修 士 論 文

ワークステーションクラスタによる
高速並列処理に関する研究

指導教官　堀口進 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

奥野弘之

1998 年 2 月 13 日

目 次

図 目 次

表 目 次

第 1 章

はじめに

1.1 研究の背景と目的

近年，計算機 1 台あたりの価格性能比の向上によりワークステーションをネットワークで接続し PVM(Parallel Virtual Machine)[2] や，MPI(Massage Passing Interface)[10] などのメッセージ通信ライブラリを利用してワークステーションクラスタを構築し，仮想並列計算機として並列処理を行わせる研究が盛んになってきた [14]．これによって専用の並列計算機を用意しなくても身近なワークステーションを用いて容易に並列処理の環境を入手することが出来るようになった．しかし，この様なワークステーションクラスタでは，ワークステーション間のデータをやりとりを行うメッセージ通信が処理時間の多くを占め，並列処理のボトルネックとなるため，投入台数に見合う処理性能の向上が得られていないという問題がある．

この様な問題を解決するため，ワークステーションクラスタにおける並列処理の性能向上の手法として，ハードウェア，ソフトウェア両方面から様々な取り組みが行われている．

ハードゥエア面での取り組みには，通信を行うネットワークの高速化がある．ワークステーションを接続する従来の 10Base-T ハブのスイッチング化による通信帯域の向上や，FDDI，ATM，100Base-T ハブ、近年開発されたギガビット LAN の Myrinet[15] による通信速度そのものの向上などがある．

ソフトウェア面からの取り組みとしては，PVM にスレッドを用いるようにした TPVM[9]，lighteight process を用いるようにした LPVM[8]，pardiPVM,ActiveMessage[5]，イリノイ大で開発された FM(FastMessage)[16] などがある．

しかしながら，これらは性能を向上させる反面，導入に大きなコストがかかったり，専用のソフトゥエアやインタフェースを持っており既存のプログラム資産などの利用が困難であるなどの問題点がある．

本研究では標準の PVM や MPI と，10Base-T スイッチハブを用いて構成するワークステーションクラスタにおける効率的な処理方法の提案を行う．提案にあたり PVM と MPI のメッセージ通信性能についての検討を行う．その検討結果より PVM のブロードキャストの通信速度を向上させる階層型ブロードキャストについて提案し，その性能について評価と検討を行う．

10Base-T スイッチハブを用いたワークステーションクラスタに適した処理方法を提案するため，ソーティング，N クィーン問題，TSP(Travelling Salesman Problem) を実行し，その実行性能についての評価を行う．評価の結果より，10Base-T スイッチハブを用いたワークステーションクラスタにおいて有効な並列処理方法の提案を行う．

1.2 本論文の構成

本論文の構成は，以下のとおりである．

第 2 章では，ワークステーションクラスタとメッセージ通信ライブラリについて述べる．また現在取り組まれているワークステーションクラスタやメッセージ通信ライブラリの性能向上に関する研究について述べる．

第 3 章では，スイッチハブをネットワークに用いたワークステーションクラスタを構築し，そのメッセージ通信性能の評価を行なう．評価結果よりメッセージ通信の高速化について，スイッチハブを利用した階層型ブロードキャストの提案を行い，性能評価と検討を行う．

第 4 章では，アプリケーションとしてソーティング，N クィーン，TSP(Travelling Salesman Problem) を実行し並列処理性能についての検討を行い、ワークステーションクラスタに適した処理方法を提案する．またアプリケーションに対するメッセージ通信ライブラリの通信性能の影響について，3 章において評価したメッセージ通信ライブラリの通信性能を元に検討する．

第 5 章でまとめと今後の課題について述べる．

第 2 章

ワークステーションクラスタ

2.1 はじめに

ワークステーションクラスタは，既存のネットワークとワークステーションに PVM や MPI などのメッセージ通信ライブラリを利用しており，容易に並列処理環境を入手することが出来る．しかし，この様に構成されたワークステーションクラスタは処理時間におけるメッセージ通信の割合が大きく，投入台数に見合う処理性能が得られない問題がある．

本章ではワークステーションクラスタの現状について述べるとともに，ワークステーションクラスタにおける処理性能向上についてハードウェア，ソフトウェア両分野で現在取り組まれている方法について述べる．

2.2 ワークステーションクラスタ

ワークステーションクラスタとは，複数のワークステーションをネットワークで接続し，メッセージ通信ライブラリを用いて WS 間でデータをやりとりし，並列処理を行うものである．

ネットワークには既存の TCP/IP のネットワークを用いることが多いため，10Base-T ハブに接続された，ネットワーク速度の低いものであった [14]．近年では 100Base-T ハブの入手が容易になってきたため，ワークステーションクラスタのメッセージ通信速度は改善される傾向にある．

また，最近ではワークステーションよりも更に安価に構成が可能として，ワークステー

ションの代わりに PC を用いる PC クラスタ [1] も登場している．

2.3 メッセージ通信ライブラリ

メッセージ通信ライブラリは，ワークステーション間でのメッセージのやりとりを行い並列処理を実現するものである．

本節では，代表的なメッセージ通信ライブラリである PVM と MPI について述べる．

2.3.1 PVM(Parallel Virtual Machine)

PVM[2] は 1990 年に ORNL(Oak Ridge National Laboratory) で開発された，メッセージ通信ライブラリで，ネットワークに接続されたワークステーションをはじめとする異機種 UNIX マシン群を単一の分散メモリ型の仮想並列計算機—バーチャルマシン—として利用することを可能にする．

PVM は大きく 2 つに分けられて構成される．一つは pvmd3 と呼ばれるデーモンで，バーチャルマシンを構成する各ワークステーションに常駐しており，ワークステーション間の通信処理や，バーチャルマシンを構成するマシンの管理などを行っている．もう一つは，PVM インタフェースルーチンのライブラリで，libpvm3.a と呼ばれる．このライブラリはメッセージ通信，プロセスの生成，バーチャルマシンの再構成のための各種サブルーチンを提供する．

また PVM では，処理を行うプロセス間の通信方法に，pvmd によって通信を行う方法と，TCP を用いて直接通信を行う方法の 2 種類の方法が提供されている．

pvmd によって通信を行う場合は，UDP ソケットを用いて通信が行われる．

TCP による通信を明示的に指定しない限りデフォルトでは pvmd による通信方法が使われる [3]．

2.3.2 PVM に対する拡張

PVM に対して行われている各種の拡張について述べる．

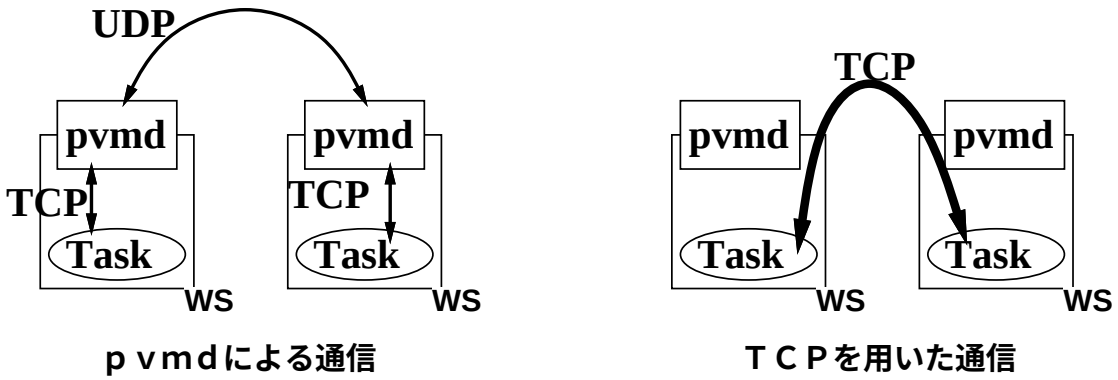


図 2.1: PVM のタスク間通信モデル

TPVM(Threads-oriented PVM)

PVM の処理単位であるタスクをスレッド単位で扱えるようにしたもので，Adam Ferrari らによって開発された [9].

スレッドを利用しているため，プロセスの処理速度が向上するという利点がある.

しかし，PVM からプログラミングインタフェースが変わるという問題点がある.

FastPVM

PARDI と呼ばれる，UNIX と WindowsNT の混在環境でメッセージ通信，RPC を実現するライブラリを用いる PVM で，イギリスの Perihelion 社が開発した.

メッセージ通信のオーバーヘッドが PVM に比べて半分に減少するという大きな特徴を持つが、商用のため PVM システムのようにフリーで入手することが容易ではない.

LPVM(Lightweight process PVM)

LPVM は，SMP(Symmetry array Multiprocessors) などでの使用を目的として開発された.

プログラムはプロセス一つで動作し，プロセスには lightweight process を用いているため PVM に比べて幾らか性能が向上している.

しかし，本研究で用いるワークステーションは単一 CPU であるため，この LPVM は動作させることが出来ない.

