数 値 解 析 37－2
（1991. 7. 17）

UNIX LAN環境における数値解析の分散処理

矢吹道郎
上智大学理工学部

畠山正行
茨城大学工学部

概要

UNIXネットワークの普及に伴い、ネットワークを利用した分散処理が実現可能となってきた。しかし分散処理実現のためのプロセス間通信インターフェースの利用は、数値計算ユーザなどにとっては容易でなく、複数マシン上で実行されるような数値計算の分散処理プログラミングの実現は困難である。本研究では、数値計算プログラマのためのプロセス間通信ライブラリ、および、それらライブラリからアクセスされる分散処理ユーティリティを作成し、並列性の明らかなモンテカルロ法によるシミュレーションに適用し実験を行なった。その結果、マクロな並列性を持つ数値計算アプリケーションにおいては有効な分散処理が可能となることが確認された。

DISTRIBUTED PROCESSING FOR NUMERICAL CALCULATION IN UNIX LOCAL AREA NETWORK ENVIRONMENTS

Michirou Yabuki
Sophia University
7-1 Kioi-cho, Chiyoda-ku, Tokyo 102, JAPAN

Masayuki Hatakeyama
Ibaraki University
4-12-1 Nakanarusawa-cho, Hitachi-shi, Ibaraki 316, JAPAN

Abstract

Distributing processing through networks is now widely available on UNIX workstation environments. For numerical calculation programmers, however, the inter-process communication facilities needed for distributing processing are not easy to use. Therefore, a numerical calculation distributed over several machines is also difficult. In this paper, a system for assisting inter-process communication is described. The system consists of libraries for inter-process communication and utilities accessible from the libraries. This system was used for a simulation by a Monte-Carlo method which has explicit parallelism. The result demonstrates that this system can be effectively used for numerical calculations that have macro-parallelism.

1 はじめに

ＵＮＩＸワークステーションならびにローカル・エリア・ネットワークの低価格化、高速化に伴い、ネットワーク環境が広く普及するようになってきた。そのため、多くの数値計算ユーザの環境もパーソナル・コンピュータあるいは大型機からＵＮＩＸネットワーク環境に移行している。ＵＮＩＸオペレーティング・システムにはネットワークを介してのプロセス間通信の機能が備っており（ソケット・インターフェース）、複数計算機を利用した分散処理が実現可能となっている。しかしプロセス間通信インターフェースは通信のためのプリミティブが容易されているにすぎず、システム・プログラマにおいてもその利用は容易ではない。そのため、数値計算ユーザが同時に複数の計算機を駆使して実効計算速度を向上させるようなアプリケーションを作成することは不可能に近い。

本研究では数値計算プログラマがソケット・インターフェースを意識することなくプロセス間通信を利用でき、また、単一計算機上で実効可能なソース・コードを大きく変更することなく利用可能な分散処理システムの構築を目指している。ここでは、特定の目的に限定せず汎用的利用が可能で、かつシステムが複雑とならないことを目標とした。本システムはプロセス間通信のための基本ライブラリ、および、ライブラリからアクセスされるユーティリティからなっており、特に大規模数値計算では必須の配列の利用を念頭において設計されている。

本論文では、構築したシステムの概要ならびに、モンテカルロ法によるシミュレーションに適用した実験結果、問題点、ならびに今後の展望について述べる。

2 システムの概要

2.1 システムの設計

本システムは数値計算アプリケーションの分散処理の支援を目的としている。そのため、

- プロセス間通信が容易に利用できること
- 汎用的な利用が可能であること
- システムが複雑でないこと

を目標としてシステムは設計を行なった。

容易なデータ通信の実現は、プロセス間通信のための基本ライブラリををユーザ・インターフェースとして用意し、UNIX のプロセス間通信のプリミティブであるソケット・インターフェースをユーザが意識せずに、データ通信を行なうことを可能とすることによって達成している。とくに、本システムでは数値計算に必須な配列の処理を念頭においているため、配列を単位としたデータ通信を可能とした。また、汎用性については、問題固有の制御あるいは処理を行なう部分と汎用的に利用可能なデータ管理を行なう部分に分離することによって実現している。

