ICOT Technical Memorandum: TM-0236

TM-0236

並列論理型言語に基く
オペレーティング・システム

岸下　誠，大田由紀子，神田陽治
(富士通)
田中二郎

October, 1986

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191-5
Telex ICOT J32964

Institute for New Generation Computer Technology

並列論理型言語に基くオペレーティング・システム
An Operating System Based on a Parallel Logic Language

田中二郎　岸下　誠　大田由紀子　神田陽治
(ICOT)　(富士通SSL)　(富士通)

ICOTでは並列論理型言語GHC [Ueda 85] に基き、KL1 [Furukawa 85] を設定し、並列論理型言語に関する各種の研究を行ってきた。本論文では、並列論理型言語とOSの背景について述べた後、我々の計画中である並列論理型言語に基くオペレーティング・システム(OS)の構想について述べる。計画中のOSの諸機能、構成例について述べた後、現在の問題点についても簡単に述べる。

1.はじめに

第五世代コンピュータ・プロジェクトとは、高度の知識情報処理と並列計算機アーキテクチャとを論理型プログラミングの枠組で統一的に扱っていこうとするプロジェクトである。ICOTでは、そのため既に並列論理型言語GHC[Ueda 85]に基づいた核言語KL1[Furukawa 85]を設定し、アーキテクチャ、言語処理系、プログラミング・スタイル、アプリケーション等から各種の研究を行なってきた[Tanaka 86a]。

アーキテクチャの研究は、主として並列処理マシンの構築というハードウェア・レベルで行われてきたが、研究の進展と並行し、多くの問題の所在が明らかになってきた。それらは、①並列システムにおける、job 管理、user管理等の実行制御方式、②並列システムにおける、入出力、割込み、例外処理の扱い、③並列、分散環境でのプログラム・コードの分配、管理方式、④プロセッシング・エレメント間の負荷の分配、分散方式、等の問題である。

こういった問題は、ハードウェアやファームウェア・レベルのみで解決されるべき問題では無く、より上位のソフトウェアを含め、解決が成されなければならない。これらの問題は、開発されるべきハードウェアの有効利用といった視点からのみならず、ハードウェア設計自体を明らかにするためにも、研究の伸展が急務となっている。

2.並列論理型言語とOSの背景

並列論理型言語には、GHC やKL1 の他に、Gregory & Clark らによるRelational Language やParlog [Clark 85] 、Shapiro によるConcurrent Prolog [Shapiro 83]などがある。歴史的にはGHC やKL1 はこれらの言語の改良版として提案されたものである。

これらの言語の発展過程を振り返ってみると、まず言語仕様決定の後、prologによるimplement がなされいる。しかしながらprologによるimplement には、実行速度や走らせることのできるプログラムの大きさ等に制約があり、c 言語等による直接のimplement が行われるようになっている。c 言語によるimplement においてはprologの言語環境や述語などを利用することができないので、環境やOS作りに関心が移ってきている。

並列論理型言語では、『perpetual なゴール』を『プロセス』と、『プロセス間のストリーム通信』を『プロセスの通信』と見立てることが出来る。これらの言語によるシステムプログラムの記述は以前より試みられており[Clark 84, Shapiro 84] 、並列論理型言語用のOSを作ろうという構想は、これらの試みの発展と考えることができる。

3.並列論理型言語に基くOSの構想

我々の想定するOSのイメージを図に示したのが図1 である。

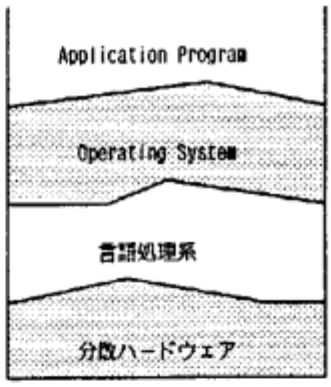

図1 並列論理型言語によるOSの全体像

この図によっても解るように、並列ハードウェアの上には、（おそらくファームで実現される）並列論理言語の処理系がのり、OSはその上位に位置することとなる。言語処理系の上にOSがのるというのが単一言語系の計算機システムの特色であり、これはOSの軽量化に貢献すると同時に、言語処理系自体にOS諸機能実現のためのprimitive を持つことを要求することとなる。

