共有メモリベースのシームレスな並列計算機環境を実現するオペレーティングシステムの構想

青木 秀貴[†]　山添 博史[††]　五島 正裕[†]　森 眞一郎[†]　中島 浩[†]　富田 眞治[†]

[†] 京都大学大学院工学研究科
[††] 京都大学工学部

本稿では、ノード間で緊密な協調動作をおこなう並列アプリケーションの実行を可能とする分散システムとして「コンピュータ・コロニー」を提案する。我々はコンピュータ・コロニーの一実現として、高速なリモートメモリアクセスを実現する通信ボードを付加した共有メモリベースのワークステーションクラスタと、その上で動作する分散 OS Colonia を提案する。Colonia ではミッション-ユニットモデルと呼ぶプログラミングモデルを採用し、並列アプリケーションの効率的な実行を支援する。

An Operating System for Seamless Parallel Computing Environment Based on Shared Memory

Hidetaka AOKI[†], Hiroshi YAMAZOE[††], Masahiro GOSHIMA[†],
Shin-ichiro MORI[†], Hiroshi NAKASHIMA[†], Shinji TOMITA[†]

[†] Graduate School of Engineering, Kyoto University
[††] Faculty of Engineering, Kyoto University

In this paper, we propose the computer colony as a distributed system for parallel processing. As an implementation of the computer colony, we are developing a cluster of workstations which communicates with each other by shared memory implemented by additional communication boards that enable fast remote memory access. We are also developing a distributed operating system, referred to as Colonia, for the cluster. Colonia adopts a programming model calld the mission-unit model for efficient execution of parallel applications.

1　はじめに

安価なワークステーションやパーソナルコンピュータの高性能化が進むにつれ、それらを LAN (Local Area Network) で結合した分散システムが広く普及してきている。しかし現在のところ、この環境はデータの共有や周辺機器の共有のプラットホームとして利用されているに過ぎない。将来的にはさらに、プロセッサ資源やメモリ資源などの計算パワーを共有することで、システム全体としての処理能力の向上と、並列アプリケーションの効率的な実行が期待される。これは次のようなものである。

分散システムの規模は大きなものでコンピュータ数百台から数千台にも及ぶが、ネットワーク上の計算機資源、特にプロセッサ資源やメモリ資源の利用率は一般にかなり低い。分散システム上のすべての計算機に負荷がかかっているような状況はあまり存在せず、また時間帯によってはほとんどの計算機資源が利用されないままとなっている。これらの遊んでいる計算パワーを有効に利用すれば、システム全体としての処理能力を向上できるはずである。ただしその実現にあたり、シームレスあるいは位置透過といったキーワードを欠かすことはできない。あらゆる資源を位置透過に利用できて初めて、計算機資源の有効利用が可能となる。

また、システム中の多数のプロセッサ資源を利用することで、並列化された種々のアプリケーションプログラムの効率的な実行が期待される。今日の並列処理は、並列計算機の上で、科学技術計算やデータベース処理などの特定のプログラムを実行するに留まっている。しかし将来的には、ワープロソフト, 表計算ソフトなどの日常的なアプリケーションソフトや、CAD, コンパイラなどのエンジニアリング用途のソフトなども並列化されるようになり、分散システムの上で実行されるようになると考えられる。

ただし、上記のようなネットワーク中の計算機資源の共有に重点を置くあまり、リアルタイム性への対応が疎かになってはならない。特にマルチメディアは、情報伝達の手段としてあるいはユーザインタフェースとして必須技術であり、それを実現するためのリアルタイム性の保証は、システム構築における重要な課題である。

このように、将来のコンピュータ利用を考えた時、並列アプリケーションの実行を可能とし、さらにはマルチメディアにも対応し得るシームレスなマルチユーザ環境を構築することが重要であると言える。そこで本稿では、分散システムを拡張したコンピュータ・コロニー (computer colony) と呼ぶ環境を提唱する。