第三の目標であるシステムの単純さは、システムの機能とのトレードオフとなる。一般に分散処理では、並列性の認識ならびに通信の同期が重要となるが、本システムでは並列性の認識は行なっていない。処理の並列性をユーザ・プログラムから正しく認識することはきわめて困難であり、本研究では取り上げていない。従ってアプリケーションを作成するユーザは、問題固有の並列性を認識し、処理が分散されるようにプログラミングを行なう必要がある。

本システムは以上の目的を達成するため、以下の３つプロセスにより稼働するよう設計を行なった。

1. **データ管理プロセス**
   通信の支援を行ない、総合的なデータの管理を行なう
2. **制御プロセス**
   アプリケーションに固有なデータの制御ならびにデータ管理プロセスとクライアントプロセスとの通信の制御を行なう
3. **クライアント　・プロセス**(複数)
   実際のデータ処理（数値計算）を行なう

以上のプロセスはすべてユーザ・プロセスとして実行される。

各プロセス間の通信においては信頼性を考慮し、すべてストリーム・ソケットにより実現した。通信のオーバーヘッドについては、問題解決のための処理（実際の数値計算）に比べて大きな問題とはならないと考え、通信の効率は特に追及していない。

2.2 システムの構成

本システムは 2.1 節で述べたように、データ管理プロセス、制御プロセス、クライアント・プロセ

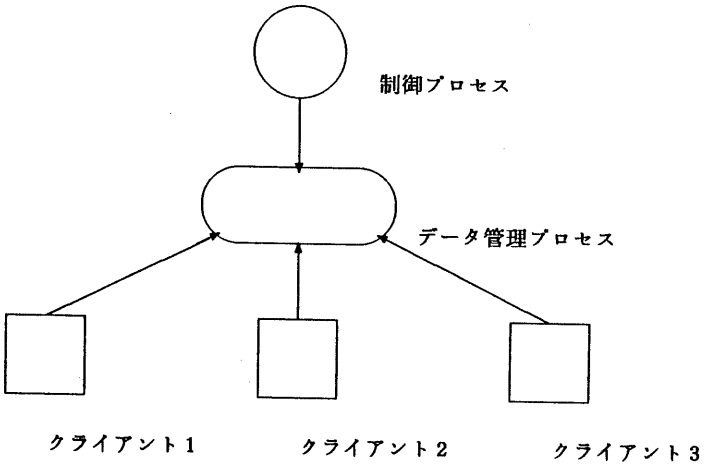

図 1: プロセスの関係

スの3つのプロセスから構成されている。各プロセスの関係を［図1］に示す。

図に示されている矢印はプロセス間の接続の関係であり、実際の通信路でもある。従って現在の仕様では、クライアント・プロセス間の通信、あるいは、クライアント・プロセスと制御プロセス間の通信はサポートされていない。

各プロセスの実行はすべて同じホストであっても、すべて異なるホストであっても構わない。以下に各プロセスの概要について述べる。

データ管理プロセス

データ管理プロセスがシステムの汎用性を担っている。すなわち、配列を主としたデータの集積、分散、通信を行なうプロセスで、概念的には共有メモリ管理プロセスとして捉えることができる。データ管理プロセスは受動的プロセスであり、制御プロセスによる制御、あるいは、クライアント・プロセスからの通信要求に従って処理を行なう。現在の仕様では制御プロセスおよびクライアント・プロセスとのデータ通信のための通信路はすべての処理が終了するまで保持される。すなわち、1プロセスに対して1つのソケット・ディスクリプタを消費していることになるため、データ管理プロセスに接続されるプロセスの総数はオープン可能なディスクリプタ総数以下に制限される。

管理されるデータはすべて配列として登録され、その大きさに応じてメモリ領域が確保される。そして、通常の変数名に対応する個別の名前が与えられる。制御プロセス、クライアント・プロセスは登録された名前を用いてデータの参照、書き込みを指定する。

データ管理プロセスが有する機能を以下に示す。

- 制御プロセスの登録
- クライアント　・プロセスの登録
- データの送信
- データの受信
- データの転送
- データの要求
- 管理データの操作
- クライアント　・プロセスの削除
- システムの終了

ここで“管理データの操作”とは、データ管理プロセス内に存在する個別のデータ領域の確保、拡張、縮小、開放、およびデータのパーミッションの設定、変更を意味している。

データ管理プロセス内では、複数の処理が同時に行なわれることはない。そのために１つの処理が行なわれている間は、他の処理要求は待たされることになる。このため、システムの通信の管理が容易となり、また、データに対する同時アクセスによる破壊を避けることが可能となっている。