我々の仮説は、並列論理型言語においては『プロセス』や『同期』を言語レベルで扱えることから、言語仕様そのものがOSの諸機能実現に自然にmatch するのではないかということである。

これらを『OSの設計の方針』としてまとめると以下の 2 点に集約される。

①OSは前節で述べたソフトウェアの面での諸問題の解決を目指すものであること。即ち、並列マシンを有効に動かせ

るものとすると同時に、並列アーキテクチャ研究を加速するようなものでありたい。
②並列論理型らしいOSでありたい。即ち、単に低レベルのコーデングを積み重ねてOSを作るのではなく、実際に並列論理型言語でOSを構築することにより、並列論理型言語がOS構築に適した言語であることを立証したい。

4.想定ハード、言語処理系

ここでまずOS構築の基礎となる想定ハードウェア及び言語処理系について簡単に述べておきたい。

まず想定ハードウェアであるが、とりあえずOSのターゲットとして、ICOT中期計画で構築されるMulti-PSI [Taki 86] を想定することにする。

Multi-PSI とは、複数台のPSI マシンに接続ネットワーク・ハードウェアを加え、格子状に接続した分散システムで、PSI どうしは共有メモリは持たず、メッセージ・パケットの交換を通じて通信を行う。

次に言語処理系であるが、対象言語としてKL1 を想定する。KL1 については現在も開発中の言語であり、KL1 言語系自体もさらに幾つかの言語要素に分かれるのであるが、ここではとりあえずKL1 を『GHC のguard 部分にシステム述語だけを許すFlat GHC (FGHC) に、分散環境上でのプログラムの実行を指示するpragmaを加えたもの』と考える。

Multi-PSI 上では、KL1 プログラムが分散実行される。分散環境下においては、ユニフィケーション一つにとっても時間がかかる。変数の値を得るには、変数の存在するPSI にメッセージを送り、その返答を待つ必要がある。その間、プロセスはサスペンドするとみなければならない。この分散実行については、すでに様々な検討がなされ、実際にシミュレータ、処理系も幾つか試作されている[Murakami 85、Tanaka 86b、Ohara 86、Miyazaki 86]。

Multi-PSI における言語処理系とOSとの接点を考えた場合、少なくとも理想的には言語処理系でシステムの基本的機能は実現されるべきである。すなわち入出力やコードのローディングなどを除くシステムの基本的機能は（一つのPSI 内では）実現出来、また何かが起きたら割出せる基本的メカニズムは言語処理系が持ち、OSはそれから先を担当すべきである。

即ち、Multi-PSI 上のKL1 分散処理系は次の特徴を持つ。
①メッセージの自動生成

KL1 プログラムが実行される過程で接続ネットワーク間を渡るメッセージ・パケットは自動的に生成される。言い替えるとメッセージ・パケットはプログラマが陽に扱うのではなく処理系が合成や分解をおこなう。
②Priority Scheduling

何かが起きたらそれに即応して処理の行える基本的メカニズムとしてPriority Scheduling 機能を持つ。1 つのProcessing Element(PE)内では、仕事はPriorityの高いものから処理される。一般にシステムの受け持つ仕事の中にはuserの仕事よりPriorityの高いものがあると考えられ、何段階かの優先度で仕事を処理する機能を持つことは重要である。
③分散環境上での仕事管理機構

分散環境上でユーザ・プロセスの失敗がOS全体を失敗させず、また一般にユーザ・プロセスは幾つかのPEに渡って実行されると考えられ、OSが幾つかのPEに渡ったユーザ・プロセスの資源や実行を制御、管理しながら走らせることができるための機構が必要である。

5. PPSとLogix

現在ParlogやConcurrent Prolog については、環境やOS作りに焦点が移ってきていると前述したが、これらの言語における環境やOS作りの現状についてまとめてみる。