2　コンピュータ・コロニー

2.1　定義

コンピュータ・コロニーとは並列アプリケーションを指向した分散システムであり、オペレーティングシステム (Operating System, OS) を含めた総合的な環境を指す。

コロニー (colony) とは「群体」であり、多数の個体が集まった上で相互に協調し、全体であたかもひとつの個体であるかのようにふるまう生物をいう。コンピュータ・コロニーでは、システムを構成するノードが緊密に協調動作し、全体を構成する。ノードは物理的には分散しているが、機能的には完全にネットワーク透過であり、ユーザの視点からはコンピュータ・コロニーはひとつの巨大な計算機資源としてのみ捉えられる。コンピュータ・コロニーはマルチユーザで利用され、各ユーザはその上で並列アプリケーションを効率よく実行することができる。

2.2　位置付け

コンピュータ・コロニーは、分散システムをベースとしたシステムである。マルチユーザによる並列アプリケーション実行のためのプラットホームとしては分散システムの他に、集中システムによるもの、すなわち巨大な計算パワーを有する1台の並列計算機を中心に構成され、端末を通じて複数ユーザが利用する環境が考えられる。

両者を比較した時、分散システムには開発者およびユーザにとって多くのメリットがある。まず開発者にとってのメリットとしては、ハードウェアの開発コストが低いというものが挙げられる。またユーザにとっては、システムの導入コストが低く、システムの拡張性および可用性に優れるというメリットがある。また、UNIX などのネットワーク OS を利用した従来型の運用も可能であり、ソフトウェア資産の継承も望める。これらのすべてが、ユーザにとって導入のきっかけとなり得るものである。

しかしその一方で、並列アプリケーションの処理能力では並列計算機に劣るのもまた事実である。分散システムにおける要素マシン間の通信をどれだけ高速にしても、並列計算機におけるプロセッサ間通信はさらに高速である。並列処理をおこなう上で通信の性能は重要な要素であり、特に細粒度並列処理において、並列計算機の方が高い能力を発揮する。

分散システムの持つこの欠点を克服し、並列アプリケーションの実行能力を実用レベルにまで高めるのが、コンピュータ・コロニーである。コンピュータ・コロニーにより、分散システムの持つ導入の手軽さ、運用の柔軟さなどの利点を生かしつつ、従来の分散システムでは困難とされてきた高性能を達成することができる。

ただし、コンピュータ・コロニーが並列アプリケーションの実行を可能にすると言っても、それでもなお、並列計算機の方が高速である。しかし、並列数値計算などの大規模並列処理自体が目的でない場合、すなわち通常の並列アプリケーションを実行することが主目的である場合には、手軽に利用できる並列処理実行環境として、コンピュータ・コロニーは魅力ある環境であると考える。

2.3　コンピュータ・コロニーの要件

コンピュータ・コロニーの実現には、いくつかの要件がある。

まず、並列アプリケーションの実行を可能とするため、高速・低遅延の通信機構をユーザに解放できなければならない。現在の分散システムでは、通信がボトルネックとなり、並列処理 (特に細粒度処理) の実行が大きく阻害されると同時に、システムの利用形態自体に強い制約を課されている。通信の遅さは主に、各マシンのネットワークインタフェースが低速な I/O バスに接続されていることに起因している。これにより、I/O バスがデータ転送のボトルネックとなる、資源の保護のため通信時に必ず OS が介在するなどの問題が生じていた。

また、システム全体の資源の利用率を上げ、それによってシステム全体として出来る限り高い性能を得るため、プログラムのネットワーク透過な実行を支援しなければならない。ここで、位置透過なアクティビティの実行機構や、アクティビティの動的な移送が必要となる。アクティビティから利用される資源は、アクティビティの実行位置に依らずアクセスできなければならない。また、並列アプリケーションにおける複数アクティビティ間の通信も、動的な実行位置の変化に柔軟に対処し得るものでなければならない。

