RPCによる並列計算

平山　弘　　佐藤　創太郎

システムデザイン工学科

Parallel Computing by RPC

Hiroshi HIRAYAMA and Soutarou SATOU

Abstract

In this paper, we confirm that the large problem can be performed very fast by RPC (Remote Procedure Call) on some distributed WindowsNT workstations. And we discuss possibility of high performance, usability and low cost computing system, consist of PC workstations. Today, PC network system, such as Local Area Network, exists as an usual environment. We think that is instructive that discussion of a way to make good use of that system, and moreover, in the near future's retired workstations those are working as main machine now.

Key Words:　Paraleru computing, RPC, remote procedure call, personal computer

1. はじめに

インターネットが日常的に用いられている今日、ネットワークを用いたコンピュータの利用は以前よりもずっと身近なものとなってきている。ネットワークシステムには、たくさんのコンピュータが接続され、それぞれの単独のコンピュータとして利用されている。これらのコンピュータを単独で利用するだけでなく、ネットワークで接続された多くのコンピュータをひとつのコンピュータとして活用する方法が研究されている。

すでに存在するコンピュータネットワークをどのように活用するかについてさまざまな研究が行われている。

本研究では、現在非常に良く使われているWindowsNTマシンをRPC（リモートプロシージャコール）及びマルチスレッド[1]を用いて、多数のコンピュータを一括利用することによって、大規模な計算処理を効率よく分散し、かつ使い勝手のよい利用法を調べた。

一台のクライアントに複数の計算サーバを置き、計算処理を分割する。このような計算処理方法で、規模の大きい計算の計算時間の短縮化を図った。

このような研究は、これまで高価なUNIXをオペレーティングシステムとするワークステーションを使って行われてきたが、安価なパーソナルコンピュータを使ったものはあまりない。パーソナルコンピュータを利用したとしても、オペレーティングシステムがUNIXの場合が多い。

2．RPC（Remote Procedure Call）

RPC（Remote Procedure Call）は、ローカルマシン上で実行される関数だけでなく、ネットワークを介した他のマシン上で実行される関数（サブルーチン）を呼び出し、利用できる機能である。

RPCを利用したアプリケーションは大規模

な計算を行う場合に、他の高性能なコンピュータに計算部分を移譲して、その実行結果を受け取ったり、ネットワークを介して、一つのサーバが集中管理しているデータベースへのアクセスをスムーズにしたりすることを目的に開発されたものである。アプリケーションによる負荷を分散させたり、逆に一個所に処理を集中させたりする機能である。

RPCの機能は、クライアントとサーバ間のデータの受け渡しをNDR（Network Data Representation）という共通のデータ形式で行う。浮動小数点の形式が全く異なるコンピュータ間でも透過的に利用することができるようになっている。今回用いたWindowsNT上に実装されたMicrosoftリモートプロシージャコール(Microsoft RPC)は、OSF(Open System Foundation) の DCE(Distributed Computing Environment）によるRPC機能と互換である。

計算の負荷を分散させる目的でRPCを用いる利点は、複数ある計算サーバの演算能力を同じには揃える必要がないという点にある。このため、すでに存在するPCワークステーションを用いて、計算力の高い一つの計算システムを作り上げるために、何の変更も必要がないという点である。

3．マルチスレッド

オペレーティングシステムが実行中のプログラムを管理する単位の一つにプロセスがある。プロセスとはメモリにロードすることが出来るプログラムの単位であり、オペレーティングシステムは、プロセスごとにプログラムに与えるためのヒープメモリ領域等の資源をを割り当てる。

WindowsNTでは、最近のオペレーティングシステムが、すべてそうであるように、プログラムを実行するためのCPU資源の割り当てをスレッドという単位で管理する。スレッドはプロセスに与えられたメモリ空間を用いてプログラムの命令を実行する。一つのプロセスは複数のスレッドを持つことが出来、単一プロセス内の複数のスレッドはそれぞれに与えられたスタック領域を除いて、ヒープ領域や、大域メモリ空間を共有する。

