システムソフトウエアと
オペレーティング・システム 60−25
マルチメディア通信と分散処理 61−25
（１９９３．７．９）

リアルタイム画像生成システム用OSの機能と実装

藤井 茂樹　米田 泰司　日高 教行　浅原 重夫　鷲島 敬之

松下電器産業(株) 映像音響研究センター メディア研究所

リアルタイム画像生成システム AVIP は複数のノードを疎結合したマルチプロセッサである。この疎結合型マルチプロセッサを対象とした並列オペレーティングシステム AVIP-OS は、複数ノードにまたがる並列プログラム同士をお互いに協調させて高速動作させることを目的としている。AVIP-OS は CMU の Mach3.0 をベースにしており、各ノードごとに動作するマイクロカーネルと、あるノードで動作するシステムサーバ群から構成され各々が協調動作する。

本稿では、AVIP-OS の機能の 1) 複数ノードへのプロセス生成　2) 複数ノード間のデータ共有　3) 複数ノード間の同期処理、排他制御　について述べるとともに、その実装状況を報告する。

The Function and Implementation of an Operating System for Real-Time Image Generating Systems

Shigeki Fujii　Yasushi Yoneda　Noriyuki Hidaka
Shigeo Asahara　Takayuki Sagisima

Media Research Laboratory, Matsushita Electric Industrial Co.,Ltd.
1006 Kadoma,Kadoma-shi,Osaka 571 JAPAN

The real-time image generating system AVIP is a multi-processor system which consists of loosely connected multiple nodes. AVIP-OS is a parallel operating system for multi-processors. Its purposes are cooperation and fast execution of parallel programs which run on multiple nodes. AVIP-OS is based on Mach 3.0 and consists of micro-kernels and system servers. The micro-kernels which exist on each node and the system servers on only one specific node act cooperatively.

In this paper, we discuss the functions of AVIP-OS: 1) Process creation on multiple nodes　2) Data sharing among multiple nodes　3) Synchronization and exclusive control among multiple nodes　and describe implementation of these functions.

1 はじめに

我々は、映像システムのコアプロセッサとして画像生成・処理に特化したアーキテクチャを有する高速グラフィックスエンジン(AVIP:Audio Video Information Processor) を現在開発中である。採用したアーキテクチャは並列処理による高速演算と大容量データ格納を実現するため、複数のプロセッサと専用ハードウェアを搭載したプロセッサノードを数十個格子結合した疎結合型マルチプロセッサ構成である。

AVIP では、画像生成プログラムや画像処理プログラムなどが複数同時に動作する。これらのアプリケーションプログラムはプロセッサノード上の複数のプロセッサで並列動作し、さらに数十個のプロセッサノード間でデータ通信を行ないながら並列動作する。

これらのアプリケーションを動作させるためのプラットフォームとして、疎結合型マルチプロセッサを対象とした並列オペレーティングシステム AVIP-OS の開発を行なっている。

OS の構成を決定するにあたり、AVIP でプログラムを実行するための要件として、1) プログラミングの容易性 2) ノード内の並列処理の記述 3) ノード間の並列処理の記述 を上げた。これらを満たすために、AVIP-OS のベースとして CMU の Mach3.0 を採用した。AVIP-OS は各ノードごとに動作するマイクロカーネルとあるノードで動作するシステムサーバ群から構成され各々が協調動作する。