並列アプリケーションの作成・実行に適したシステムイメージやプログラミングモデルの提供も、プログラム作成を支援するものとして欠かせない。柔軟で容易な並列プログラミングを可能とするため、自然な並列プログラムの記述を支援するための通信機構や、通信や処理の粒度を気にすることなく並列化するための高速なアクティビティ生成機構などの提供が重要であると考える。

3 共有メモリによるコンピュータ・コロニーの実現

コンピュータ・コロニーの満たすべき種々の要件は、共有メモリをベースとすることで次のように自然に達成することができる。

まず、共有メモリをベースとした通信では、MMU (Memory Management Unit, メモリ保護ユニット) によるメモリ保護機構を利用することで、通信機構をユーザに対して安全に解放することができる。通信機構が一度ユーザに解放されると、個々の通信に対してはOSの介在を必要としないため、高速・低遅延の通信が達成される。さらに、この共有メモリをベースとした通信方式は、様々なアーキテクチャに対して一般化しやすいという特徴を持っている。

また並列プログラムを記述する際、共有メモリを利用することで、ユーザはオブジェクトの位置やサイズを気にする必要がなくなる。さらに、共有メモリを利用した資源へのアクセス, 共有メモリを利用したアクティビティ間の通信をおこなうことで、位置透過なプログラムの実行を自然に実現できる。

我々は共有メモリによるコンピュータ・コロニーの実現例として、共有メモリベースのワークステーションクラスタというハードウェア環境と、そのためのオペレーティングシステム **Colonia** を提案する。

ただし、今までコンピュータ・コロニーが共有メモリによって実現されると述べてきたが、実際には、共有メモリこそがコンピュータ・コロニーの実現における本質だと我々は考えている。

共有メモリをベースとしたシステムは、そもそも位置透過性を本質としている。そこでは、プログラムで利用される物理的に分散した計算機資源が、ユーザのアドレス空間という論理空間内に配置される。ユーザから見ると、必要な資源がすべて自身のアドレス空間内に用意されており、それらを単一のイメージで、しかもOSの余計な介在なく自由に利用することができるのである。

これに対しメッセージ交換型のシステムは、計算機資源の独立性を本質としている。計算機資源の独立性は、分散システムにおける資源の分散ときれいに対応付けられるものであるが、そこには独立性を維持するためのOSによる高い壁が存在している。

コンピュータ・コロニーにおける資源とプログラムの密接な関係、およびアクティビティ同士の間の密接な関係を効率よく実現する上で、共有メモリは必要不可欠であると考えている。

3.1 共有メモリベースのワークステーションクラスタ

共有メモリによるコンピュータ・コロニーのハードウェア環境の実現のひとつとして、従来のマシン (ワークステーションおよびパーソナルコンピュータ) に高速な通信を提供する専用のハードウェアを付加し、それによって共有メモリベースのワークステーションクラスタ (および PC クラスタ) を構築することを提案する。具体的には、クラスタを構成する各要素マシンのプロセッサバスに通信ボードを接続する。その大まかな構成を、図1に示す。なお、すべての要素マシンがディスクや従来ネットワークへのインタフェースを有する必要はない。

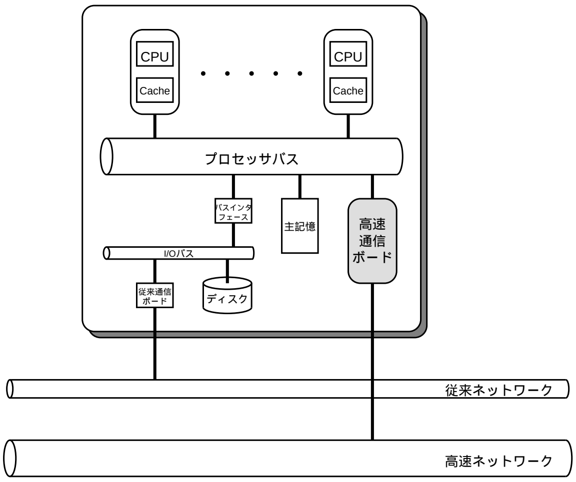

図 1: 要素マシンの構成

この通信ボードの基本的な機能は、ローカルマシンの物理アドレスとリモートマシンの主記憶を対応付けることである。すなわち、メモリアクセス命令発行時にプロセッサバスを流れる物理アドレスを拾い、必要に応じてリモートメモリアクセスという形の通信を実現するものである。