まずParlogであるが、現在PPS(Parlog Programming System)と呼ばれるプログラミング環境が完成している[Foster 86b]。PPS はSUN 上で動き、その特徴は①幾つかのquery を同時に評価できるmulti-processing、②query を評価中にさらに他のquery を入力できること、及びそれらの実行やi/o の柔軟な制御、③プログラム実行中に未定義述語が現れたときuserに尋ねるquery-the-user機能、④幾つかの世界を同時に扱え、ある世界の知識を用いてquery を解くことができるモジュール化機能、等にある。

次にConcurrent Prolog であるが、現在の言語的関心は、Concurrent Prolog のguard 部分にシステム述語だけを許すFlat Concurrent Prolog (FCP)に移ってきている。Logix はSUN 上で動くFCP のプログラミング環境であり[Silverman 86]、①独特のモジュール化機能及び他のモジュールの述語を呼び出すremote procedure call 機能、②queryに使われる変数の束縛値を管理したり解放したりするuser interface機能、③multi-process 環境でのresolvent の概念に基いたdebug 機能等を特徴としている。

またこれらのPPS やLogix は分散implement をそれぞれ意識しており、PPS はsequent 社のBalance 8000による(shared memoryの）分散implement を計画中であり、一方Logix はintel のHypercube マシンで動く分散implement版が既に公開されている。我々のOS設計の指針もこれらのアプローチの延長線上にある。

6.OSの諸機能

OSの主な機能と思われるものを以下に列挙する。
①並列システムにおける、プロセスの管理、実行制御方式、例外処理

OSはユーザ・プロセスを制御しながら走らせることが必要である。またOSはユーザ・プロセスの資源を管理でき、job 管理やuser管理機能を持つことが望ましい。またユーザ・プログラムの実行中に発生するエラー、例外などもOS側から検出し、OS側からの指示などにより実行を再開できることが望ましい。
②並列システムにおける、入出力、割込みの扱い、

入出力は特定のストリームへデータをput することにより実現される。一般にOSは入出力Deviceからメッセージを受け取ると、それに応じ、割り込み処理プログラムが起動する。こういったプログラムは他より高いpriorityを持つと考えられ、速やかに処理される。
③並列、分散環境でのプログラム・コードの分配、管理方式

分散システム上で実行されるプログラムについては、簡単化のため、全てのPEがあらかじめ同じプログラムを持っていると仮定することもできる。しかしながら、全てのPEが同じプログラムを持っていると仮定するのは、

将来、非常にPEの台数が多くなった時を考えると現実的でない。必要に応じ、他のPEもしくは外部記憶装置等からプログラム・コードを持ってくるメカニズムを用意しておく必要がある。

④PE間の負荷の分配、分散方式

KL1 プログラムにおいては、プラグマ指定により、負荷の分散が静的に指定されるが、動的に、実行時にも負荷分散の調整を行うことがが必要である。

7. OSの構成例

前節に示したOSの諸機能を踏まえ、OSの構成例を示したのが図2 である。

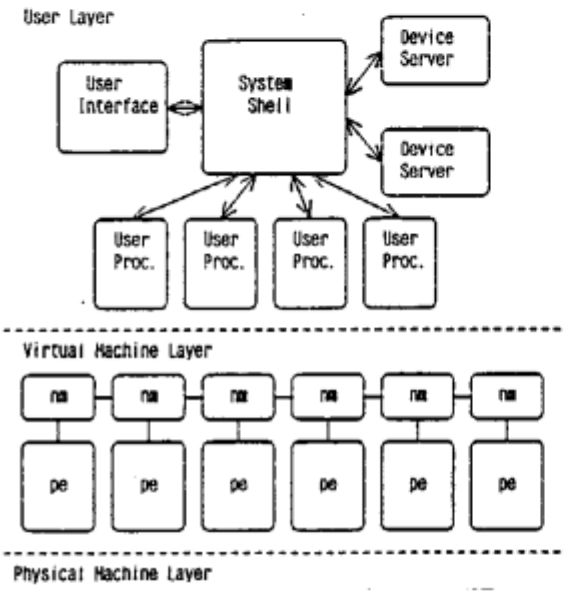

図2　OSの構成例

まずPhysicalなマシン上にVirtual Machine Layer を構築する。Virtual Machine Layer は必ずしもPhysicalなマシンとは一対一に対応するとは限らず、例えばPhysicalなマシンが2x3 のmeshであったとき、Virtual なマシンは無限gridであっても良いし、或いは『近山方式』として知られる、単位平面に均一にprocessing powerが拡がっているようなモデルであってもよい。User Layerはその上に組み立てられる。ここではVirtual Machine Layer として 6個のPEと 6個のNetwork manager(NM) からなる仮想マシンを仮定している。

