取出し，追加

```
get_free_process
get_scheduled_process
get_ready_process
put_free_list
put_scheduled_list
put_ready_list
```

・プロセスのスイッチ（プロセススケジューラの呼び出し）

```
switch_PARK   (PARK-Prolog)
switch_FGHC   (FGHC)
```

（２）環境操作

・ECBの獲得，解放

```
allocate_ECB
deallocate_ECB
```

・ECBからの値の取出し，追加

```
get_ECB
copy_ECB   (PARK-Prolog)
put_ECB    (FGHC)
```

（３）チャネル操作　(PARK-Prolog)

・チャネルの生成，消滅

```
mkchan
rmchan
```

・チャネルのロック，アンロック（相互排除命令）

```
lock_channel
unlock_channel
```

・チャネルで同期待ちのプロセスの存在チェック，チャネル内のプロセスの取出し，追加（実行中断／再開）

check_send_link
check_receive_link
get_send_process
get_receive_process
put_send_process
put_receive_process

（4）共有変数操作　(FGHC)

・共有変数のロック，アンロック（相互排除命令）
lock_variable
unlock_variable
・共有変数で同期待ちのプロセスの存在チェック，共有変数で同期待ちのプロセスの取出し，追加
check_variable
get_suspend_process
put_suspend_process

5．おわりに

本稿では，マルチプロセッサシステム上で，2種類の並列論理型言語（PARK-PrologとFGHC）を実現の方式と，これらの言語の並列実行を支援するモニタについて述べた．

謝辞　本研究は一部文部省科学研究費補助金（奨励研究（A））によっている．

参考文献

[1] 松田秀雄，小畑正貴，増尾剛，金田悠紀夫，前川禎男：並列PrologマシンＰＡＲＫについて，Proc. of Logic Programming Conference '85, 2.4(1985).
[2] 宮崎敏彦，瀧和男：multi-PSIにおけるFlat GHCの実現方式，Proc. of Logic Programming Conference '86, 7.2(1986).
[3] Ueda,K.: Guarded Horn Clauses, Proc of Logic Programming Conference '85, 9.3(1985).
[4] Warren,D.H.D: An Abstract Prolog Instruction Set, SRI Technical Note 309(1983).