ここで、ユーザプログラムにおけるメモリアクセスは、仮想アドレスに対しておこなわれるという点に注意する。そのため、プログラムから本通信ボードを利用してリモートマシンとの通信をおこなうには、仮想アドレスから物理アドレスへの対応付けと、物理アドレスからリモートマシンの主記憶への対応付けを、あらかじめ設定しておく必要がある。

しかし一度そのコネクションを確立しさえすれば、あとは通常のメモリアクセスによって通信をおこなうことができる。具体的には、次のような手順となる。

1. ユーザは仮想アドレス空間にアクセスする。
2. MMU が仮想アドレスを物理アドレスに変換し、プロセッサバス上に流す。
3. 通信ボードが物理アドレスを拾い、他ノードとの通信の必要性を検知すると、実際の通信処理をおこなう。

個々の通信にはMMUによる仮想アドレスから物理アドレスへの変換が含まれており、これにより通信ボードは、通常の主記憶と同じ形で保護されている。保護の問題がMMUによって解決されるため、通信機構をマルチユーザに対して安全に解放することができる。その結果ユーザは、OSの介在しない高速・低遅延の通信を利用することが可能となっている。

高速通信用の通信プロトコルも、従来との互換性を意識したTCP/IPベースのものである必要はなく、またそれほど高機能である必要もない。なぜなら高速ネットワークはローカルな環境内で閉じたものであり、外部とのTCP/IPによる通信は、I/Oバスを経由した従来ネットワーク上で提供すればよいからである。

この通信ボードにより、すべてのユーザが高速・低遅延の通信機構を利用可能となり、中粒度あるいは細粒度の並列処理への道が開かれる。従来からのハードウェア資産を生かし、高性能な並列計算システムを低価格で実現できるということも大きな利点である。

3.2 分散OS Colonia

共有メモリベースのワークステーションクラスタというハードウェア環境内の各種計算機資源を管理し、その上に共有メモリベースのシームレスなマルチユーザ並列環境を実現するのは、ソフトウェア環境であるColoniaの役割である。

Coloniaの動作するハードウェア環境としては、次の3点を想定している。

- ハードウェア環境は非均質であると想定している。具体的には、各要素マシンの有するプロセッサの計算能力および搭載プロセッサ数、主記憶量、ディスクの有無およびディスク量などに、多様性があるとする。また、マシン間の通信距離も一般に非均質であると仮定する。
- 要素マシン間のリモートメモリアクセスにおいて、ハードウェアによるマシン間のキャッシュ整合性制御はあってもなくても構わない。なぜなら、ソフトウェアによる実現も可能だからである。ただし、厳密に速度を要求する環境では、ハードウェアによる支援がある方が望ましい。
- 命令セットの異なるプロセッサが混在するコンピュータ・コロニーを考えることもできる。ただしその場合、プログラムのコード領域を共有できないことを主な理由として、システムの運用に制約が生じてくる。そこでColoniaではさしあたって、オブジェクトコードレベルでの互換性を有するプロセッサで構成されたシステムを想定する。

Coloniaの果たすべき役割のうち最も基本となるのは、共有メモリベースの並列計算環境を実現するための次の2点である。

- 物理的にメモリを共有していないノード間で、共有メモリを実現する。
- その上で、任意のアクティビティ(Coloniaではユニットと呼ぶ)間でのメモリ空間の共有を可能にする