プロセスは最低でも一つのスレッドを持つ。プロセスが作成されるとき、そのプロセス内で動作しているスレッドをプライマリスレッドと呼ぶ。プログラムコード内でプログラムエントリポイントとなる部分と、それから呼び出される通常のサブルーチンの実行されるシーケンスがそれである。

WindowsNTで、一つのプロセス中に複数のスレッドを生成するためには、プライマリスレッド内のどこかでCreateThreadというシステムコール(API)を用いて行う。新しいスレッドのエントリーポイントを指定しAPIを呼び出すと、新しいスレッドが開始され、APIの呼び出し側では制御が即座に自分に帰ってくる。プライマリスレッドから見れば、そのAPIを呼び出した時点から、自己のメモリ領域を共有する他のプログラムが起動したように見える（たとえばC言語ならば、複数の関数が同時に実行されているような感じに見える）。

一つのプロセス内に複数のスレッドが作られる場面とは通常、時間のかかる処理とユーザーその他から発生するイベントの処理を分離し、プログラムの応答性を効率よく確保したい場合などである。

本研究では、時間のかかる処理とは、RPC呼び出し実行される部分である。複数のスレッドを生成する目的は、イベントの発生に対する応答性の確保ではなく、RPC呼び出し部分を独立したスレッドに行わせることで、一つのプロセス内で多数のRPC呼び出し、処理を同時に実行出来るようにするためである。

4．サービス

クライアントーサーバ型のソフトウェアシステムにおいて、サーバプログラムは通常、起動された直後から待機状態に入り、クライアントからの要求が発生するまでシステムのバックグラウンドに常駐する。

一度起動すると、要求があるまでシステムのバックグラウンドで動作しつづけるようなプログラムを、UNIXではdaemonという。MS-DOSではこのようなプログラムのことをTSR（Terminate and Stay Resident）と呼んだ。マルチタスク型の他のオペレーティングシステムと異なり、一定間隔であるイベントの発生を監視する必要のあるMS-DOSのTSRは自立的にCPUの実行時間を奪い、自分の仕事を終えたらフォアグラウンドのアプリケーションに制御を戻すことが必要だった。

WindowsNTで、このようなサーバプログラム、特にシステムの起動と同時に起動している必要のあるプログラムをサービスと呼ぶ。サービスは通常WindowsNTシステムの起動と

同時に実行が開始され、そしてシステムに特に誰かがログインしていない間も、誰かがログインしている間もバックグラウンドで動作しつづける。

具体的には、例えばRPCサーバソフトウェア等をサービスとして登録しておけばPCワークステーションの電源を投入するだけでクライアントの要求待ち状態に入る事が出来るため使い勝手が良くなる。

5. RPC処理の概要

5.1 RPC処理の流れ

RPC処理の流れは、つぎのようになる。

1）まず最初にクライアントアプリケーションがローカルシステム上にあるクライアントスタブを呼び出す。

2）クライアントスタブはクライアントアプリケーションから渡されるデータをサーバに送信するために NDR 形式に整える（マーシャルする）。

3）さらに、クライアントスタブは、クライアント RPC ランタイムライブラリ内の関数を呼び出して、RPC 要求とそのデータの送り先であるサーバシステムをネットワーク上で探し出してそこに送る。