制御プロセスとクライアント・プロセス

制御プロセスが問題固有の処理の流れを制御している。従って、１つのアプリケーションにおける制御プロセスは１つであるべきである。データ管理プロセスとクライアント・プロセスは制御プロセスの指示に従って実行が行なわれなければならない。しかし、クライアント・プロセスはデータ管理プロセスのように完全な受動プロセスとはなっていない。その理由としては、将来のシステムの拡張のために自由度を確保したためである。そのため、制御プロセスによる制御との整合性のないクライアント・プロセスの記述も可能となってしまうため、注意が必要である。

制御プロセスはクライアント・プロセスとしての性格を合わせ持つことができる。言い方を変えれば、制御プロセスはデータ管理プロセスの制御の機能を有するクライアント・プロセスと考えることが可能である。従って、制御プロセスが処理の一部を負担するようなアプリケーションであっても構わない。

制御プロセスとクライアント・プロセスから可能となる処理要求を以下に示す。

- データ管理プロセスへの登録
- データ送信
- データ受信
- データ要求
- メンバ参照

制御プロセスのみにおいて可能となる処理要求を以下に示す。

- データ管理プロセスにおける管理データの操作
- 終了

2.3 実行環境

本システムで実行されるプロセスはすべてユーザ・プロセスとして実行されるため、プロセスが実行

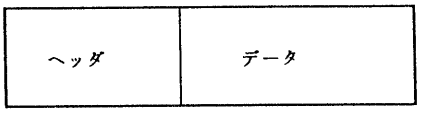

図 2: データのカプセル化

されるホストすべてにおけるユーザの実行権が必要である。また、現在のところ実行プログラムを実行されるホストに自動的に配置するメカニズムは備っていないため、すべての実行プログラムが利用するホストに予め配置されていることを前提としている。ＮＦＳによるホーム・ディレクトリの共有がなされている場合が、もっとも望ましい環境と考えられる。

制御プロセスからの起動（現在未実装）を可能とするためには、rsh（リモート・シェル）による実行が可能であることが望ましい。rsh が不可能である場合には、遠隔実行のためのユーザ・デーモン・プロセスがシステムに必要となるが、現在は実装されていない。

2.4 通信の仕様

通信はこれまで述べたようにストリーム・ソケットを用いて実現され、通信路はプロセスが終了するまで保持されている。通信プリミティブとしては一般に RPC(Remote Procedure Call) が優れていると言われているが、ここではシステムを簡素化するために、メッセージ・パッシングあるいはトランズアクション的な手法により実現されている。

ストリーム型の通信路において、メッセージ・パッシング的なデータの送受信を実現するため、すべてのデータ、あるいは要求メッセージは［図２］のようにデータ・ヘッダをつけ、カプセル化されている。

2.5 実行の流れ

実行の流れを示すため、初期値を持ったデータを２分割し、２つのクライアント・プロセスで処理し、その結果を表示するアプリケーションを例とする。処理の流れは［図３］のようになる。