これらを実現した上で、さらにシームレスな並列計算環境を構築すべく、以下の項目の達成を目指す。

- 位置透過性の実現 ··· ユーザに計算機資源の物理的な位置を意識させない。さらには、プログラムの実行位置さえも意識させない。
- 並列アプリケーションの支援 ··· 並列アプリケーションの効率的な実行を支援するための、プログラミングモデルと実行メカニズムを提供する。
- マルチユーザ環境の実現 ··· システム内のすべての計算機資源を管理し、マルチユーザに提供する。その際、各ユーザが各種資源に対して要求するQoS (Quality of Service)を出来る限り満足させる。ただしマルチユーザ環境である以上、不当であると判断された要求は受理されない。
- システム全体としての性能の向上 ··· マルチユーザの利用において、システム全体の資源の利用率を上げ、それによってシステム全体として出来る限り高い性能を得ることを目標とする。

4 Coloniaの概要

4.1 システム構成

Coloniaは、いわゆるマイクロカーネル方式の構成を取る。すなわち、マイクロカーネルがプログラムの実行と通信に必要となるごく基本的な機能のみを提供し、それ以外のOS機能は複数のシステムサーバによって実現される。マイクロカーネル方式による構成の利点は多く知られているが、その中でも、カーネルのオーバヘッドの時間を束縛(bound)できるという性質が重要である。これはリアルタイム処理の実現において本質的な部分であり、マルチメディアへの対応に欠かせないからである。

Coloniaカーネルは、以下の基本概念を提供する。

- ミッション ··· 実行中のプログラム、あるいはユーザがコンピュータシステムの中に立てる代理である。ミッションはひとつまたは複数のユニットから成る。
- ユニット ··· アドレス空間の単位であると同時に、逐次実行の単位である。ひとつのユニットは、必ずひとつのミッションに属する。ユニットはノードに対して割り付けられ、その中で実行される。
- 共有メモリ ··· 同一ミッションあるいは異ミッションを問わず、任意のユニット間での共有メモリを提供する。
- シグナル ··· 任意のユニットに対して非同期事象を通知する。

なお、Coloniaは物理的な資源の位置を次の単位で管理する。

- プロセッサ ··· アクティビティのディスパッチはプロセッサに対してなされるが、それ以外に個々のプロセッサを意識することはない。
- ノード ··· ノードとは、物理的にメモリを共有する、密結合された物理的資源の集合体である。具体的には、共有メモリベースのワークステーションクラスタにおける要素マシンである。
- テリトリ ··· テリトリとは、Coloniaによって管理されるノードの全集合である。

4.2　プログラミングモデル

Colonia では、共有メモリを利用可能な分散環境の中で並列プログラムの効率よい実行を実現するモデルとして、ミッション-ユニットモデルを採用している。Coloniaではプログラムをひとつのミッションとし、その中の並列性をユニットによって記述する。コードなどの共有可能なデータをミッション内で共有し、コンテクストはユニット毎に独立して持つことで、ユニットの高速な生成および移送を実現するものである。

4.2.1　ミッション-ユニットモデル

Colonia におけるプログラムの実行は、ミッション-ユニットモデルに基づいておこなわれる。これは次のようなものである。

- ユーザはシステムにミッションを投入する。ミッションとはユーザがコンピュータシステム内に立てる代理であり、システムに投入された一連の仕事をおこなうプログラムである。ひとりのユーザは、同時に複数のミッションを投入することが可能である。
- ミッションは、協調動作する複数のユニットよりなる。ユニットとはミッションの完了に不可欠な存在であり、それぞれが固有のアドレス空間を持ったアクティビティである。各ユニットはノードに対して割り付けられ、ノード内のプロセッサで実行される。
- ひとつのミッションに属するすべてのユニットは、ミッション生成時に指定したアドレス空間の固定領域を完全に共有する。また必要に応じて、ミッションの同異に依らない任意のユニット間に共有メモリを張ることも可能できる。
- ユニットは、ミッションおよびユニットを生成することができる。ミッションが生成されると、そのミッション内にルートユニットと呼ぶユニットが自動的に生成される。ユニットがミッションやユニットを生成していくことにより、ミッションおよびユニットの木が構成される。