本稿では、ノード間の並列処理の記述を行なう機能として、

(1) 複数ノードへのプロセス生成
(2) 複数ノード間のデータ共有
(3) 複数ノード間の同期処理、排他制御

について述べ、その実装について述べる。

2 システムの概要

2.1 AVIP のアーキテクチャ

AVIP のアーキテクチャを図1に示す。

AVIP は数個〜数百個のプロセッサユニットからなり、各プロセッサユニットは格子状に連結されトーラス状になっている。この結合方式を採用した理由は、

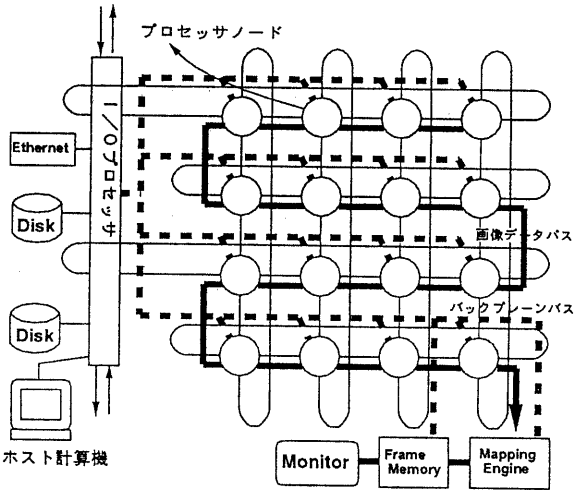

図 1: AVIP のアーキテクチャの構成図

- 柔軟な拡張性
- 一様性
- 画像生成・処理を行なうアルゴリズムとの対応
- 構成のしやすさ
- ハードウェア量が少ない
- 高速データ転送

という点からである。

これらのプロセッサユニットは画像データパスにより連結されており、画像データを高速にフレームバッファへ出力する。また、プロセッサユニットはホスト計算機のディスクとは別に高速バスでつながれたI/Oプロセッサを通してハードディスクへのアクセスが可能である。さらに、ラック内のプロセッサユニットに対し共通のデータをブロードキャストしたり、プロセッサ間の同期をとるのに用いるバックプレーンバスが存在する。

2.2 プロセッサユニットの構成

AVIP のプロセッサユニットの構成図を図2に示す。4つの i860XP とキャッシュ、通信プロセッサと4つの FIFO、ローカルメモリ (64MB)、グラフィックスハードウェアからなる。

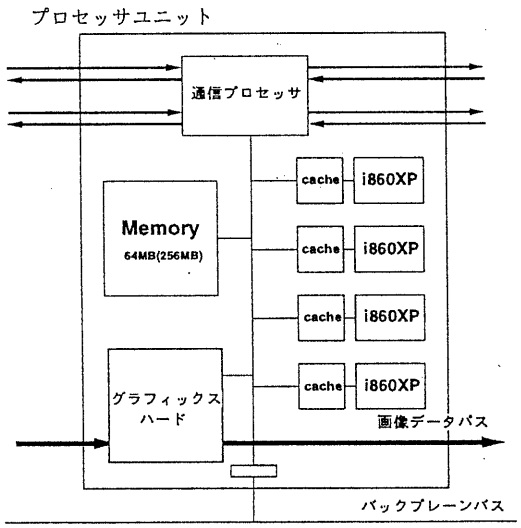

図 2: AVIP のプロセッサユニット

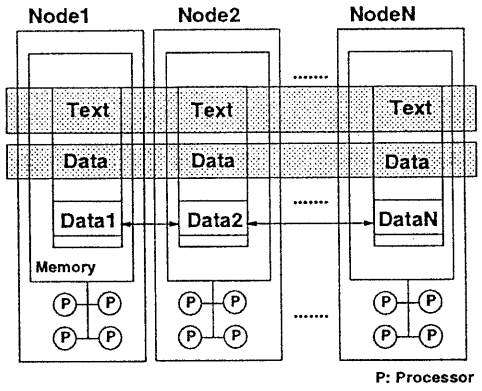

図 3: AVIP のプログラミングモデル

3 OS の構成

3.1 プログラミングモデル

グラフィックスアプリケーションを AVIP で動作させたとき、プログラムのテキストとデータの配置とその処理の観点から次の二つのプログラミングモデルがある。

- 各ノードは、同一のテキストを、またデータは分割して保持する。このデータの処理を各ノード内の 4 つのプロセッサが並列に処理する。必要に応じてノード間で同期をとる。
- 各ノードは、同一のテキストを、またデータは分割して保持するものと、各ノードごとに保持するものがある。このデータの処理を各ノード内の 4 つのプロセッサが並列に処理する。必要に応じデータを他ノードに転送する。(図 3)

3.2 設計方針

AVIP に OS を実装するにあたり、以下の設計方針をあげる。

1. プログラミングの容易性

   通常我々がプログラミングを行なっている UNIX ワークステーションと同じプリミティブが使え、ユーザが容易にプログラミングを行なえるようにする。

2. ノード内で並列処理の記述

   ノード内の 4 つのプロセッサに対し、ユーザが並列処理の記述を容易に行なえるようにする。

3. ノード間での並列処理の記述

   データパラレル性のあるプログラムを複数ノード間で、同時に動作させる。そのために、複数ノードへのプロセス生成、複数ノード間でのデータの共有、メッセージ通信、同期処理、排他制御を実現する。