2.3.3 MPI(Message Passing Interface)

MPI[10] は，1992 年に MPI Forum により提唱された，主に分散メモリ環境におけるメッセージ通信ライブラリインタフェースの標準仕様である．仕様には世界中の研究機関や，並列計算機のベンダからの意見を取り入れており，機種依存の排除や，多様な通信モードを備えている．

実際に MPI を利用するには，その規格に基づき作成された MPI の実装系を利用する必要がある．次に，この MPI の実装系について述べる．

2.3.4 MPI の実装系

2.3.3で述べたように MPI はメッセージ通信のための仕様であり，PVM のようなプログラムではない．ここでは MPI の仕様を実装した代表的な MPI の実装系について述べる．

MPICH

MPICH は，ANL(Argonne National Laboratory) により開発された MPI の実装である．

MPICH は PVM とは対照的に、プログラムの実行と同時に並列処理を制御するデーモンが起動する．また，処理を行うタスク間の通信方法は TCP による 1 種類のみとなっている．

本研究では MPI の実装系としてこの MPICH を利用する．また、次章以降では断りの無い限り MPI は、この MPICH を指すこととする．

LAM(Local Area Multicomputer)

LAM は，Ohio Supercomputing Center により開発された，MPI の仕様を全て実装した MPI の実装系である．

独自の拡張として，PVM のように動的にマシンの追加/削除を行ったり，マシンの障害の検出や復旧のためのフォルトトレラントの機構が追加されている．

2.4 まとめ

本章ではワークステーションクラスタの性能向上に対して取り組まれている研究について述べてきた．ワークステーションを接続するネットワークに高速なネットワーク機器を使用すれば並列計算機に迫る通信性能を得ることが出来るようになってきた．しかし導入のためのコストがまだ大きいため，容易に利用することが出来ないという問題がある．

また，メッセージ通信ライブラリに対して行われている各種の拡張では，様々な機能が提供される反面，特殊な実行環境を必要とするソフトゥエアであったり，既存のインタフェースと大きく異なり、ソフトウェアの再利用が困難であるなどの問題点がある．

次章では、標準の PVM や MPI を利用したワークステーションクラスタにおける効率の良い並列処理方法を提案するために、10Base-T スイッチハブをネットワークに用いたワークステーションのメッセージ通信性能の評価を行い、その性能の改善方法についての提案を行う．

第 3 章

ワークステーションクラスタにおけるメッセージ通信

3.1 はじめに

ワークステーションクラスタでは，メッセー通信はネットワークを介して行わなければならない．このため，ワークステーションの接続に用いるハブによっては通信による処理時間の増大が処理性能全体の大きなボトルネックとなってしまう．

しかし，メッセージ通信関数のベンチマークを行い，この通信時間を把握することでプログラムを改善し性能の向上を行ったり，新しい並列計算機アーキテクチャ設計のための参考にすることが可能である [12]．

本章ではスイッチハブを用いたワークステーションクラスタを構築し，そのメッセージ通信性能についての評価を行う．

3.2 本研究で用いるワークステーションクラスタ

ワークステーションは Sun SparcStation IPX を 48 台使用した．

SparcStation IPX は，CPU に Sparc 40MHz,Memory 32MByte を搭載したワークステーションで，OS は SunOS 4.1.3 である．

これらのワークステーション間をつなぐハブは 3Com 社の SuperStack II Switch1000 を利用した．

このハブは，ハブとハブのカスケードには 100Base-T のポートを一つ，ワークステーションを接続するポートには 10Base-T スイッチで 24 のポートを持つ．

メッセージ通信ライブラリは PVM と MPI の 2 つを利用した．

3.3 メッセージ通信性能の測定

ワークステーションクラスタでよく使われるメッセージ通信は 1 対 1 通信，バリア同期，ブロードキャストが挙げられる．

これらの通信を測定し，その性能を把握することはプログラムの実行効率の改善やボトルネックの発見に有効である．

そこで本節ではそれらのメッセージ通信に対してベンチマーク [13] を実行し、ワークステーションクラスタにおける PVM と MPI のメッセージ通信性能の評価と検討を行う．

通信性能の測定は，PVM と MPI それぞれについて行った．更に PVM では TCP を用いた通信を行う場合と pvmd による通信を行う場合の 2 種類について計測を行った．

以降では，MPI による結果を mpi，通信に TCP を用いた PVM を pvm_TCP，pvmd による通信を pvm_UDP，処理を行うワークステーションを PE(ProcessingElement) とする．

3.3.1 バリア同期

バリア同期とは，図 3.2に示される様に、全ての PE がこの命令を呼び出すまで処理を停止するもので，PE 間での同期をとったりする場合に使われる．

バリア同期の性能評価は，バリア同期関数の PVM では pvm_barrier()，MPI では MPI_Barrier() を実行し、その同期 1 回に要した時間により行った．

図 3.3にバリア同期の測定結果を示す．グラフの横軸は同期を行う PE 数，縦軸はバリア同期一回あたりの実行に要した時間を表す．

MPI の実行結果が 24PE までしか得られなかったのはこれ以上の台数での測定が出来なかったためである。

8PE までは pvm の方がバリア同期に要する時間が短い結果となった．しかし，pvm は PE 数の増加に伴いバリア同期の時間も大きく増加しているのに対し，mpi は PE 数の増加に伴うバリア同期時間の増加はわずかである．そのため，16PE 時のバリア同期では pvm は MPI のバリア同期時間の 2 倍近い時間がかかっていることが分かった．

図 3.1: 構築したワークステーションクラスタ

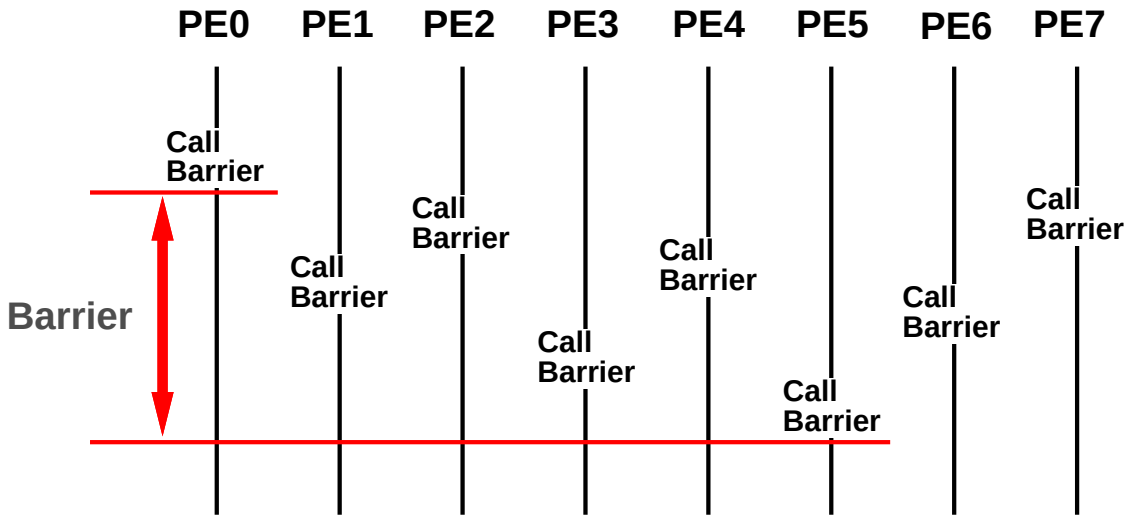


図 3.2: バリア同期の概念図

3.3.2 レイテンシ

レイテンシとは，図 3.4に示される様に PE がメッセージを送信してから次の処理に移ることが出来るようになるまでの時間を表すもので，メッセージ通信におけるオーバーヘッドを示す値としてよく用いられる．

レイテンシの性能評価は，1 対 1 通信関数の、PVM では pvm_psend()，MPI では MPI_Send() を各サイズのメッセージ毎に実行し、送信するのに要した時間により行う．

図 3.5に 1 対 1 通信におけるレイテンシの測定結果を示す．横軸は送信するメッセージのサイズを表し，縦軸はそれに要した時間を表す．

pvm_TCP では，メッセージサイズが 1KByte 以下，mpi では 512Byte 以下までは通信時間に大きな変化が現れていない．これはメッセージを送るための手続きに要する時間が送信にかかる時間よりもはるかに大きいためである．

これによってメッセージ通信時の手続きに要する時間がそれぞれ pvm_TCP は約 1.2msec，mpi では約 0.8msec であることがわかる．

唯一 pvm_UDP はメッセージサイズが 1KByte 以下においてもレイテンシの値が一定してない．これは pvmd が通信処理以外の処理も行っている影響を受けているため，このような値の乱れが生じると考えられる．