4）クライアントから送られてきた RPC 要求とデータは、サーバ RPC ランタイムライブラリ内の関数が受け取る。

5）サーバが受け取ったデータは、サーバスタブに渡されてもとの形に戻される（アンマーシャルされる）。

6）サーバスタブは復元されたデータを用いてサーバアプリケーション内にある実際のプロシージャを呼ぶ。

7）リモートプロシージャを実行して選られたデータは、再びサーバスタブに渡される。

8）クライアントスタブがデータ送信時に送ったのと同じように、サーバスタブもクライアントシステムへ送信のために、得られたデータを NDR 形式に変換する。

9）サーバスタブがサーバ RPC ランタイムライブラリ内の関数を呼び出し、クライアントにデータを送信する。

10）サーバから送り返されたデータは、クライアント RPC ランタイムアプリケーション内の関数が受け取る。

11）データは、クライアントスタブに渡されて復元される。

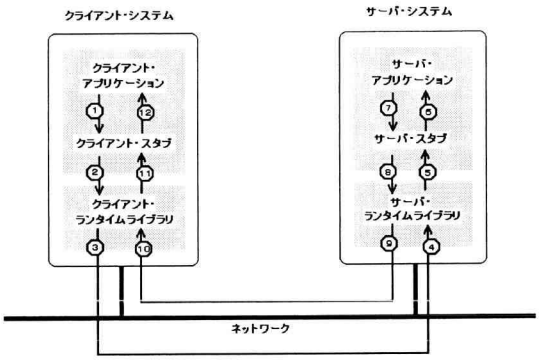

図1　RPCの処理の流れ

12）クライアントスタブは、復元されたデータをクライアントアプリケーションのメモリに書き込み、RPC 処理を終了する。

図1は、クライアントアプリケーションからRPC要求が出された場合の処理の流れを示している。

5.2 RPCクライアントの処理

クライアントからRPC機能を使用するには、リモートプロシージャの提供しているサーバを探し出し、そのバインディングハンドルを取得する処理が必要である。

今回、本研究では自動バインディングを使用するのでこの処理はクライアントスタブがRPCネームサービスを使用して自動的に行う。そのためにクライアントアプリケーションには、IDLファイルから生成されるヘッダファイルをインクルードしておく他には、特別な処理を追加する必要はない。しかし、手動バインディングを使用する場合には、クライアントアプリケーション自身でこの処理を行わなければならない。

クライアントアプリケーションから適切

なバインディングハンドルを取得する方法には、以下の3通りがある。

1．RPC ネームサービスを使用し、サーバ候補のバインディングハンドルを１つずつ取得しながら探す。

2．RPC ネームサービスを使用し、全サーバ候補のバインディングハンドルを取得してから選択する。

3．文字列バインディングを指定し、これをバインディングハンドルに変換する。

RPC機能を使用する際、まずサーバークライアント間で、プロシージャの利用に関する取り決めを決めておかなければならない（引数のデータタイプ、個数など）。この取り決めをここではインタフェースと呼ぶ。インタフェースの定義には、C言語に似た構文を持つInterface Definition Language（IDL）という言語を用いる。Microsoft RPC では、IDLはMicrosoft IDL（MIDL）として定義されている。

IDLファイルは、インタフェースを識別するためのインタフェースヘッダと、提供されているプロシージャのインタフェースを提供するインタフェースボディの２つの部分で構成する。

5.3 RPCサーバの処理

サーバは、サーバ自体のもつリモートプロシージャをデータベースに登録し、クライアントからのリモートプロシージャコール要求を待つという一連の初期化処理を行う必要がある。

1）サーババインディングを作成する。

2）エンドポイントマップデータベースにエンドポイントを登録する。

3）RPC ランタイムライブラリにインタフェースを登録する。

4）ここで、クライアントからのプロシージャコールを待機する状態に入る。

5）サーバアプリケーションを停止する時は、リモートプロシージャコールの待機状態の終了を宣言し、データベースに登録したインターフェースを削除する。

6. サービス処理

6.1 サービスの振り付け

WindowsNTでは、すべてのサービスはサービスコントロールマネージャ(SCM)と呼ばれるシステムにより管理される。SCMは既知のサービスのリストをレジストリに保存し、システムの起動時に自動的に、またはユーザーからの要求があったときに、これらのサービスを起動させる。SCMはサービスのリストとサービスのスタートアップステータスをレジストリに保存しておく。新しいサービスをインストールしたときに、このリストに追加する。後でサービスを削除する事も可能である。

サービスとして振舞うプログラムは通常の実行ファイルであるが、サービスコントロールマネージャとインタフェースを適合させるためプログラムは特別な要求条件を満たさなければならない。

6.2 サービスのインストール

作成したサービスプログラムを使用するには、これをWindowsNTのサービスとして登録しなければならない。インストールする事により、SCMにこのサービスの存在を知らせ、WindowsNTコントロールパネルの、サービスアプレットに表示されるサービスのリストに追加される。サービスのインストール、削除には、専用のインストールプログラムを作成する必要がある。