4.2.2　アドレス空間

ミッション-ユニットモデルでは、アクティビティであるユニットがそれぞれ固有のアドレス空間を有している。ただしそのアドレス空間の一部は、同一ミッションに属するすべてのユニットとメモリを共有するのに利用される。この領域をミッション内共有領域と呼び、それ以外の領域をユニット固有領域と呼ぶ。ユニットのアドレス空間を図 2 に示す。

ミッション内共有領域　ミッション内共有領域は、ミッションの生成時に指定される。ミッション内に生成されたすべてのユニットは、そのミッション内共有領域を自身のアドレス空間の同じアドレス位置にマッピングする。ミッション内のユニットはその領域を完全に共有し、このメモリ共有領域ではポインタをそのまま使うことができる。

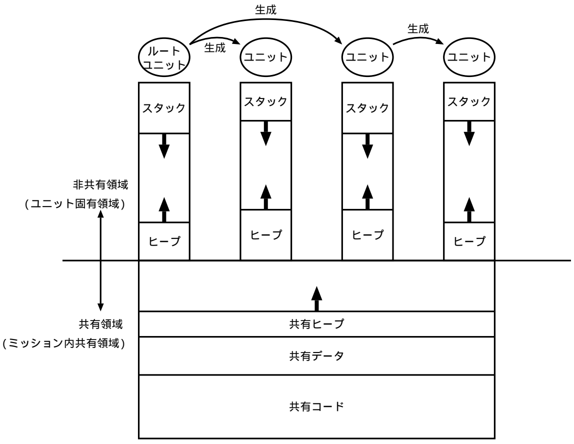

図 2: ユニットのアドレス空間

ユニット固有領域　ユニット固有領域は、ユニットが完全に自由に利用できる領域であり、スタックやユニットに固有なデータ空間として利用される。ユニット固有領域は、他ユニットから完全に保護される。そのため、他ユニットによってスタックが破壊される心配がないといった利点がある。

この領域にプログラムをロードし、ミッション内の他ユニットとは別のプログラムを実行することもできる。ただしこれはユニットの高速な起動や移送を阻害するものであり、あまり利用されるべきではない。

4.2.3　ユニット間の共有メモリ

Colonia では、ミッション内共有領域によるミッション内ユニット間の共有メモリと、エクスポート/マウント方式による任意のユニット間の共有メモリを提供する。

Colonia の提供する共有メモリで最も基本となるのが、ミッション内共有領域によるミッション内ユニット間の共有メモリである。この共有メモリは、コードやデータの共有に利用される。ミッション内共有領域はすべてのユニットの同じアドレス位置にマッピングされているため、この方式による共有メモリでは、ユニット間でポインタをそのまま渡すことができる。

さらに、エクスポート/マウント方式 (export/mount 方式) による任意のユニット間の共有メモリを提供する。この方式ではまず、あるユニットが自身のアドレス空間の一部をエクスポートするシステムコールを発行し、他ユニットからアクセス可能な領域として登録する。他ユニットはマウントシステムコールを発行し、エクスポートされている領域を自身のアドレス空間の任意のアドレス位置にマッピングする。このエクスポート/マウント方式の共有メモリでは、ミッションの同異に依らない任意のユニット間で共有メモリを張ることができる。なお、エクスポート/マウント方式で共有さ

れた領域は、すべてのユニットで同じアドレス位置にマッピングされているとは限らない。そのため、ミッション内共有領域による共有メモリと違い、ポインタは一般に利用できない。

4.2.4　ユニットの高速な生成および移送

ミッション内に新しいユニットを生成するには、まずミッション内共有領域の共有をノード間に設定し、その中のコードを利用してユニットを起動する。これにより、各ノードでプログラムをロードする手間を省くことができる。

また、あるノードで既にそのミッションに属する別ユニットが動いているなど、ミッション内共有領域が利用可能な状態にあれば、そのノードにおけるユニット生成はさらに高速におこなうことが可能となる。

ユニットの移送は、ユニットの新規生成時に必要なミッション内共有領域の準備に加え、移送元ノードから移送先ノードにユニット固有領域を移動する必要がある。ここでユニット固有領域を移送先へすぐにはコピーせず、移送元と移送先で共有を設定し、アクセス時にコピーをおこなうようにすることで、移送の効率を上げることができる。