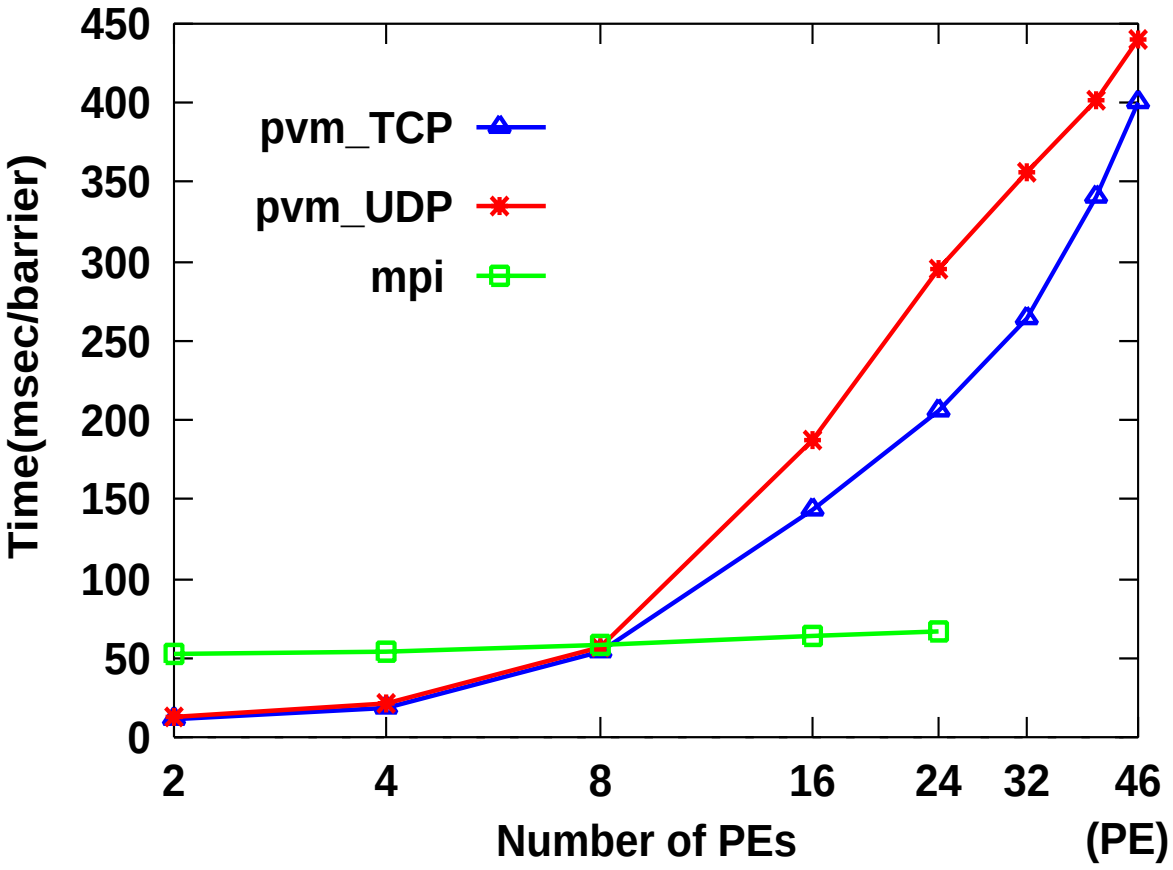


図 3.3: バリア同期の測定結果

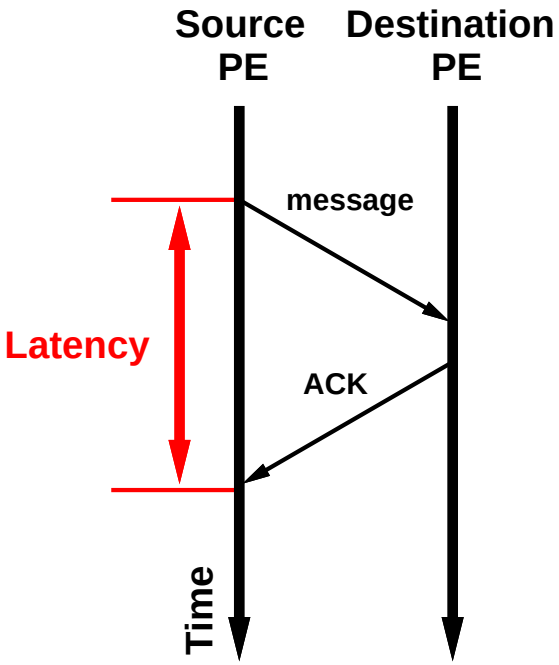


図 3.4: レイテンシの概念図

3.3.3 通信速度

通信速度は，図 3.6に示される様に、1 対 1 通信において一定時間の間にやりとり出来るデータ量を表す.

通信速度の性能評価は，1 対 1 通信関数の PVM では pvm_psend()，MPI では MPI_Send() をそれぞれ実行、1 秒間に送受信されたメッセージサイズにより行う.

図 3.7に 1 対 1 通信時の通信速度を示す．横軸は送信するメッセージサイズ，縦軸は 1 秒あたりに送信したデータ量を示す.

MPI が最も速く、最高で約 800KB、PVM_TCP が約 500KB、PVM_UDP が約 200KB というはっきりと異なった結果となった．PMV と MPI の差は両者の 1 対 1 通信関数の実装によるものと考えられる。PVM_TCP と PVM_UDP の差については、PVM_TCP ではタスク間で TCP のコネクションを用いて通信するのに対し PVM_UDP は一旦 pvmd を解してタスク間のデータのやりとりを行うため、この様な差が出るものと考えられる。

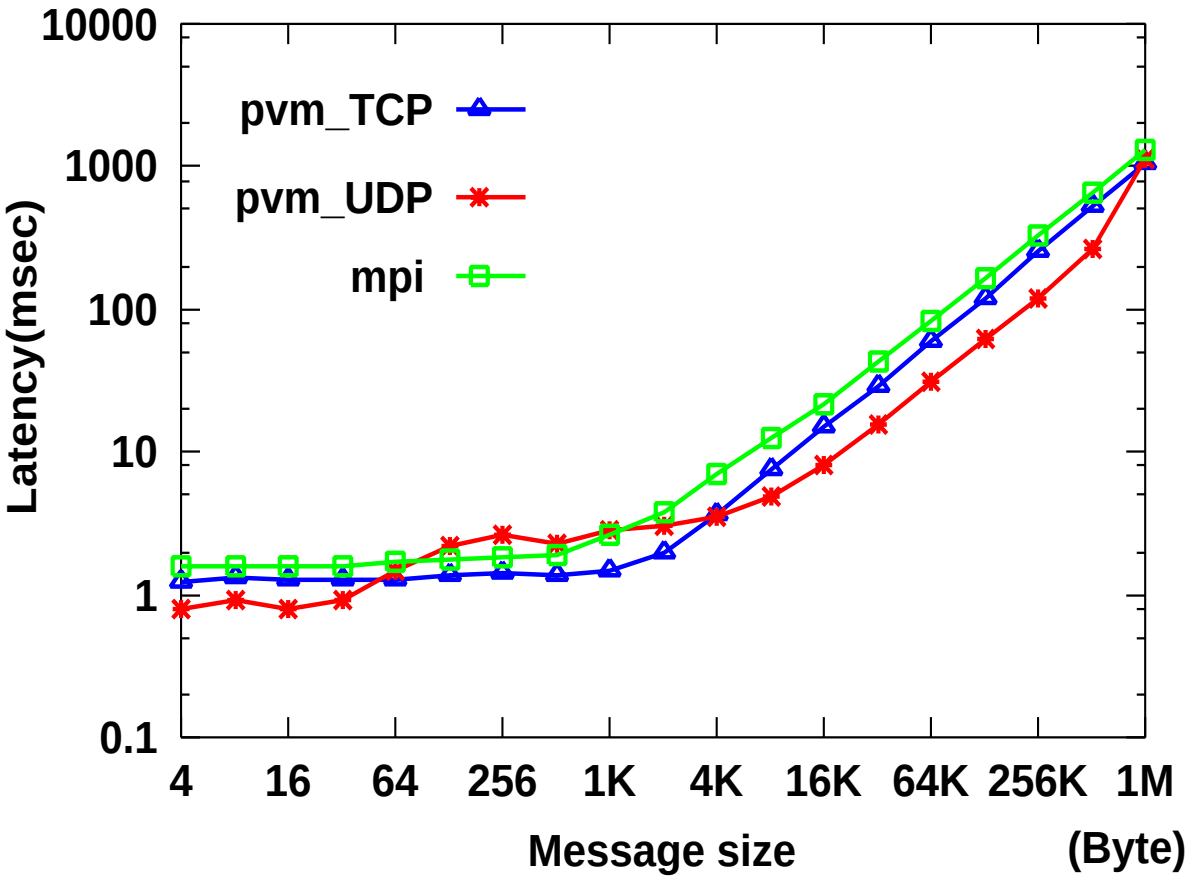


図 3.5: レイテンシの測定結果

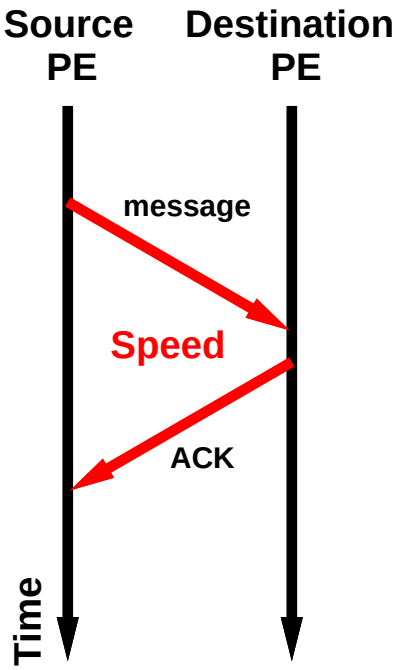


図 3.6: 通信速度の概念図

3.3.4 ブロードキャスト

ブロードキャストは図 3.8に示される様に、全ての PE へデータを送信する命令で、あるデータを動作している全ての PE へ送信する場合や，マスタースレーブ型の処理で，マスター PE からスレーブ PE へのデータ送信などに使われる．

ブロードキャストの性能評価は，ブロードキャスト関数の PVM は pvm_bcast()，MPI は MPI_Bcast() を実行し、1 秒間に送受信されたメッセージサイズにより行う。

図 3.9に、16PE によるブロードキャストの実行結果を示す．

mpi が pvm_TCP，pvm_UDP に対して 4 倍近い性能を得ている．また，4K〜256KByte でそれぞれで得られた通信速度については，1 対 1 通信時の結果の約 1/16(pvm_TCP，pvm_UDP)，1/4(mpi) となっている．

両者の結果の違いは，ブロードキャストに用いるアルゴリズムが PVM では最小木，MPI では n 分木とそれぞれ違うためと考えられる．また mpi ではサイズが 4KByte になると値が下降している．これは MPI がデータの読み込みをページサイズ単位で行っていること，本 WS クラスタで用いている SunOS4.1.3 のページサイズが 4KByte であるためである．