Virtual Machine Layer の上にはUser Layerが組み立てられる。ここでUser Interfaceとは、userから入力されたquery を扱うところで、さらにこの部分はkeyboard, tokenizer, parser, goal compilerなどを含むと考えられる。また、Device Server は各入出力機器に対応して存在し、例えばFile Server, Screen Server等が考えられよう。

このUser Layerで中心的役割を果たすのはSystem Shellである。System ShellはUser Interfaceからquery を読み取り次々にUser Processを生成する。User Processを生成する典型的技法は、[Foster 86b]にみられるような、メタ・コールである。ここでは例として、

```
call(Goal,World,Resources,Control,Status)
```

というメタ・コールを考える。ここでGoalは実行されるべきGoal、World はGoalを解くのに使われるAxiom set 、Resources はメモリ（アクティブな実行木のノード数）や実行時間(Reduction回数）などの制限、Control はメタ・コールで実行しうるプロセスの中断、再開、終了等を指示するストリーム、Statusはメタ・コールからの返答や、例外等が発生した場合に生じるメッセージを格納するストリーム、を表している。

一般に、ユーザ・プロセスで例外が発生した場合、メッセージがStatusストリームから上がってくるが、発生した例外の種類により適切な処置をとるように例外処理プログラムを書いてやればよい。

次に入出力の扱いであるが、これは入出力Deviceに対応するperpetual なプロセスを仮想的に考えればよい。Device Server は、そこからメッセージを受け取ったり、それにメッセージを送ることができる。入出力Deviceに対応するストリームを取ってくるには、組み込み述語を用意すればよい。例えば、 interrupt_stream（機番、Stream）などという組み込み述語を用意するのも一手であろう。この述語を使うことにより機番を与え、対応するDeviceに相当するストリームを取ってくる事ができる。割込み処理のプロトコルを図 3に示す。

```
Device Server 側
  ..., interrupt_stream（機番、Stream）,
          interrupt_handling(Stream) #1, ...

割込み処理の記述
   interrupt_handling([Data | S]) :-  true |
              処理(Data),
             interrupt_handling (S).
```

図 3　割込み処理のプロトコル

interrupt _stream（機番、Stream）は、たかだか一回だけしか実行されない組み込み述語である。ここではpragma『#1』を付けpriorityの指定（この場合は1)を行っている。

割込みの速やかな処理のためには、割込みの際put されるデータの量はできるだけ少なくする工夫が必要である。put されるデータの量が大きいとresponce time の低下につながる。そのため、入出力Buffer等に対する述語は組み込みで用意し、ストリームにput するデータは 1～ 2語の割込みコード及びaddress 程度にとどめるなどの工夫も有効と思われる。

8. 今後の課題

以上簡単に並列論理型言語に基くOSの構想について述べた。しかしながらこの『並列論理型言語に基くOS』については問題点も少なくない。以下簡単に今後の課題について

述べる。

①言語レベル

第一に言語処理系自体の高速化が問題である。c言語によるimplement 、若しくはfirmwareによる直接のサポートが不可欠である。GHC のような言語においてはストリーム操作の高速化が重要である。また、モジュール化、perpetual なゴールの解りやすい表記法、システム・ストリームを引数として持ち運ばなくてもよい略記法の導入など、user言語の開発も急務である。

②メタ・コールの強化

特にOS開発にあたっては、メタ・コールの強化がOS開発の鍵となる。OSの基本機能はすべてこのメタ・コールにあると言っても過言ではない。現在のところParlogはメタ・コールをprimitive として提供しており、またFCP はメタ・コールをメタ・インタプリタの形で提供している。後者の場合、柔軟ではあるが実行速度に難がある。高速化の方法としてsource-to-source transformation 、もしくは部分評価も試みられているが、自動化には至っていない。