7. 並列計算実行例

本研究で行った並列計算例は、単純な計算と行列の乗算の二つである。最初の計算例は、データの転送がほとんどない必要がないものである。行列の乗算は、かなりデータ転送を必要する例として扱った。

7.1 単純な並列計算例

単純な並列計算例として、以下の計算を行った。プログラムは、クライアントとサー

バのそれぞれあり、クライアント側ではサーバのプロシージャコール一つにつき一つのスレッドを生成する。

実際の計算処理はRPCサーバ側のプログラムで行い、クライアント側は主に計算処理の制御を行う。

計算に使用したマシンはPentium Ⅱ 400MHz Dual processorマシン（以下PⅡ400Dualとおく）。PentiumⅡ450MHz（以下PⅡ450とおく）である。

計算サーバの台数を増やす事によって計算時間をどの程度短縮化できるかを確認するために以下の計算を行った。

nという与えられた数値を1からの逆数をとってnの逆数まで加算していく計算。つまり、

$$\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n-1}+\frac{1}{n} \qquad (7.1)$$

この計算をサーバの台数分に分割する。

計算は、n=100からn=2,000,000,000までを行った。

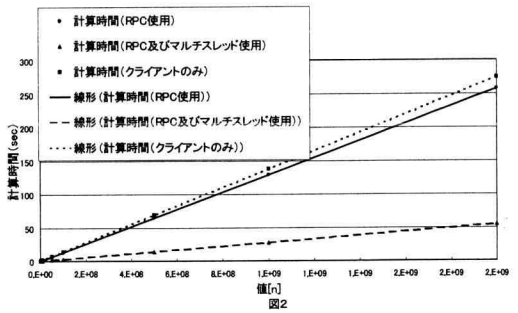

図2

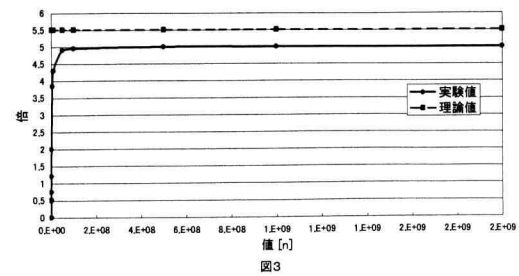

図3

1）クライアントのコンピュータにPⅡ400Dualを1台、サーバにはPⅡ400を4台使用して計算を行った。その結果を図2に示す。ならびに、この時のクライアントのみの計算時間に対してRPC及びマルチスレッドを用いた計算時間の比較を図3に示す。

2）クライアントにPⅡ400Dualを1台、サーバにPⅡ450を2台使用して計算を行った。その結果を図4に示す。ならびに、この時のクライアントのみの計算時間に対するRPC及びマルチスレッドを用いた計算時間の比較を図5に示す。

3）クライアントにPⅡ450を1台、サーバにPⅡ450を3台とPⅡ400Dualを1台使用して計算を行った。その結果を図6に示す。ならびに、この時のクライアントのみの計算時間に対するRPC及びマルチスレッドを用いた計算時間の比較を図7に示す。

4）1）の計算処理では、マシンPⅡ400DualはCPUを2台使い切っていない。2台のCPUを使うためにこのPCに割り当てられる計算をマルチスレッドにして処理をする。

ここではこの条件で、クライアントにPⅡ400Dualを1台、サーバにPⅡ450を4台使用して計算を行った。その結果を図8に示す。ならびに、この時のクライアントのみの計算時間に対するRPC及びマルチスレッドを用いた計算時間の比較を図9に示す。