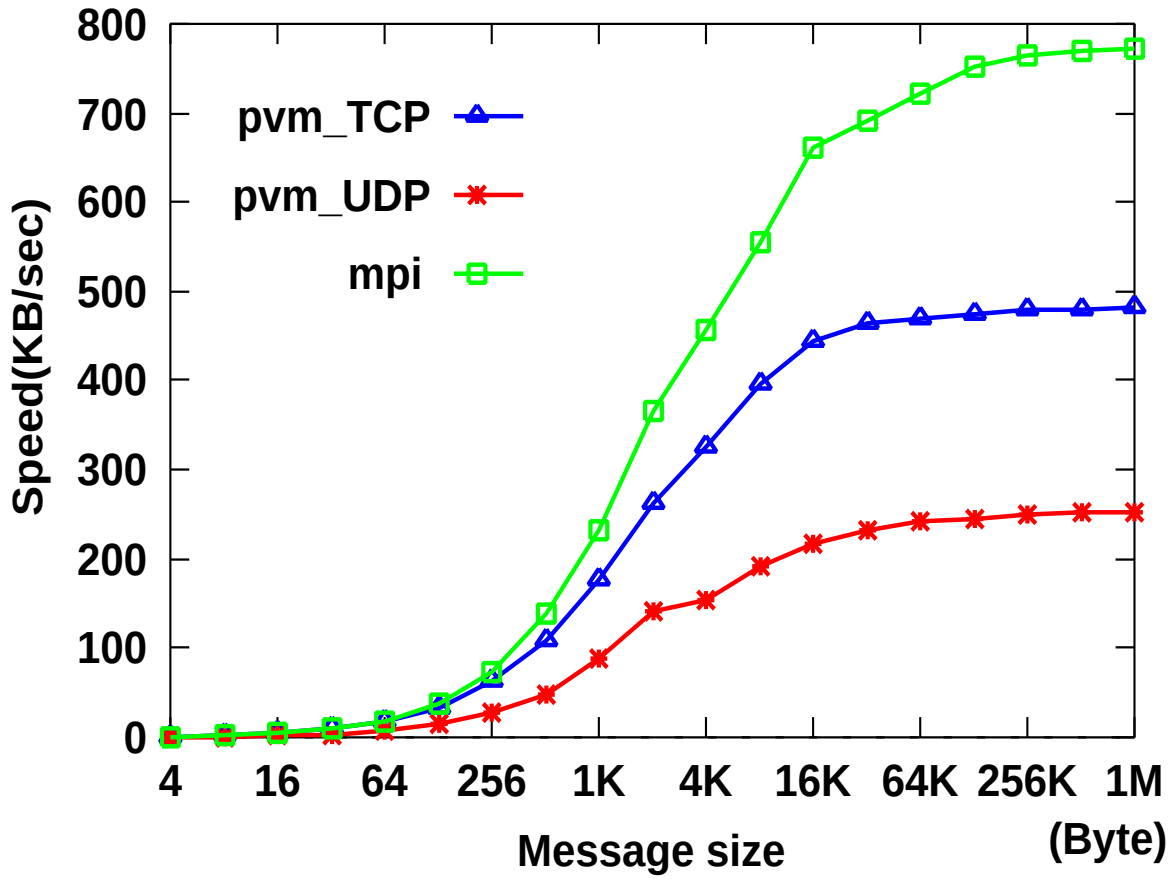


図 3.7: 通信速度の測定結果

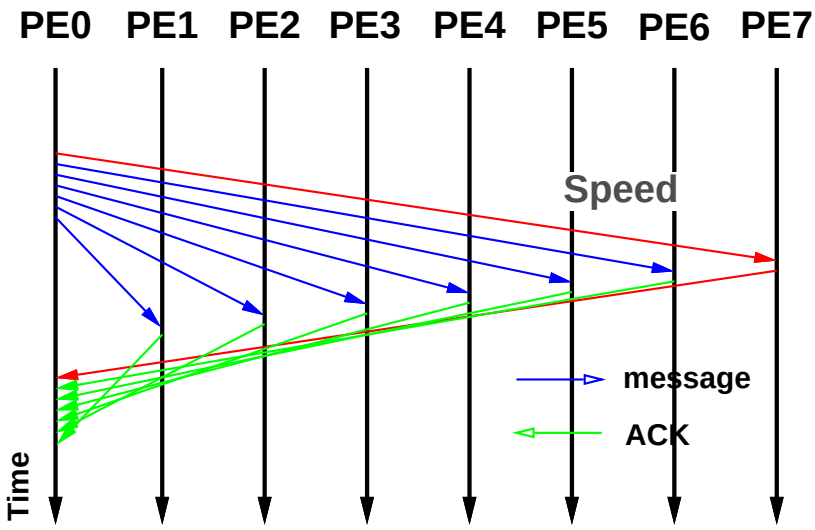


図 3.8: ブロードキャストの概念図

3.4 階層型ブロードキャストによるメッセージ通信

これまでの結果により，PVM におけるブロードキャストが著しく性能が悪いということが分かった．そこで本研究ではワークステーションクラスタに用いているスイッチハブを用いた階層型ブロードキャストを提案する．

3.4.1 スイッチハブと従来のハブの通信速度

スイッチハブは従来のハブと違い、複数の通信を同時に行ってもその通信速度が減少しないという特徴を持っている．

図 3.10にスイッチハブと従来のハブの，1 対 1 通信の増加に伴う通信速度の変化を示す．

これよりスイッチハブは 1 対 1 通信を同時に複数行っても従来のハブのように通信速度が大きく減少しないということが分かる．

この結果を元に階層型ブロードキャストについて提案する．

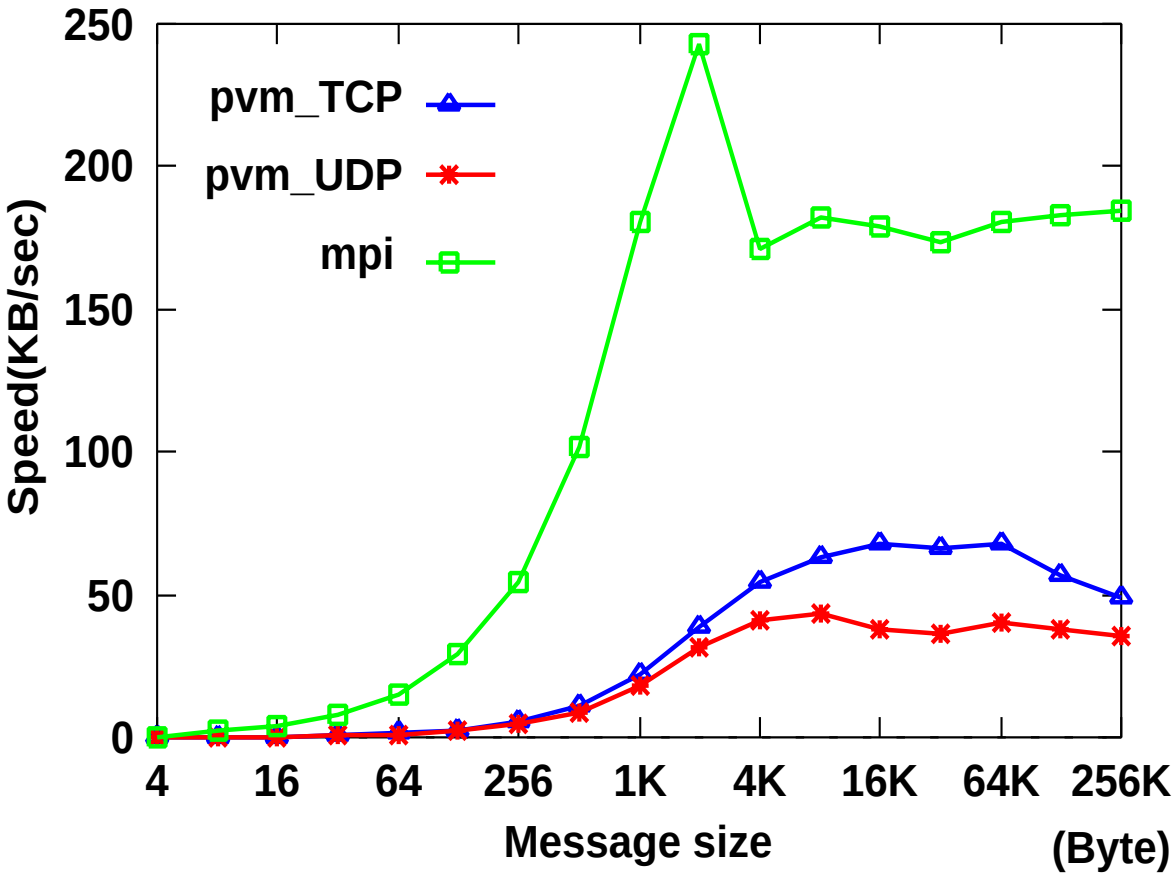


図 3.9: 16PE でのブロードキャストの測定結果

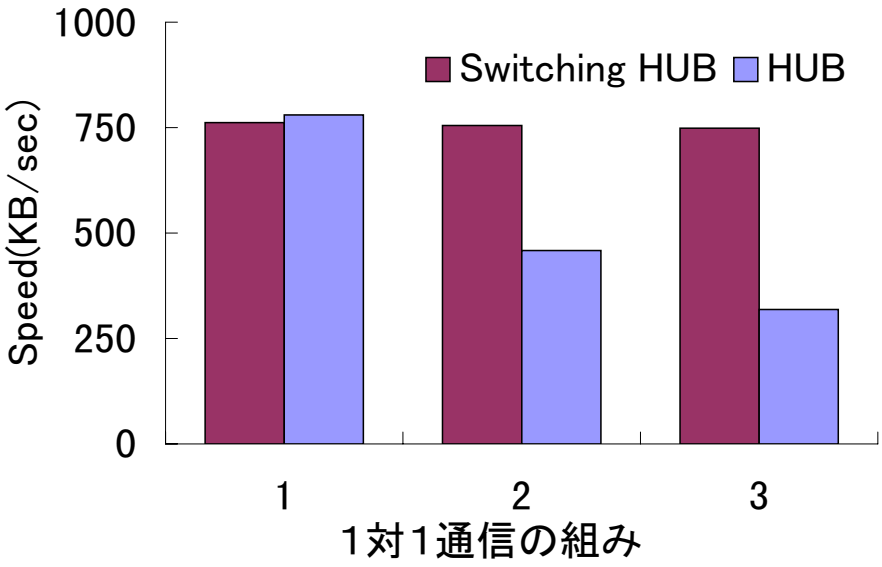


図 3.10: スイッチハブと従来のハブの比較

3.4.2 階層型ブロードキャスト

階層型ブロードキャストは，スイッチハブが同時に複数の通信が行われてもその通信速度が減少しない特性を利用し，一度に複数の PE でメッセージ通信を行い全ての PE へデータを送信するものである．

図 3.11に，8PE 時における階層型ブロードキャストにおけるデータの配送の流れを示す．

データを受けとった PE はまだデータを受けとっていない PE へ受けとったデータを送信 (転送) する．この作業を繰り返してブロードキャストを行う．この手法ではブロードキャストが $log_2 PE$数のステップ数で行うことが可能である．