4.2.5　ユニットの配置に関するプリファレンス

ユニットの配置は、プロセッサ資源, メモリ資源, 通信量などを基に、OS によって最適化される。例えば、頻繁に通信がおこなわれるユニット同士は、できる限り同じノードに配置される。ユーザがシステムにミッションを投入すると、ルートユニットを基点として次々とユニットが生成される。ユニットは OS が選択した最適なノードで実行され、また最適なノードへと動的に移送される。

OS によるユニット配置の最適化の前提があるため、下手にユーザが特定の位置にユニットを縛るべきではない。ユーザがユニットの位置を厳密に定めてしまうと、負荷の不均衡、テリトリ全体の計算機資源の利用率の低下、そしてプログラムの実行性能の低下などを引き起こす原因となる。

そこで、ユニットの位置やミッション内ユニット同士の距離に関して、プリファレンスという形でユーザが設定できるようにする。これはユニットの配置に関するユーザからの要望・提案であり、OS がどの程度それに従うべきかという優先度付きで指定される。ユニットの配置に関するプリファレンスは、OS による計算機資源割り当てのヒントとして利用される。他ユーザを含めたシステム全体の性能を阻害しない限りにおいては、OS は極力このプリファレンスに沿った資源割り当てをおこなうようにする。これにより、プログラムに関するユーザの知識を、OS が積極的に利用できるようになる。

なお、システムの提供するユニットなどで、位置を厳密に固定する必要があるものについては、最高優先度のプリファレンスを設定することで対処する。

4.3　通信

カーネルはユニット間通信のプリミティブとして、共有メモリとシグナルのふたつを提供する。

通常のデータの授受は、共有メモリを介しておこなう。共有メモリは、第 3.1 節に述べた専用の通信ハードウェアを利用して実現される。最初にユニット間に共有メモリを張る必要があり、この準備に多少コストがかかるが、load, store 命令による個々の通信は MMU によって完全に保護され、高速・低遅延の通信をユーザに対して安全に解放することができる。

しかし、共有メモリによる通信では、非同期に発生するイベントを効率よく伝達することができない。そこで、非同期イベントの効率のよい伝達手段として、シグナルを導入している。シグナルはそれほど大きくない固定長のデータを伴い、相手ユニットに対してネットワーク透過に伝達される。各ユニットはあらかじめ、自分に送られるシグナルの種類によってそれを受信するか無視するかを登録しておく。シグナルを受信する場合にはさらに、受信後に制御を移すシグナルハンドラを登録することができる。シグナルハンドラ内でおこなう処理は、ユーザの自由であり、例えば別ユニットを生成して処理の並列性を高めることも可能である。

共有メモリとシグナル以外の通信機構は、現在のところサポートする予定がない。メッセージなどを利用したい場合は、サーバやライブラリの中で共有メモリとシグナルを組み合わせることで実現する。

5　まとめ

本稿ではコンピュータ・コロニーを提唱し、その共有メモリベースの実現として、共有メモリベースのワークステーションクラスタと、その上にシームレスなマルチユーザ並列計算環境を提供する OS Colonia の構想について述べた。

現在、Sun Microsystems 社の SPARCstation 20 をノードとするシステムの構築を目指して、ハードウェアおよびソフトウェアの開発を進めている。

参考文献

[1] 前川 守, 所 真理雄, 清水謙太朗. “分散オペレーティングシステム UNIX の次にくるもの”, 共立出版, 1991.

[2] A. S. タネンバウム (引地信行, 引地美恵子 訳). “OS の基礎と応用 — 設計から実装, DOS から分散 OS Amoeba まで”, トッパン, 1995.

[3] J. ベーコン (藤田昭平, 篠田陽一, 今泉貴史 訳). “並行分散システム — オペレーティングシステム, データベース, 分散マルチメディアシステムへの統合的アプローチ”, トッパン, 1996.