③OSの階層の構成化

OSの階層化の方法として、何段にも仮想マシンの階層を作る方法[Taylor 86] 等が提案されている。OSの階層化の方法やmulti-userのサポート等は今後の課題である。現状ではPSI マシン 6台を接続ネットワークで格子状に接続したMulti-PSI V1が既にハード的に完成しているので、現状のPSI 6 台を使ってのmulti-userサポートなど、一つの可能性として考えられる。

④ 5世代OSを目指して

我々の作ろうとしているのは 5世代言語によるOSである。既存のOSを並列論理型言語に書き換えるのではなく、例えばfile system はどうあるべきか、terminalとのinterface は何が望ましいのか考え直してみる必要があるように思われる。

9.謝辞

本研究は、第５世代コンピュータ・プロジェクトの一環として行なわれたものである。本研究の一部は、ICOT第一研究室、第四研究室の諸氏と行なった議論をふまえている。特に第四研究室の近山氏、滝氏、第一研究室の宮崎氏に感謝する。また本研究の機会を与えて戴いたICOTの古川室長、富士通国際研の北川会長、榎本所長に感謝する。

［参考文献］

[Clark 84]K.Clark and S.Gregory: Notes on Systems Programming in Parlog 、in Proceedings of the International Conference on Fifth Generation Computer Systems 1984、pp.299-306、ICOT、1984.

[Clark 85]K.Clark and S.Gregory: PARLOG: Parallel Programming in Logic, Research Report DOC 84/4 , Dep. of Computing, Imperial College of Science and Technology, Revised 1985.

[Foster 86a]I.Foster:The Parlog Programming System (PPS) 、Vesion 0.2、Imperial College of Science and Technology, 1986.

[Foster 86b]I.Foster: Logic Operating Systems: Design Issues 、draft 、Imperial College of Science and Technology, 1986.

[Furukawa 85] 古川他、核言語第一版説明資料ICOT, 1985.

[Hirsch 86]M.Hirsch et al.: Layers of Protection and Control in the Logix System 、Weizmann Institute, Israel, 1986.

[Miyazaki 86] 宮崎、滝: Multi-PSI におけるFlatGHC の実現方式、The Logic Programming Conference '86、ICOT、1986、pp.83-92.

[Murakami 85] 村上: マルチ・プロセッサにおけるユニファイヤ構成法の検討、Muilti-SIM検討会内部資料、ICOT、1985.

[Ohara 86]大原他:FGHC ソフトウェアシミュレータの試作、The Logic Programming Conference '86、ICOT、1986、pp.93-101.

[Shapiro 83]E.Shapiro: A Subset of Concurrent Prolog and Its Interpreter 、ICOT Technical Report 、TR-003、1983.

[Shapiro 84]E.Shapiro: Systems Programming in Concurrent Prolog 、in Proceedings of the 11th Annual ACM Symposium on Principles of Programming Languages、Salt Lake City、1984、 pp.93-105.

[Silverman 86]W.Silverman et al.: The Logix System User Manual 、Version 1.21、Weizmann Institute, Israel, July 1986.

[Taki 86]K.Taki: The Parallel Software Reseach and Development Tool: Multi-PSI System、France-Japan Artificial Intelligence and Computer Science Symposium 86, 1986, ICOT, pp.365-381.

[Tanaka 86a]J.Tanaka et al.: Guarded Horn Clauses and Experiences with Parallel Logic Programming 、Fall Joint Computer Conference、Dallas、Texas 、Nov. 1986.

[Tanaka 86b]J.Tanaka et al.:Distributed Implementation of FGHC -Toward the realization of Multi-PSI System- 、unpublished メモ、1986.

[Taylor 86]S.Taylor et al.: A Layered Method for Process and Code Mapping 、Weizmann Institute, Israel,1986.

[Ueda 85] K.Ueda: Guarded Horn Clauses、ICOT Technical Report 、TR-103、1985.