このブロードキャストアルゴリズムを 1 対 1 通信の関数を利用してプログラムし，pvm_bcast() と比較した．

図 3.12と図 3.13にその結果を示す．pvm_bcast() は pvm_bcast()，hierarchical は階層型ブロードキャストの結果である．

256KByte のブロードキャストでは PE 数が 8 になった時点で pvm_bcast() が hierarchical を下回る結果となっている．

1MByte のブロードキャストでは 4PE の時点で pvm_bcast() が，hierarchical を下回り，

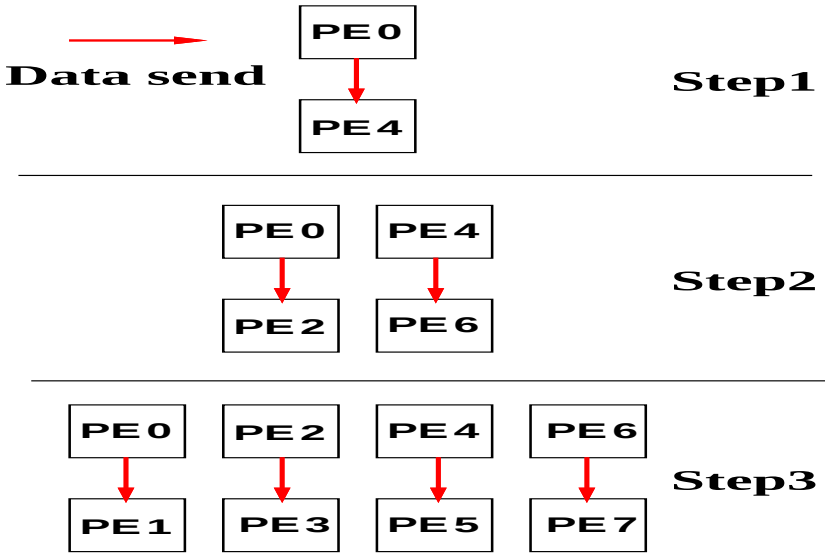


図 3.11: 階層型ブロードキャスト

16PE では pvm_bcast() と hierarchical の差は 4 倍近くになっている．

この様に階層型ブロードキャストを利用することで pvm_bcast() に比べて高速なブロードキャストを行うことが可能であることが分かった．しかし，これは 8PE では 256KB 以上，4PE では 1MB 以上のデータのブロードキャストを行わなければ効果がない．

これは，階層型ブロードキャストが PVM の提供するメッセージ通信ライブラリを用いて行われていること，各ステップにおけるメッセージのやりとりにおけるオーバーヘッドが発生し，性能の低下を招いていると考えられ。この問題の改善が課題として挙げられる．

階層型ブロードキャストの改善

提案した階層型ブロードキャストは多数の PE や，大きなサイズのメッセージの場合効果が大きいが，少ない PE や，小さなメッセージでは pvm_bcast() の速度よりも下回ってしまった．

これを解決する方法としてブロードキャストする PE 数やメッセージのデータサイズの大きさによって pvm_bcast() と階層型ブロードキャストを使い分けるようにすれば両者の良い性能を持つブロードキャストが実現されると考えられる．

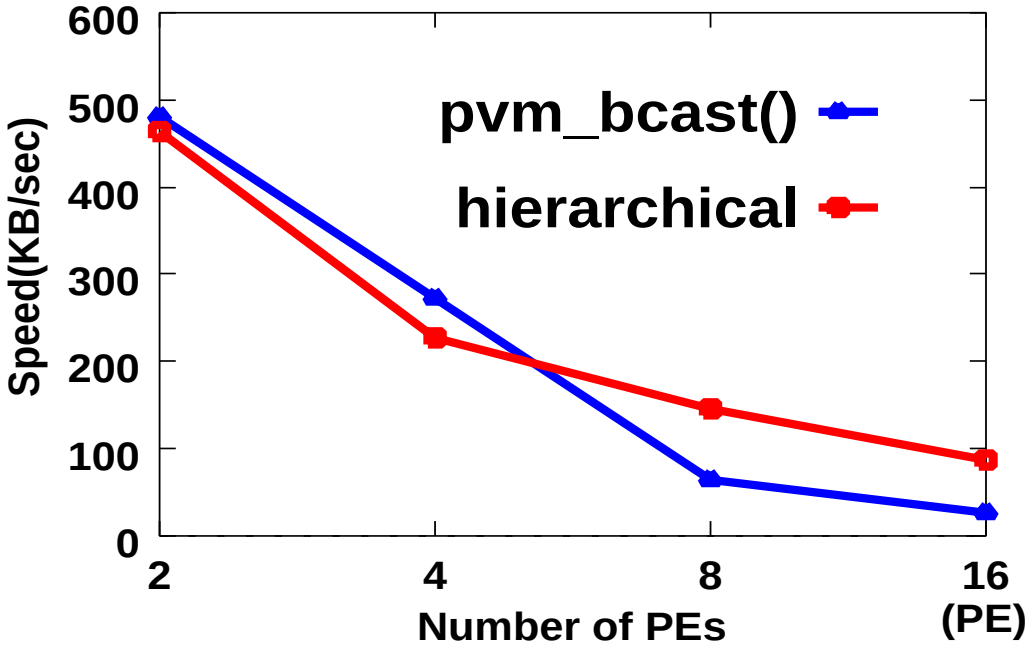


図 3.12: 256KB データのブロードキャスト

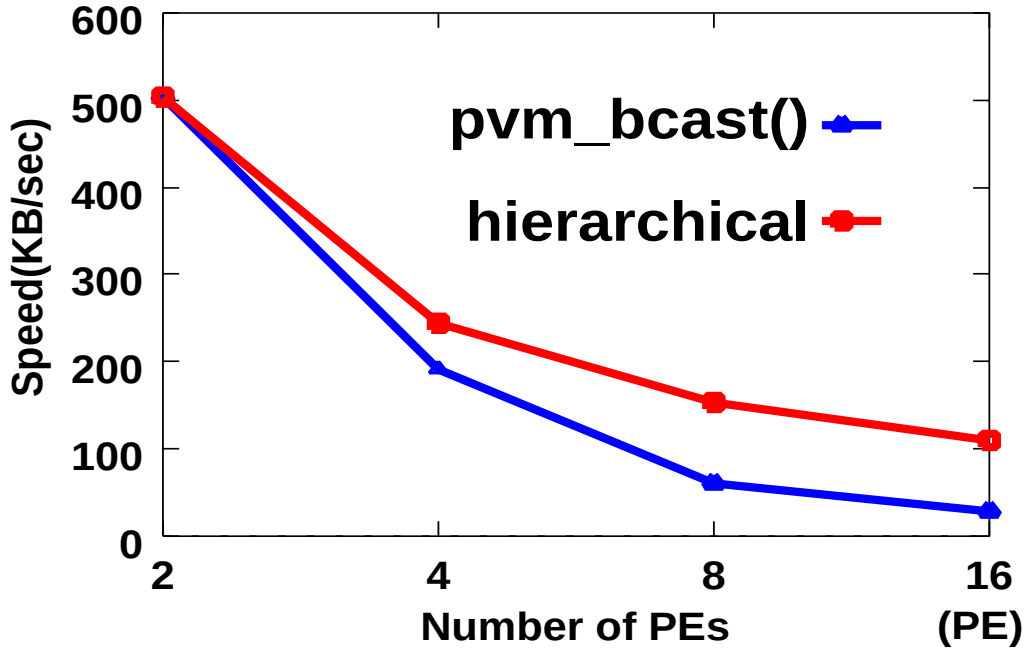


図 3.13: 1MB データのブロードキャスト

3.5 まとめ

本章ではPVMとMPIを用いてワークステーションクラスタを構築し，メッセージ通信性能の測定を行った．

PVMではTCPによる通信を使えば良い性能を得られることが分かった[3]．しかし，TCPによる通信は常にできるとは保証されていない[3]のでTCPによる通信が出来ないような場合は通信性能が低下してしまうことに注意しなければならない．

MPIは，測定した全ての点においてPVMよりも優れた結果を得た．しかし，MPIでは30台以上での実行が出来ないなど，ワークステーションクラスタに用いるメッセージ通信ライブラリとしては致命的ともいえる不具合も確認された．

メッセージ通信性能の測定結果よりPVMのブロードキャスト性能を向上させる手法として，スイッチハブの特性を利用した階層型ブロードキャストを提案した．多数のPEや大きなデータのブロードキャストではこの階層型が有効であることが確認された．またこの階層型ブロードキャストとPMVのブロードキャストをPE数とデータサイズによって使い分ける選択型ブロードキャストについて提案を行った．

次章ではワークステーションクラスタにおける並列処理性能についての検討を行う．

第 4 章

ワークステーションクラスタの並列処理性能

4.1 はじめに

前章の結果より，10Base-T スイッチングハブを用いたワークステーションクラスタのメッセージ通信性能が分かった．

本章ではソーティング，N クィーン問題，TSP をワークステーションクラスタで実行し，その処理性能について検討し，その結果からワークステーションクラスタに有効な処理方法についての提案を行う．