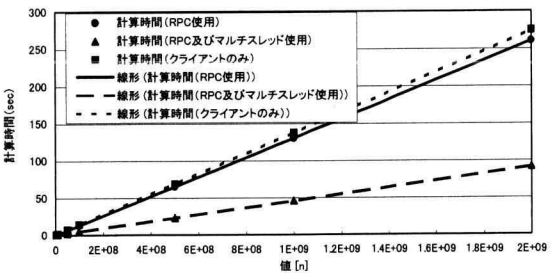

図4

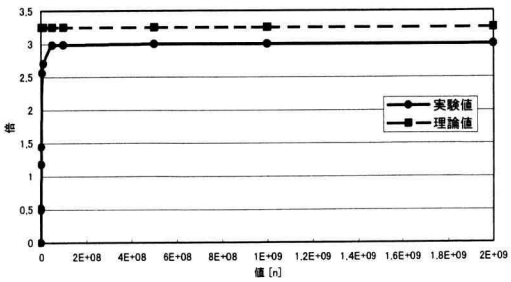

図5

これらの計算時間の結果を、次の図10

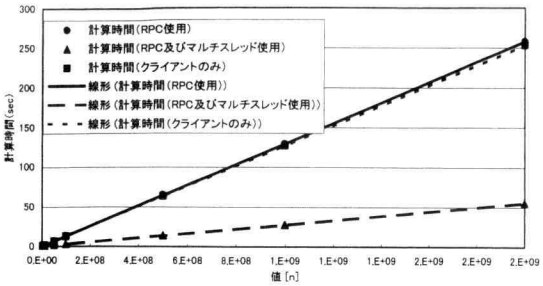

図6

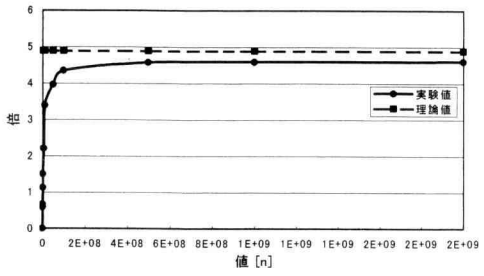

図7

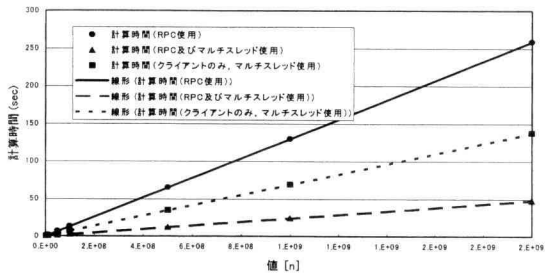

図8

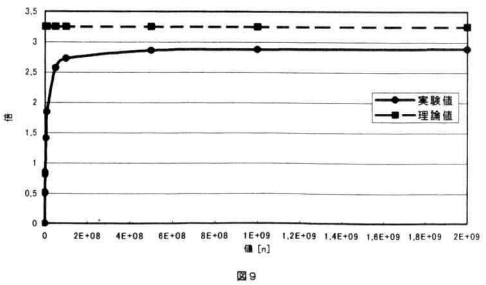

図9

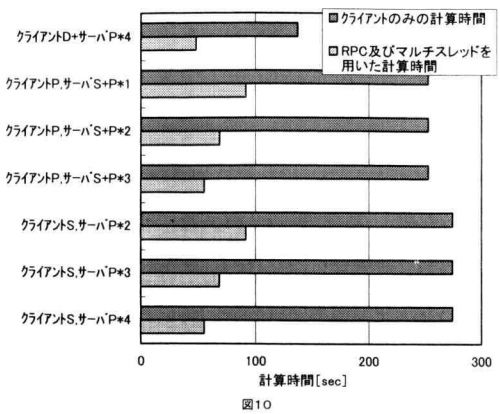

図10

に表す。

図３，５，９，７では、クライアントのみで行った計算時間と、RPC及びマルチスレッドを使用して行った計算時間を比較するために使用したPCの総クロック数をクライアントのクロックで割り、それを理論値とした。

クライアントにPⅡ400 Dualを１台、サーバにPⅡ450を４台、３台、２台と計算を行った。この場合での、それぞれの理論値と実験値と比較する。

計算式中でnが100,000 以下の場合、実験値が理論値に及ばず低い結果を出す。実際の計算に費やされる時間よりも、データ転送や、RPCの手続きにかかるオーバーヘッドの割合が大きすぎるためRPC及びマルチスレッドを用いた方が、かえって時間がかかってしまう。