3.3 構成

AVIP-OS は CMU の Mach3.0 をベースにしている。

Mach を採用した理由は、以下のことがあげられる。

1. マルチプロセッサに対応していること

   通常 UNIX ようなシングルプロセッサ対応の OS では、カーネルモードで動くプロセスは、ただひとつとしておりカーネル内のきわどい資源の排他制御は、割り込みレベルで管理している。

マルチプロセッサに対応したMachでは、各資源ごとにロック変数を持ち、排他制御が行なわれている。

2. マイクロカーネルであること

マシン依存の部分とそうでない部分が切り分けられており移植性が高い。また、新しい機能を追加する場合に、システムサーバとして実装することができ、変更が容易である。

Machでは、クライアントタスクとサーバタスク間のインターフェイスを生成するMIG(Mach Interface Generator)が用意されている。これを使うとシステムサーバを比較的容易に構築できる。

3. メッセージ通信をベースとしていること

AVIP-OSは、マルチプロセッサ構成のノードが疎結合されており共有メモリをもたない。したがってノード間はメッセージを使った通信を行なう。

Machは、NORMA(NO Remote Memory Access)モデルに対応しており、リモートホストのプロキシーポート(Proxy port)を得ることにより、ローカルホストと同様にリモートホストのカーネルプリミティブを起動することができる。

4. カーネルデバッガーの機能が豊富なこと

マイクロカーネル自体にデバッグの機能が組み込まれており、タスク、スレッドの状態や、タスクごとの仮想アドレス空間、各ポートなどの状況を詳細に見ることができる。また、ブレイクポイントの設定、実行などが容易にできる。

AVIP-OSは、大きく3つの構成要素からなる。その構成を図4に示す。

1. マイクロカーネル

各ノードに配置され、ノード内のスレッドのスケジューリング管理、メモリ管理、プロセス間通信をサポートする。

2. システムサーバ群

ある一つのノードで動作し、UNIXのシステムコールをサポートするUNIXサーバや、次節で述べる機能を実現するための管理サーバがある。

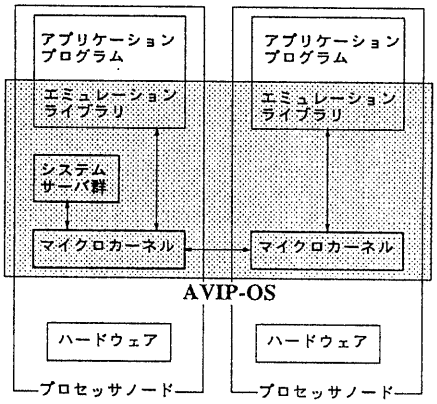

図4: AVI-OSの構成

3. エミュレーションライブラリ

各ユーザプロセスは、UNIXシステムコールを実現するために、仮想アドレス空間の一部にエミュレーションライブラリを置く。

4 OSの機能

AVIP-OSは、Mach3.0をベースにしているので以下の機能を持つ。

1. UNIXと同等の機能

4.3BSD UNIXと互換性を持った環境を提供するため、プログラミングが容易である。

2. ノード内並列処理

プログラムはプロセスとして実行される。プロセスは仮想アドレス空間やポート名などを持つタスクと実行の単位であるひとつ以上のスレッドからなる。ひとつのプロセス内でスレッドを複数起動し、ノード内の各プロセッサごとにスレッドを動作させ、ノード内で並列処理を実現する。

3. NORMA対応

他のノードのプロキシーポートを得ることにより、自分のノードと同様に他のノードのカーネルプリミティブを起動することができる。

本章では、AVIP-OS での複数ノード間で並列処理を行なうための機能について述べる。

4.1 複数ノードへのプロセス生成

4.1.1 機能

AVIP-OS で、新たにプロセスを生成する方法は UNIX と同様に fork() である。が、これは同一ノードにのみプロセスを生成し、他ノードにはプロセスを生成しない。そこで、ひとつのアプリケーションを複数ノードで並列動作させるために、複数のノードへ同時にプロセスを生成できる機能を持たせた。