［図３］より、データ管理プロセスが受動プロセスであることが理解される。データ管理プロセス

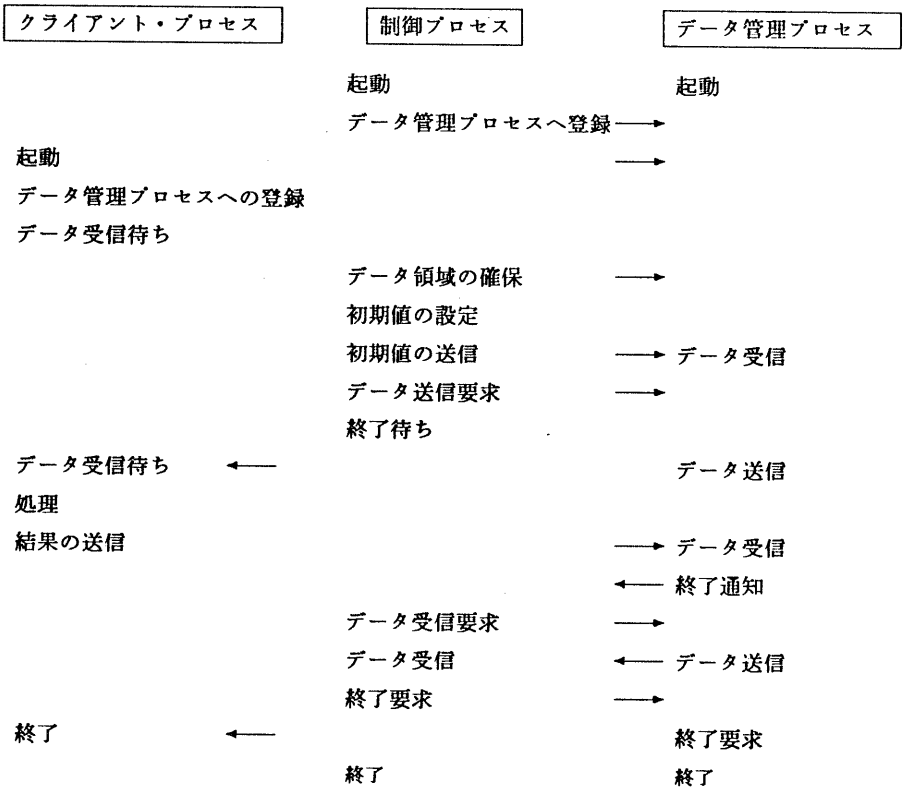

図 3: システムの実行の流れ

は汎用目的で作成されており、問題固有であるべきデータ領域の大きさは予め予見することができないので、起動時にはデータ領域を保持していない。データ領域の確保、開放もまたすべて制御プロセスのからの要求に従って行なわれる。

制御プロセスはアプリケーションの実行に対応して、正しく作成されなくてはならない。クライアント・プロセスから送られたデータを読み込まないような記述を行なった場合においても、それを検知する手段は存在しない。

クライアント・プロセスのプログラミングは、シーケンシャル・プロセスの場合のプログラミングに対して、登録、データの送受信、その他若干の修正及びエラー処理をほどこすことによって可能であり、大幅な変更は必要としない。

［図3］におけるデータ送受信、登録などのクライアント・プロセスおよびデータ管理プロセスで使用される分散処理実現のためのプリミティブは、すべてユーザ・ライブラリとして用意されており、ユーザ・プログラムでは関数呼び出しとして実現される。

3 応用例

3.1 シミュレーションの流れ

本システムの応用例として、モンテカルロ法による衝撃波のシミュレーションを取り上げた。これまでに述べたように、本システムは並列性の認識を行なわない。また、通信のオーバーヘッドに対して処理（数値計算）の割合が高いアプリケーションに向いているため、モンテカルロ法による数値計算のように、マクロな並列性が陽に認識されるアプリケーションの実現が効果的で、かつ、実現が容易である。

モンテカルロ法の応用例において、データ管理プロセスでは

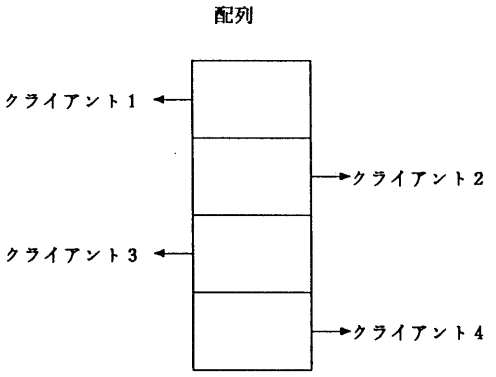

図 4: 配列の分割の様子

1. データの分散
2. データの集積

繰り返し行なわれ、クライアント・プロセスでは、

1. データの受信
2. データに対する演算処理
   ランダム変数を用いた分子の衝突のシミュレーション
3. データの送信

が繰り返される。また、制御プロセスではデータ管理プロセスでは、

1. 初期値データの送信
2. データの分散指示
3. すべてのデータの集積待ち
4. データ要求
5. データの入れ替え
   境界での分子の入れ替え
6. データ送信
7. 最終データ処理　(出力)

が行なわれ、2から6が繰り返される。

ここでのデータは大規模配列で、分子運動を示しており、この例では最大 90000 × 3 の double の 2 次元配列となっている。配列の分割の様子を［図 4］に示す。