4.2 多量のメッセージ通信を行う処理

4.2.1 ソーティング

ソーティングは与えられたデータを昇順に並べ替える処理で，様々なプログラムの中で用いられるアルゴリズムでもある．

本研究では次のように並列ソートを行う．

1. ソートするデータの生成とスレーブへの分配
2. 各スレーブ PE でクイックソートを実行

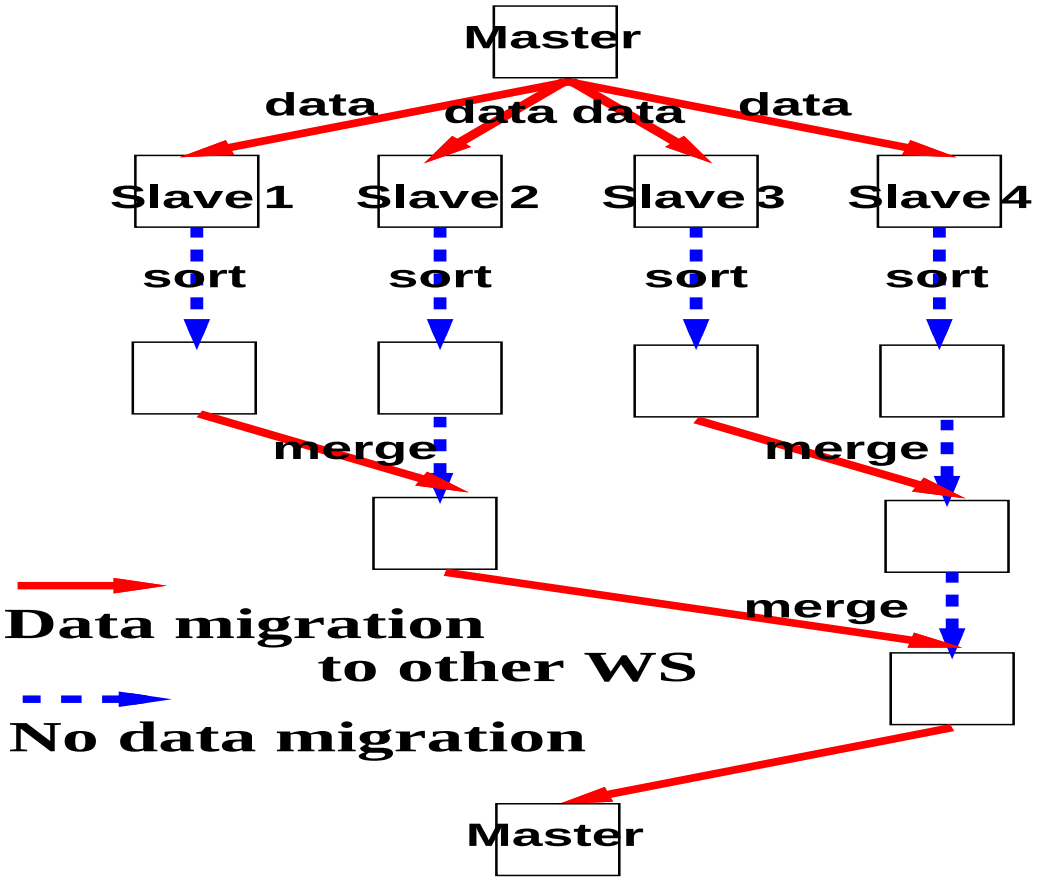


図 4.1: ソーティングにおけるデータの流れ

3. ソートしたデータをスレーブ PE 間でマージする

4. 全てマージしたデータをマスターへ送信

5. ソートされたデータのチェック

1 と 5 の処理をマスターが行い，2〜4 をスレーブが行う．
図 4.1に，ソーティングのアルゴリズムにおけるデータの流れを示す．

4.2.2 ソーティングの実行

このソーティングを 100 万個のデータで PVM と MPI それぞれについて実行し，ソートするデータの配布，ソート処理，マージ処理に要した時間，通信が行われた時間につい

て測定した．

実行結果

図 4.3に WS クラスタ上で PVM と MPI を用いて 100 万個のデータ (サイズ:4Mbyte の float) をソートした処理実行時間を示す．実行時間は，データ送信 (Datasend)，ソート (Sorting)，マージ処理 (Merge(Comp))，マージにおける通信 (Merge(Comm)) の処理時間の割合を示した．

ソート部分の並列化によりソート時間が線形的に減少しているが，同時にマージにかかる時間が増加している．

マージにかかる時間のうち 2/3 近くが通信に費やされており，これがマージの時間増大の大きな原因と考えられる．

実行時間に関しては，MPI の方がわずかに PVM より速いが，ベンチマークで得られたような差はほとんど現れなかった．

処理速度向上比については，それぞれ最高で PVM が 1.22 倍 (16PE 時)，MPI が 1.35 倍 (8PE 時) しか得られないという結果となった．

4.3 メッセージ通信を伴わない処理

4.2節において，メッセージ通信量が多いとワークステーションクラスタの処理性能があまり向上しないという結果が得られた．

本節ではメッセージ通信を伴わない並列処理を行い，その処理性能についての検討を行う．

4.3.1 N クィーン問題

N クィーン問題とは，$n \times n$ のチェス盤において，n 個クィーンの駒を置き，それが互いの駒の移動する筋に当たらないような配置を全て捜し出すことを問題としている．

図 4.4および図 4.5に，$n = 4$ 場合の例を示す．図 4.4は，3 個置いた時点で残りの 1 つが置けなくなっている．図 4.5は，4 つ全てを置き，それぞれの駒の移動する筋には当たっていない．

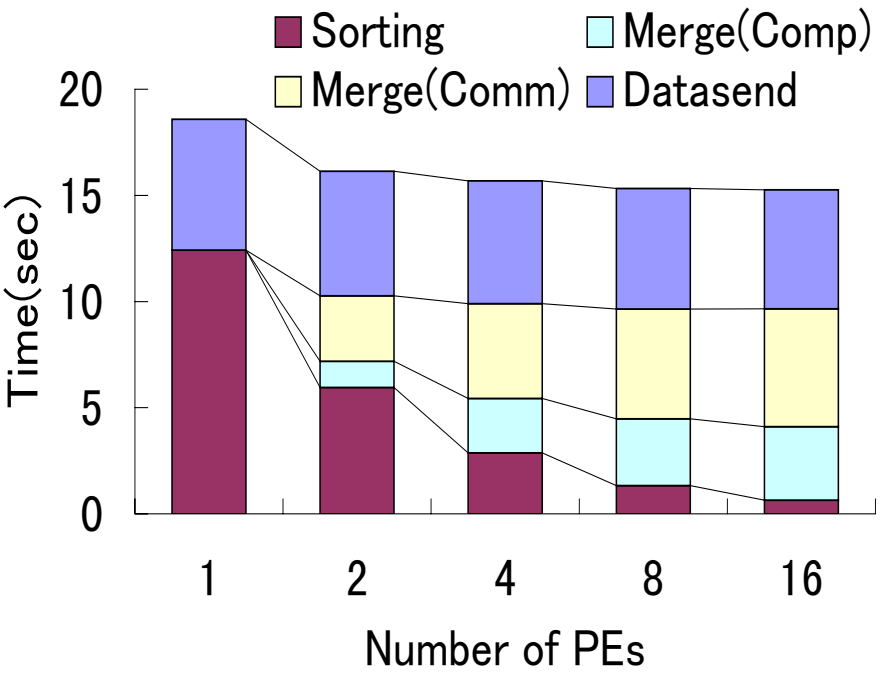


図 4.2: データ分配，ソート，マージの実行時間 (PVM)

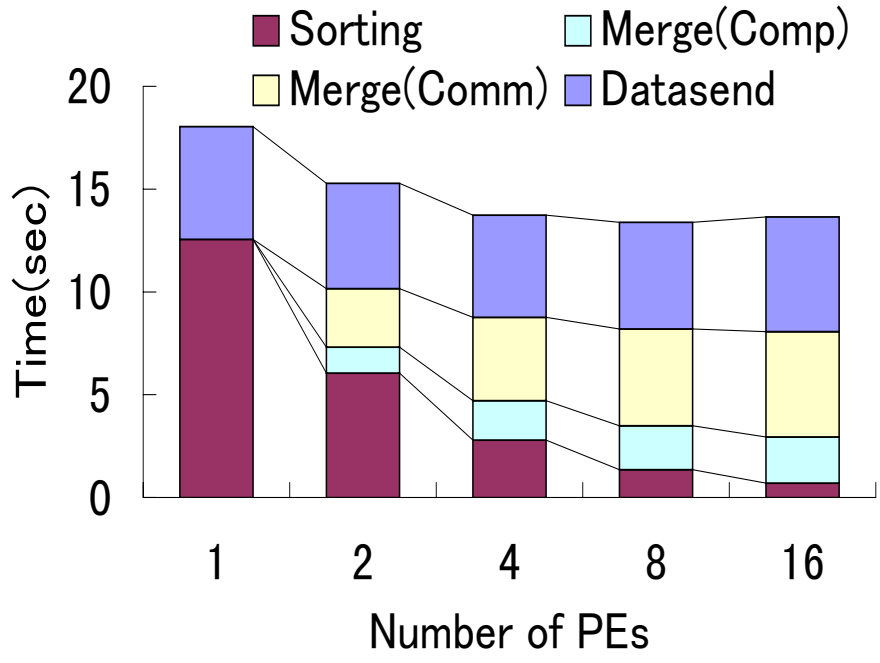
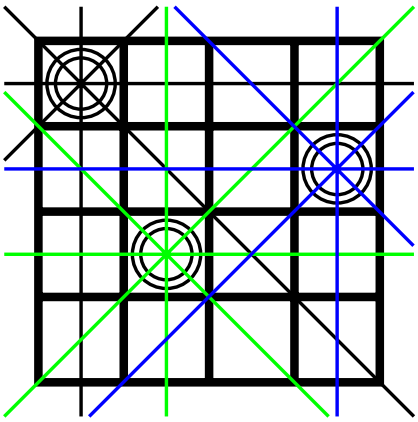
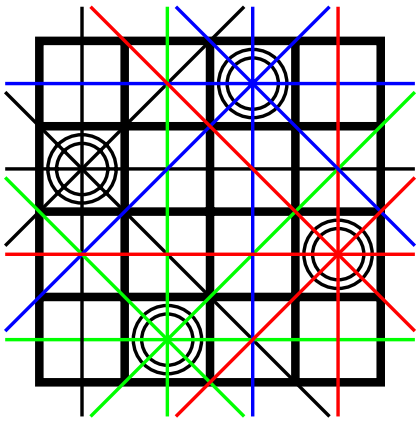


図 4.3: データ分配，ソート，マージの実行時間 (MPI)



Failed

図 4.4: $4x4$ での N クィーン問題の例 (失敗)



Succeede

図 4.5: $4x4$ での N クィーン問題の例 (成功)

N クィーン問題を解くためには，N 個のクィーンの配置を全て調べることだが，これは明らかに効率が悪い．そこで駒を置く度に，次にどこかの位置に駒を置けるかどうかを調べながらとく方法が考えられる．この方法によって，N クィーン問題は n 分木として表し木の探索問題として捉えることが出来るようになる．さらにこの木の探索では，解の得られないような配置を示す木は解が得られないことが分かった時点で探索を打ち切り，新しい木の探索を行う．