他ノードにプロセスを生成する際に、プロセスの仮想アドレス空間を継承する。通常は、コピーオンライトで他ノードに仮想アドレス空間が作成される。ところで、AVIP-OS では、vm_wired() を拡張して、プロセスの仮想アドレス空間をすべてメモリにはりつける (wired) という機能 (vm_wired_all()) を持っている。この vm_wired_all() によりすべての仮想アドレス空間が wired になっているプロセスは、ダイレクトに仮想アドレス空間がコピーされる。

4.1.2 プリミティブ

- mfork(listlen, nodelist, pidlist)

mfork() は nodelist で指定されたノードにプロセスを生成する。新しいプロセス（子プロセス）は呼び出しプロセス（親プロセス）の資源を継承する。子プロセスの PID(ProcessID) はシステム全体で一意になるように付けられ、親プロセス PID は親プロセスの PID が付けられる。親プロセスは pidlist に生成された各プロセスの pid のリストが返される。子プロセスの pidlist の中身は無効である。ひとつでもプロセスの生成が失敗するとエラーとなり、子プロセスは生成されない。また、nodelist が指定されない場合はエラーとなる。

4.2 複数ノード間のデータ共有

4.2.1 機能

AVIP はノード間では、共有メモリをもたない。したがってノードにまたがるアプリケーションを動作させた場合に、データを互いに参照する機能が必要である。

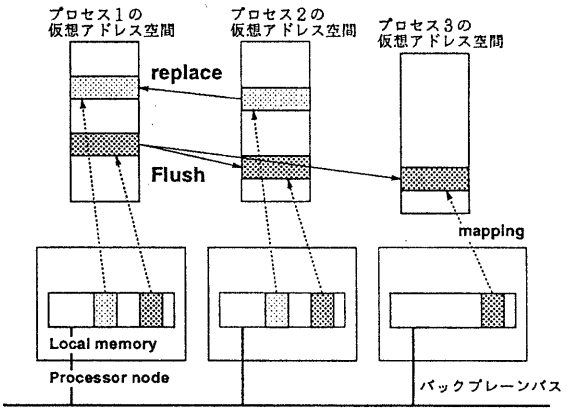

図 5: 物理メモリの動作

そこで、ノードにまたがるプロセス間で共通に扱うデータを読み書きできるようにするために、物理メモリを定義する。

物理メモリは複数のノード内で、同じ実アドレスと大きさをもった実メモリ領域の組（グループ）に対応する。

物理メモリにデータを読み書きするために、各ノードでプロセスの仮想アドレス空間に物理メモリをマップする。ユーザはマップした領域に読み書きを行ない、必要に応じてこの領域(物理メモリ) に対して、フラッシュ(全物理ノードに一斉転送)、リプレイス（あるノードの物理メモリに置き換える）の操作を行なうことにより、疑似的な共有メモリをサポートする。

ユーザが陽にフラッシュ、リプレイスを行なうことができるために、完全な共有メモリシステムに比べ余計なデータ転送は発生しない。

4.2.2 プリミティブ

- phys_create(),phys_open(),
  phys_close(),phys_delete()

  物理メモリを作成、オープン、クローズ、削除する。
- phys_map()

物理メモリをユーザのアドレス空間にマップする。

- phys_replace()

  アドレス空間にマップされた物理メモリを指定されたノードの物理メモリで置き換える。

- mflush()

  アドレス空間にマップされた物理メモリ領域を指定されたノードに転送する。

- mread()

  アドレス空間にマップされた物理メモリ領域にファイルを読み込み、指定されたノードに転送する。

4.3 ノード間の同期、排他制御

4.3.1 機能

複数ノードでそれぞれ動作するプロセス間で同期処理や、ある資源の排他制御を行なう機能をもたせた。

AVIP ではこれらをノード間で高速に実行するために、同期処理、排他制御のためにレジスタセットが用意されている。AVIP-OS はこのレジスタセットを使って、ハードウェアバリア、ハードウェアイベント(高速同期処理)、ハードウェアロック(排他制御)を行なう。

4.3.2 プリミティブ

- barrier_create(),barrier_open(), barrier_close(),barrier_delete()

  ハードウェアバリアを作成、オープン、クローズ、削除する

- barrier_wait()