［図 4］では配列を 4 つに分割しているが、分割はソース・コードでなく、制御プロセス実行時の引数によって変更可能である。分割数を増やした場合は、当然、それに応じて実行されるクライアント・プロセス数も増加させなくてはならない。

モンテカルロ法による分子運動のシミュレーションでは、分割数をあまり大きくとることはできない。なぜなら、分子に対してその運動領域が充分大きくないと、シミュレーションが成り立たなくなるからである。すなわち分割されたシミュレーションでは、

- 分割部分に仮想的な仕切り板を挿入する
- 分割部分における部分シミュレーションを行なう
- 分割データを集積する
- 仕切り板を取り除く
- 分子の入れ替えを行なう

というシーケンスで、実行が行なわれる。そのため、分割数を増加させることは仮想的仕切り板の挿入箇所を増やすことになり、本来のシミュレーション対象の環境と大きく異なってしまうことになる。

3.2 実行結果

CPU 能力が同等のホスト 4 台に、モンテカルロ法によるシミュレーションを分割した所、単一ホストでの実行時間が 30 分程度のものが 8 分程度と、ほぼ分割数に比例した処理速度の向上が見られた。しかし分割割合は一定であるため、当然であるが、その内の 1 台のホストの負荷が高い場合や CPU の処理能力が劣る場合、そのホストがボトルネックとなり、最終処理時間が著しく悪化した。

3.3 自動分割

3.2節で示されたような問題を解決するには、ホストの負荷、あるいは CPU 能力を判断する手段が必要となる。そのため、システムにデータ送信からデータ受信までに要する時間を計測する機能を与えることによって、クライアント・プロセスの処理能力を推定する方法を手段を持たせるように変更を行なった。さらに、測定された時間を元に分割割合を変更し再分割を行なう、自動データ送信機能を付加した。

自動データ送信機能を確認するため、800 × 10 × 10 の整数のデータ (320Kb) を 4 つのクライアント・

クライアント	マシンタイプ	1回目		2回目	
		データ量 [byte]	時間 [sec]	データ量 [Byte]	時間 [sec]
クライアント1	SUN3	80000	61	32800	25
クライアント2	SUN3	80000	61	32800	25
クライアント3	SUN4	80000	18	111200	25
クライアント4	SUN4	80000	14	143200	26

表 1: 自動分割における分割の様子と処理時間

プロセスに分割し処理を行なうテスト・プログラムを作成した。実験に使用したホストは、CPU能力の差が明らかに現れるよう、SUN3およびSUN4の2種類ホストを各々2台用いた。テスト・プログラムでの分割の様子と処理時間の結果を［表1］に示す。

［表1］に示されるとおり、ほぼ一定した処理時間が得られた。分割は処理に不都合が生じないよう、最上位次元のレベルで行なわれている。しかし自動分割機能は実験段階であり、分割方法やエラー処理などに幾つかの重要な問題を残しているため、正式には実装されていない。今後の最重要課題として捉えている。

4 今後の課題

本システムはプロトタイプであり、解決すべき課題も数多く存在しており、現在継続して開発を続けている。今後の課題として改良すべき点を以下に列挙する。

- **パイプライン通信**
  現在の通信の仕様はすべてトランズアクション的であるため、データが継続して発生するような場合に、データを細かく分割して通信を行なわなくてはならない。そのため、そのような場合にはプログラミングも複雑となり、効率も低下してしまう。そこで、ストリーム・ソケットの特質を活かした通進路のサポートを行なう。
- **データ管理プロセスを経由しないデータ通信**
  本システムではすべてのデータ通信はデータ管理プロセスを経由して行なわれる。しかし、前項のパイプライン処理などのように、クライアント・プロセス同士あるいはクライアント・プロセスと制御プロセスのデータ通信も必須となる場合があり、直接のデータ通信のサポートも必要となる。
- **間欠的な接続**
  すべてのクライアント・プロセスは終了までその通進路を保持するため、データ管理プロセスに接続されるクライアント・プロセスの数が利用可能なディスクリプタに制限される。データ通信が間欠的である場合に、接続を保持しないクライアント・プロセスが可能となれば、データ管理プロセスがサポートできるクライアント・プロセスの数の制限を排除できる。しかし、接続を保持しないクライアント・プロセスに対しては、データ管理プロセスでは終了したのかあるいは処理中であるのか等、その状態を把握することができない、という問題が生じる。
- **プロセスの自動起動と動的起動**
  現在必要プロセスの自動起動のメカニズムは実装されていない。すべてのプロセスを制御プロセスからの自動起動として、アプリケーション実行を簡素化する。また、自動起動が可能となれば、必要に応じた制御プロセスによるクライアント・プロセスの動的起動も可能となる。
- **ユーザ・デーモン・プロセス**
  rshなどが実行不可能な場合に、制御プロセスによるクライアント・プロセスの動的起動をサポートするためには、実行のためのユーザ・デーモン・プロセスが必要となる。
- **オーサライゼーション**
  通進路の確立においてオーサライゼーションのメカニズムは実装されていないため、悪意のユーザの妨害には対処できない。この問題は今後の課題の一つであるとは認識しているが、必須であるとは考えていない。