計算量が十分に多い場合、つまりnが50,000,000 以上もあれば、実験値は理論値のおよそ９０％の値になる。

クライアントにPⅡ４５０を1台、サーバにPⅡ４００Dualを1台とPⅡ４５０を3台、2台、1台とで計算を行った場合と、クライアントにPⅡ４００Dualを1台使用し、そのPⅡ４００Dualにマルチスレッドとして計算を行わせた場合にも上と同じことがいえる。クライアントにＰⅡ４５０を使用した場合に、実験値は理論値のおよそ９４％、ＰⅡ４００Dualの場合はおよそ９０％の値になった。

また、サーバの台数を増やしていくことで、計算時間も台数を増やした分に見合うだけ計算時間も短縮されていく。

今回作成したプログラムでは、計算は等分に区切られてそれぞれの計算サーバに割り当てられる。問題となるのは、本研究での計算方法では、かかる計算時間が並んだ計算サーバのうちもっとも遅いマシンで足並みがそろえられてしまうという点である。本研究では行ってはいないが、利用する計算サーバを自動で検索し、それを利用するというような方法を将来取ろうとするならば、あらかじめ利用する計算サーバの計算力を評価できなければ、単に計算サーバの台数を増やすだけでは計算がかえって遅くなってしまう。

7.2 行列の乗算

n行n列行列の乗算を、以下の条件で行った。計算の分割はサーバの台数分である。

1) 使用するネットワークが 10BASE-T、クライアント１台、計算サーバを４台とした場合。

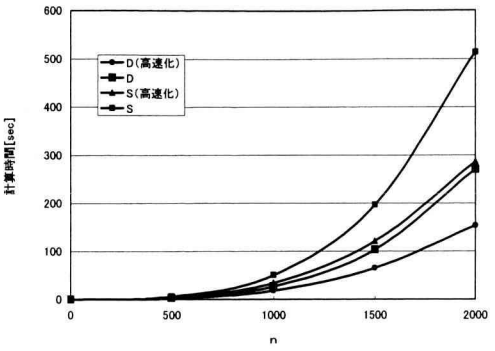

図11

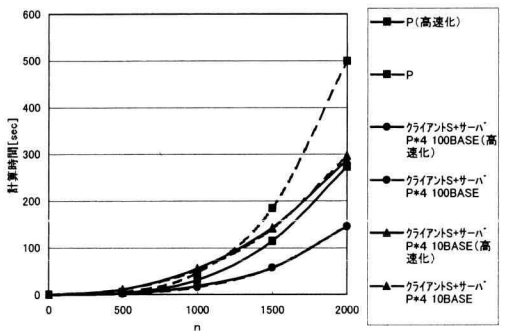

図12

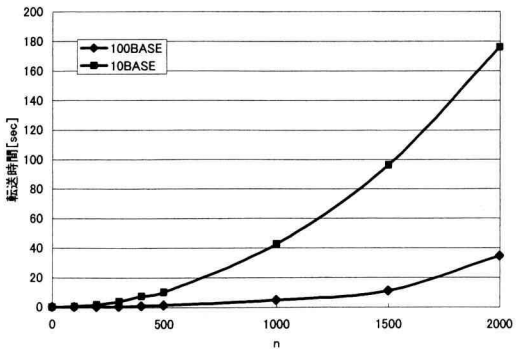

図13

2) 100BASE-T で、クライアント１台、計算サーバを４台とした場合、１、２とも、クライアントはデータの転送、結果の受信のみを行う。

3) コンピュータ１台で計算を行った場合以上の条件で、コンピュータの機種を替えてこの計算を行いこの時の計算時間を比較した。

計算は500行500列から2000行2000列までを行う。また、計算は配列をそのまま読んで行ったものと、配列を計算前に一度、乗算に最適なように並べ直してから（つまり、乗算演算子の右側にある行列を転置させ、計算中で依存するデータが配列内で連続に並ぶようにする）行ったものがある。