  ハードウェアバリアによる待ち合わせを行なう

- barrier_ready()

  ハードウェアバリアで待っているものを起こす

  ハードウェアバリアには、WAIT モードと READY モードがあり、WAIT モードの場合には、create 時に指定されたノード全てが barrier_wait() を呼ぶまで、wait する。READY モードの場合は、create 時に指定されたノード全てが barrier_wait() を呼び、かつあるプロセスが barrier_ready() を呼ぶまで wait する。

- event_create(),event_open(), event_close(),event_delete()

  ハードウェアイベントを作成、オープン、クローズ、削除する

- event_wait()

  ハードウェアイベントによる待ち合わせを行なう

- event_ready()

  ハードウェアイベントで待っているものを起こす

  ハードウェアイベントには、WAIT モードと READY モードがあり、WAIT モードの場合には、open 時に WAIT モードを指定した場合にのみ使用でき、event_ready() が実行されるまで、wait する。READY モードの場合は、open 時に READY モードを指定した場合にのみ使用でき、event_wait() で待っているプロセスを全て起こし、スタートさせる。

- lock_create(),lock_open(), lock_close(),lock_delete()

  ハードウェアロックを作成、オープン、クローズ、削除する

- p_lock()

  ハードウェアロックのロック、ロックをかけることができるまで wait する。

- cp_lock()

  ハードウェアロックのロック、ロックをかけれないときは何もしない。

- v_lock()

  ハードウェアロックのアンロック

5 OSの実装

5.1 mfork()の実装

mfork()は、UNIXのシステムコールとして実現した。mfork()が呼ばれると一旦マイクロカーネル内に飛び込んだあと、エミュレーションライブラリ内に戻り、ここで子プロセスが次に動作すべきコンテキストを作成したのち、UNIXサーバにメッセージを送る。

UNIXサーバは、親プロセスと同じ仮想アドレス空間をもったプロセスを指定されたノードに生成する。仮想アドレス空間の作成には、NORMAのメモリ管理を使ったコピーオンライトによる方法と、ダイレクトに全ての仮想アドレス空間をコピーする方法の2種類がある。vm_wired_all()により仮想アドレス空間がすべてwiredになっている場合は、後者の方法により作成する。

5.2 物理メモリの実装

5.2.1 物理メモリの獲得、解放

物理メモリは、複数のノード内で同じ実アドレスと大きさをもった実メモリ領域の組であり、システムでユニークになるように名前で管理されている。名前の管理は、ネームサーバ(snames)に依頼する。

各ノードでは、同じ実アドレスと大きさを持つ実メモリ領域を管理するために、仮想メモリのための仮想メモリページとは別な領域をOSの立ち上がり時にあらかじめ確保しておき、その領域内で物理メモリの獲得、解放を行なっている(physmem_allocate(), physmem_deallocate())。物理メモリを獲得すると物理メモリオブジェクトへのポートを得ることができる。

5.2.2 物理メモリ管理サーバ

各ノードで獲得されている実メモリ領域の実アドレス、大きさ、物理メモリオブジェクトに対応するポートを管理するために、システムサーバとして、物理メモリ管理サーバがある。物理メモリ管理サーバは、phys_create()の要求により各ノードに物理メモリを作成し、phys_open()の要求により物理メモリを操作する権限(物理メモリオブジェクトに対するポート)を与える。さらに、phys_close()要求により物理メモリを操作できなくし、phys_delete()の要求により各ノードの物理メモリを解放する。

5.2.3 物理メモリのマップ

物理メモリ(実メモリ領域)はユーザの仮想アドレス空間にマップされアクセスされる。ユーザの仮想アドレス空間にマップする方法として、deviceとして実メモリを登録しdevice_map()により作成されたメモリオブジェクトをvm_map()を使いマップする方法があるが、ページフォルトを起こして初めてページテーブルに登録するので遅延が発生する。そこで、メモリオブジェクトの代わりに物理メモリオブジェクトを指定してvm_map()を使い、カーネル内でダイレクトにPMAPモジュールを呼んでページテーブルに登録する。