- **エラー処理**
  通信に精通したユーザが利用するわけではないので、エラー処理を徹底する必要がある。
- **データ管理プロセスへの演算の依頼**
  データ管理プロセスは通信と媒介とデータの管理のみを行ない、演算処理機能を持たないため、単純な演算であっても、必要データをクライアント・プロセスあるいは制御プロセスに移し、演算を行ない、データ管理プロセスに返すという手続きを取らねばならない。簡単な配列の演算などはデータ管理プロセスに依頼し、データ管理プロセス内で行なわれることが望ましい。
- **デッドロック回避**
  誤った処理の流れを記述するとデッドロックとなってしまうため、タイムアウトを利用したデッドロック回避のメカニズムが必要となる。しかし数値計算においては、実行時間はマシンの負荷、メモリ利用量などによって大きく左右されてしまい、タイムアウトの設定は困難である。ある程度の安全メカニズムは実装可能であるが、本質的な解決はできない。
- **配列でないデータの利用**
  すべてのデータ通信は配列として行なわれるため、単純変数であっても1次元1要素の配列として取り扱わなければならならず、ユーザ・インターフェースおよび、データ通信のオーバーヘッドとなっている。単純変数のデータ通信もまた必要であろう。
- **データ表現**
  本システムでは、データのバイナリ表現が同じであると仮定しており、データ通信においてデータ変換を行なっていない。汎用性のためには、SUN RPCのXDR(eXernal Data Representation)のようなメカニズムが必要となる。
- **ユーザ・インターフェース**
  すべてのシステムはC言語により開発されており、ユーザ・インターフェース・ライブラリもまたCの関数呼び出しである。しかし多くの数値計算プログラマは依然としてFORTRANを利用しており、FORTRANライブラリも必要となろう。
  また、現状では細かな制御を可能とするため、ライブラリ呼び出しにおける引数の数が多くなっており、引数の指定が繁雑である。しかし、この問題は制御の自由度とのトレードオフであり、根本的には解決できない。将来的には簡易版のライブラリの作成も必要であるかもしれない。

5 まとめ

本研究では数値計算プログラマがネットワークを介して、複数の計算機を利用した数値計算の分散処理を支援するシステムを構築した。システムはモンテカルロ法によるシミュレーションに応用され、その有効性が示された。

しかし現在のプロトタイプには数多くの課題が残されており、より汎用的な、また、より良いユーザ・インターフェースを有したシステムとするため、さらに改良が必要である。

UNIXネットワーク環境が一般的となった現在、ネットワーク資源ならびに計算機資源を一般ユーザが有効に利用するには、本システムのような機構がなくてはならないであろう。

なお、本システム完成後は希望サイトへの配付を考えている。

参考文献

1. **Gregoly R. Andrews and Fred B. Schneider,** "Concepts and Notations for Concurrent Programming," *Computing Surveys,* Vol. 15, No. 1, March 1983, pp.3-43
2. **Lyon B.,** "Sun Remote Procedure Call Protocol Specification," *Sun Microsystems Technical Report,* 1984
3. **Lyon B.,** "Sun External Data Representaion Protocol Specification," *Sun Microsystems Technical Report,* 1984
4. 矢吹道郎, 他、"数値計算環境におけるプロセス間通信のインターフェース"、情報処理学会第35回全国大会、1987