$n = 4$ の場合は次のようになる．

root から 1 の木に探索を行った場合，そこから先に進めるのは 3 か 4 である．順番に探索を続けて 3 に到達するがこの時点で解が得られないことが分かる．

N クィーン問題のアルゴリズム

実行する N クィーン問題のアルゴリズムは，n 分木の探索を現在駒がどこに置かれているかという情報を元に探索を行い，これ以上置けないことが判明した場合は探索を打ち切り，一つ上の木へと戻り，探索を続ける．

4.3.2 N クィーン問題の実行

前節で述べたアルゴリズムに対して，本研究では探索木のルートから発生している木の探索を各 PE に割り当てて探索の実行時間を計測した．

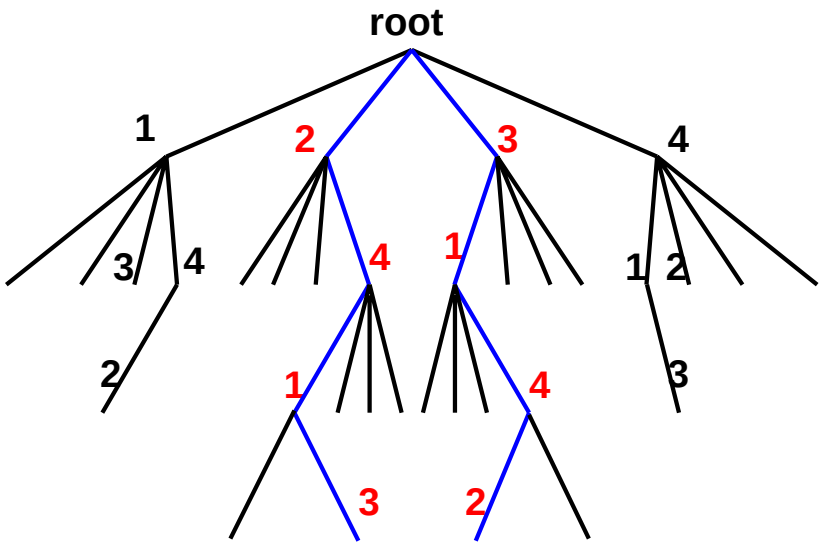


図 4.6: $N = 4$ の N クィーン問題の探索木へのマッピング

N=16 では，1,2,4,8,16PE による実行，N=14 では，1〜14PE それぞれでの実行を行い，各ノードで得られる解の数とそれに要した時間を表 4.1〜表 4.4に示す．

また図 4.7に速度向上比を示す．

実行結果より，N=16 の場合では動作台数とほぼ一致する実行速度の向上が得られた．一方 N=14 の場合の実行では，PE が 2〜7，7〜13 までは線形的な速度向上が得られなかった．これは，負荷の割当をノードの 1 レベルまででしか行っていないため，8PE での実行では，1 つのノードを解いて終了する PE と 2 つのノードを解いて終了する PE が現れる．そのためにこの様な結果になると考えられる．

4.4 ワークステーションの協調動作による処理

4.4.1 Travelling Salesman Problem

TSP(Travelling Salesman Problem) は，N都市を 1 度ずつ通り再び出発した地点に戻って来る最短巡回路を求める問題で，NP 完全問題に属する．

このため N が大きくなると計算時間が非常に大きくなるという性質を持っている．

本節では TSP を並列実行し，その処理性能についての検討を行う．

表 4.1: N=16 における各ノードで得られた解の数 (16PE での実行)

ノード	1	2	3	4	5	6	7	8
解の数 (個)	436228	569531	736363	892999	1050762	1160280	1249262	1290831

ノード	9	10	11	12	13	14	15	16
解の数 (個)	1290831	1249262	1160280	1050762	892999	736363	569531	436228

表 4.2: N=16 における各ノードでの計算時間 (16PE での実行)

ノード	1	2	3	4	5	6	7	8
時間 (sec)	1102	1119	1136	1150	1149	1154	1148	1152

ノード	9	10	11	12	13	14	15	16
時間 (sec)	1153	1155	1153	1150	1142	1132	1120	1103

表 4.3: N=14 における各ノードで得られる解の数 (14PE での実行)

ノード	1	2	3	4	5	6	7
解の数 (個)	11892	16488	23024	27494	32163	34760	36977

ノード	8	9	10	11	12	13	14
解の数 (個)	36977	34760	32163	27494	23024	16488	11892

表 4.4: N=14 における各ノードでの計算時間 (14PE での実行)

ノード	1	2	3	4	5	6	7	8	9	10	11	12	13	14
時間 (sec)	57	64	69	71	73	74	80	81	80	78	70	74	64	57

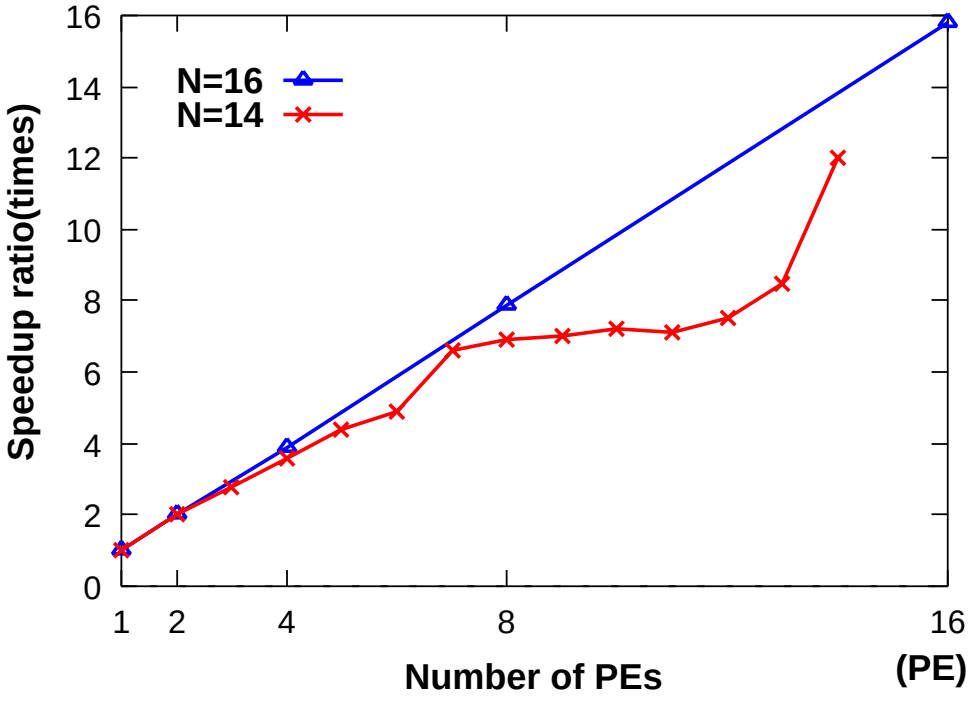


図 4.7: Nqueen 問題の実行速度向上比

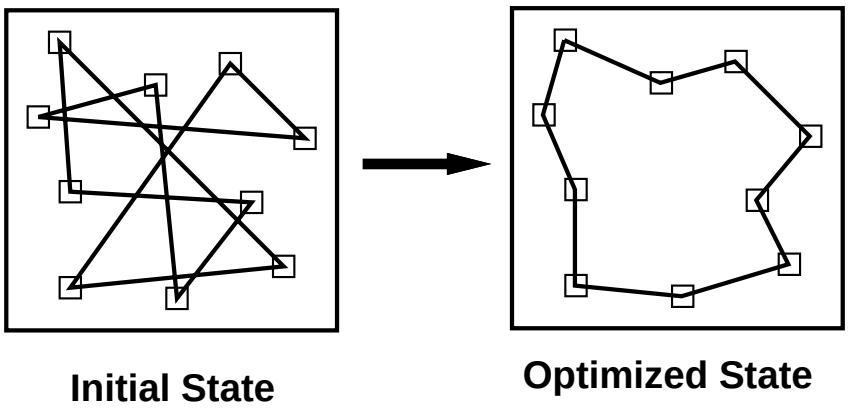


図 4.8: TSP(Travelling Salesman Problem)

図 4.9: TSP による巡回路最適化の例

TSP のアルゴリズム

TSP において一つの最短巡回路を捜し出すアルゴリズムは以下の通りである.

1. 初期配置を与える

   配置されている都市に対して巡回順序を与える.

2. 巡回順序の入れ換え

   巡回順序の適当な所を抜き出して，抜き出した順序を反転させて元の場所に戻すか，別の場所へ挿入する．どちらの操作を行うかはランダムで決まる.

3. 入れ換えた順序の有効性の検討

   入れ換えた後の巡回路が入れ換える前よりも短くなるかどうか評価する．入れ換えた順序が有効であれば実際に入れ換えを行う．入れ換えた順序が入れ換え前よりも長い経路になる場合は入れ換えは行わない.

4. 入れ換えを行っても経路が短くならない

   現在の順序を最短経路として出力

このアルゴリズムで 64 の都市において 64 個の解を見つけるまでの実行時間を中心としたワークステーションクラスタにおける TSP の処理性能の検討を行う.

4.4.2 単一 PE による TSP の実行

並列化による処理性能向上の度合を比較するために，まず単一の PE で TSP プログラムを実行した.

実行は，64 の都市の最適巡回経路を 64 種類見つけるまで実行し，実行に要した時間，TSP プログラムの実行回数，発見した解の数，同じ解を見つけた回数について計測した.

単一 PE での実行結果

表 4.5に実行結果を示す.

64 都市における 64 個の最適解を見つけるまでには約 1 時間半かかることが分かった.

また，TSP 実行回数より解が 40%の確立で発見されることが分かる.

時節ではこの結果を元に，複数の PE で実行した場合の処理性能について検討する.