計算結果は以下のようであった。

1）コンピュータ１台での計算時間の結果を図１１に示す。（注）D=PⅡ400Dual、S=PⅡ400、P=PⅡ450 とおく

2）RPC 及びマルチスレッドを用いたマトリックスの乗算の計算時間を図１２に示す。

3）このマトリックス計算では、転送するデータの量が多いため転送時間も計った。クライアント１台、サーバ４台で、ネットワークに１００ BASE を用いた場合の転送時間と１０ BASE を用いた場合の転送時間を図１３に示す。

マトリックス乗算で、配列をそのまま読んで計算を行った場合と、配列を1度最適に並び替えた場合とで比較してみると、1台のコンピュータで行った場合およそ1.4倍から1.8倍に高速化されたことがわかる。

コンピュータ1台で、このように計算を行った場合には処理を最適にすることだけで、これだけ計算を高速化することができる。

クライアントを1台、サーバを4台でRPC及びマルチスレッドを用いて計算（計算はサーバで、クライアントは計算をしないものと、サーバで計算を、クライアントで配列の転置を行ったもの）を行った場合およそ0.6倍から1.0倍になった。

この場合は、配列を置き換えてメモリアクセスを短縮した時間と、クライアントでの配列の転置作業の時間とが変わらないために、計算時間の短縮を計ることはできなかった。しかしサーバにかかる負荷としては、クライアントで転置行列をおこなった方がはるかに少ない。新たに配列を格納するメモリを使用しないですむ分こちらの方が有効だと思われる。

1台（PⅡ450）で計算を行った場合と、クライアント1台（PⅡ400 Dual）、サーバ4台（PⅡ450）の5台でRPC及びマルチスレッドを用いて行った場合では、1台で計算を行った場合に対しておよそ２倍から３．４倍に高

速化された。１台のクライアントに対して計算サーバ4台を用いるので、およそ４倍に高速化される事が期待されるが、行列の乗算などはデータ転送等のオーバーヘッドがやはり問題となるため実際にはそうならない。

クライアント1台、サーバ4台でのデータ転送時間を100BASE、10BASEとで比較すると、100BASEの場合、10BASEの場合と比べ約８～９倍になる。実行結果をみると、100BASEの場合には配列の数が増えるほど理想に近くなっていることがわかる。これは計算量が少ない場合よりも、多い場合の方がデータ転送等のオーバーヘッドの割合が小さくなるためである（つまり、行列の大きさnに対して、データの転送量がn^2なのに対して、乗算はn^3かかる）。

ネットワークに10BASEを使用する場合、各サーバ毎の計算力に対してデータ転送にかかるオーバヘッドが大きすぎるためあまり実用的ではない。

8 おわりに

本研究の結果から、RPCおよびRPC呼び出し毎にスレッドを一つずつ割り当てるという方法によって、複数台使用するコンピュータの能力分だけに見合ったプログラムの高速化が可能であり、それがWindowsNT上で実装出来る事がわかる。

本研究では主に、計算時間の短縮の可能性について実験を行った。WindowsNT上でRPCを用いて大規模な計算処理を短縮できる事は確認できた。

今後問題となるのはその利用可能性についてである。今回の研究では結局、実験に用いるプログラムは作成する毎に、人手でサーバマシン上にサービスとして起動させてやる必要がある。実験として作成したようなプログラムに多少の変更を加えるたびに、繰り返し計算サーバ全てにプログラムを配送して、コンパイルし直し、それをサーバとして立ち上げて回るというのは手間のかかる作業であろう。

ここでのサーバプログラムの配送、コンパイル、サーバ起動の過程を自動化できれば、本研究でのこの計算システムはもっと利用価値の高いものになるだろう。

参考文献

[1] Marshall Brain, 郡司芳昭 訳, 三田典玄 監修: 「Win32システムサービスプログラミング」、プリンティスホール、1996

[2] 及川卓也：WindowsNT4.0完全技術解説、日経BP社、1997

[3] Steve Kleiman, Devang Sha, Bart Smaalders: 実践マルチスレッドプログラミング、アスキー出版局、1998