5.2.4 物理メモリの操作

mflush()、phys_read()は、バックプレーンバスを利用したブロードキャスト機能とI/Oプロセッサ間の高速ネットワークを利用してデータの転送を行なう。mread()は、UNIXサーバ内でファイルを読み込む際のバッファ領域に物理メモリをマップすることにより実現する。

5.3 同期、排他制御の実装

5.3.1 同期ワイヤー

ハードウェアバリア、ハードウェアイベント、ハードウェアロックはいずれもAVIPで用意されている、イベントスイッチレジスタ、イベントコントロールレジスタ、割り込み通知レジスタ、割り込みマスクレジスタを使用して実現される。イベントスイッチレジスタは8組用意されており、残りのレジスタはその8組に対応したビットを持っている。したがってハードウェアとしては8組のイベント処理を実現でき、そのハードウェア1組を同期ワイヤーと呼ぶ。

5.3.2 同期ワイヤー管理サーバ

ノード間で同期ワイヤーを使って、バリア、イベント、ロックを実現する場合、同じ組の同期ワイヤーを使用しなければならない。そこで、各ノードで使用す

る同期ワイヤーのイベントスイッチレジスタ番号を管理するために、システムサーバとして、同期ワイヤー管理サーバがある。同期ワイヤー管理サーバは、各 create() の要求により各ノードに同期ワイヤーを獲得し、各 open() の要求により同期ワイヤーを操作する権限を与える。さらに、各 close() 要求により同期ワイヤーを操作できなくし、各 delete() の要求により各ノードの同期ワイヤーを解放する。

5.4 マルチプロセッサ対応

マイクロカーネルのマルチプロセッサの対応を行う部分は、マシン依存の部分である。以下のことを実現する。

5.4.1 スレーブプロセッサの立ちあげ

マスタープロセッサが立ち上がったあと、スレーブプロセッサにリセットをかけて立ち上げる。OS の初期化処理はマスタープロセッサが行なっているので、スレーブプロセッサは、自分自身の初期化を行なう。

5.4.2 プロセッサ間通信

プロセッサ間通信は他のプロセッサにある事象を通知するときに使う。AVIP ではソフトウェア割り込みを使って実現し、次のふたつの場合に行なう。

ひとつは、あるプロセッサが TLB の更新を行なったときに、それを使っているプロセッサに知らせ、TLB をフラッシュさせる場合と、もうひとつはソフトウェアクロック割り込みを行なう場合である。

6 現状

6.1 仮ターゲットでの実装

AVIP 実機が実際に動作するまで、OS の機能を実装できない。今回の機能は大部分が非マシン依存であるので、仮ターゲットを設定しその上に OS の機能を実装した。

仮ターゲットは、マルチプロセッサ (i386 × 3) である BE ワークステーションを2台、Ethernet で接続したものとした。1台（マスターノード）でシステムサーバ群とマイクロカーネルを動作させ、もう1台（スレーブノード）ではマイクロカーネルのみ動作させた。

実機ではスレーブノードは、ディスクを持たないのでデフォルトページャはマスターノードのデフォルトページャに設定している。

mfork() と物理メモリ管理サーバは、仮ターゲット上で動作を確認した。また、マルチプロセッサ対応を行なった。

6.2 実機上での現状

マシン依存部 (マルチプロセッサ対応を除く) の開発を終え、ノード2台（シングルプロセッサ）で AVIP-OS が動作中である。

NORMA の通信部は、現在は通信プロセッサを使わずにバックプレーンバスを使った実装がなされている。

今回加えた機能は、mfork() が動作確認ができ、他の機能については現在実装中である。

7 おわりに

AVIP の複数ノード間でプログラムを実行するための機能と、その実装状況を述べた。

今後はこの OS 上でアプリケーションプログラムを実際に動作させ、OS の機能拡張を行なっていく予定である。

参考文献

[1] Mike Accetta, Robert Baron, David Golub, Richard Rashid, Avadis Tevanian, and Michael Young. "*Mach: A New Kernel Foundation for UNIX Development.*" Technical Report, School of Computer Science, Carnegie Mellon University, Pittsburgh, August 1986. Also in Proceedings of the Summer 1986 USENIX Conference, pp. 93–112

[2] David Golub, Randall Dean, Alessandro Forin, and Richard Rashid "*Unix as an Application Program*" Proceedings of the Summer 1990 USENIX Conference, pp. 87–95