表 4.5: 1PE による TSP の実行結果

時間 (sec)	TSP 総実行回数	発見した解	同解数	失敗数
5512	227	64	26	137

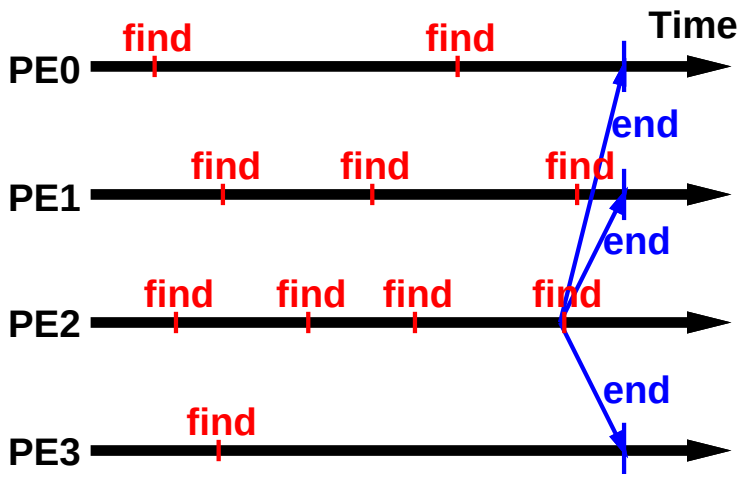


図 4.10: 分散型の概念図

4.4.3 分散型による TSP の実行

分散型実行は図 4.10に示される様に、TSP プログラムを複数の PE で同時に実行させ，64 個の解を見つけた時点で全ての PE が処理を終了するものである．

この分散動作は 1 台の PE が TSP の実行を複数回行うところを複数台の PE で同時に実行しているに過ぎず，実行時間は単一 PE で行なうものと殆んど変わりがないと予想される．しかし，TSP の解はランダムに都市の巡回順序を入れ換える作業の末に発見されるので，巡回順序の入れ換えがうまくけば，はやく解を見つけることが出来る可能性がある．

そのため，複数台で実行することにより，解をよりはやく多く見つけられた PE が現れる場合がある．これは，動作台数が多ければ多いほどそのような結果を得られる PE が存在する可能性が大きくなると考えられる．

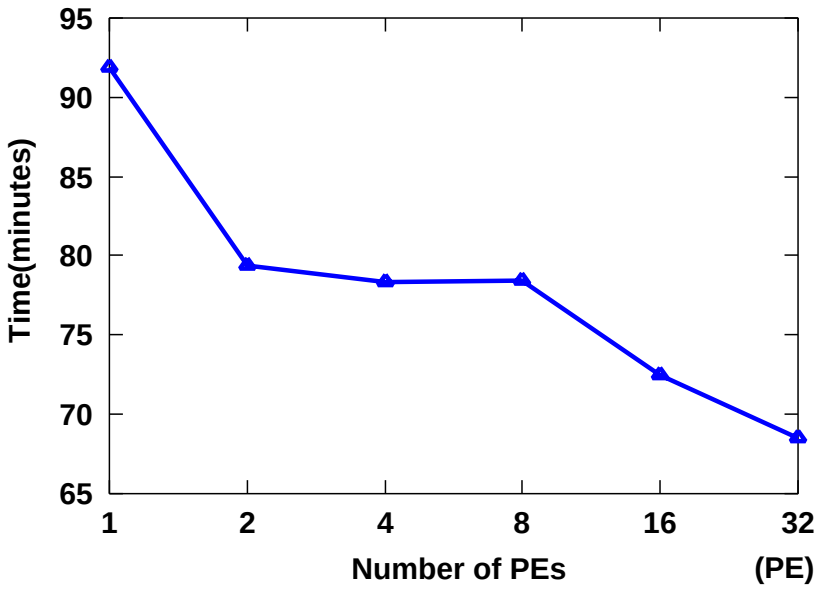


図 4.11: 分散型における TSP の実行時間

分散型での実行結果

分散型動作による実行結果を図 4.11〜図 4.13に示す.

PE 数 1 の結果は 4.4.2で測定した結果である.

予想通り処理速度の向上は幾分あったものの，32 台実行時でわずか 1.35 倍しか得られないものであった．しかし図 4.13を見ると，動作 PE 数が増えるにしたがって TSP の実行回数は減っているのが分かる．これはランダムな都市の入れ換えの出来によって 1 つの PE での TSP 実行時間が変わるということを意味する。

時節ではマスタースレーブ型で TSP を実行し，その処理性能について検討する.

4.4.4 マスタースレーブ型による TSP の実行

マスタースレーブ型実行は，図 4.14に示されるように，スレーブ PE が発見した解を収集し，64 個に達した時点で処理を終了させるものである．スレーブ PE は解を発見する度にマスター PE へ送信する。マスター PE は受けとった解が重複していない解であるかどうかを調べ、重複していない場合はその解を追加、重複していた場合はその解を破棄する。またマスター PE 自分でも解を探すという動作を行う.

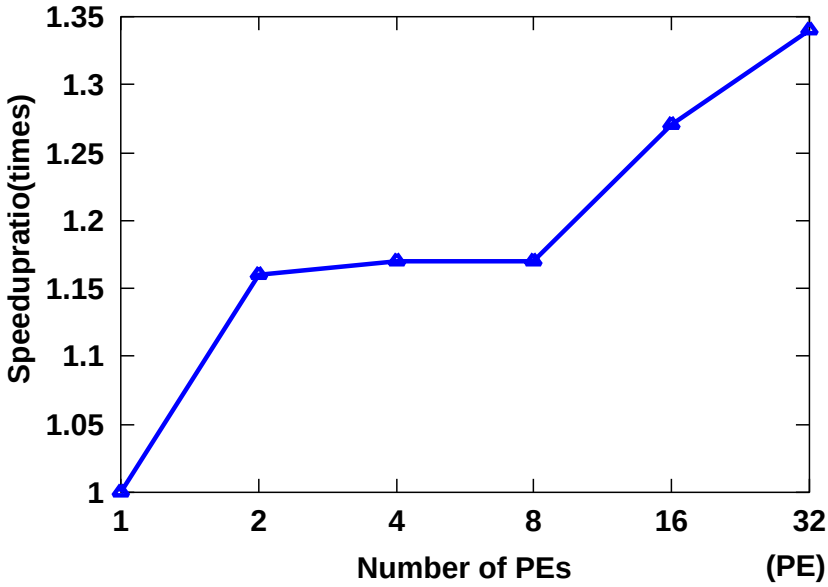
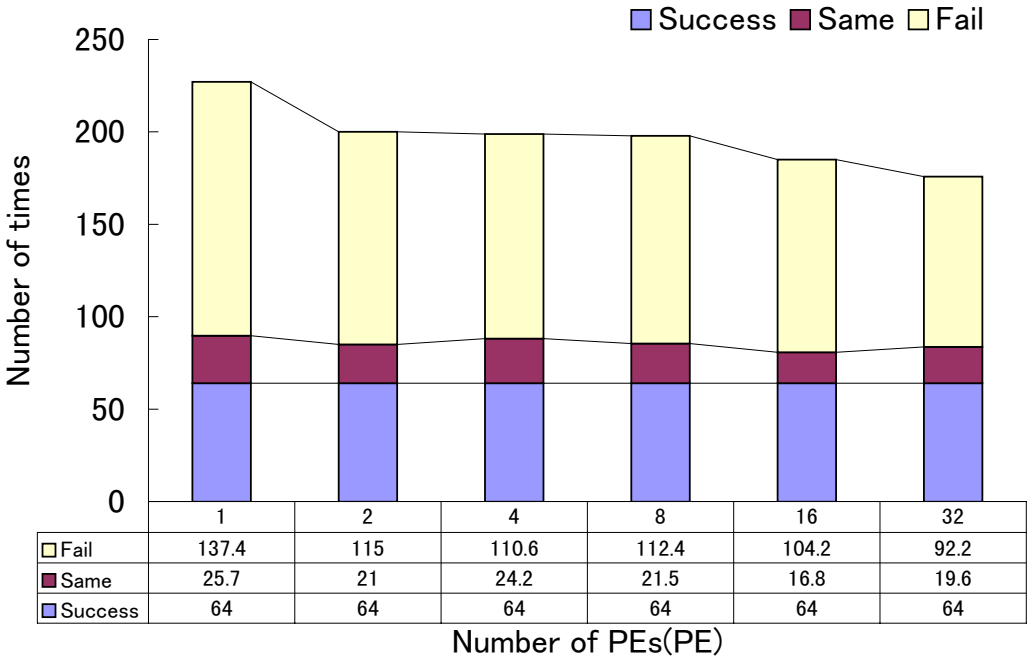


図 4.12: 分散型における速度向上比



	1	2	4	8	16	32
Fail	137.4	115	110.6	112.4	104.2	92.2
Same	25.7	21	24.2	21.5	16.8	19.6
Success	64	64	64	64	64	64

図 4.13: 分散型における TSP の実行回数

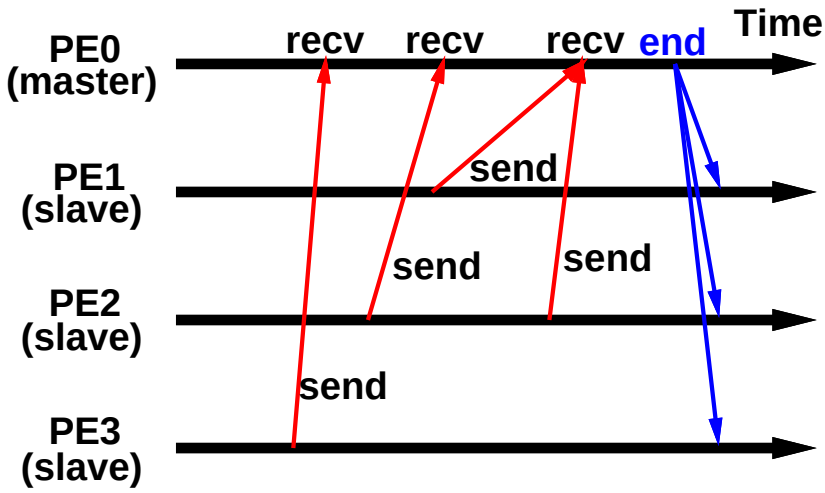


図 4.14: マスタースレーブ型の概念図

これによって各 PE は $64/PE$数の解を発見すればよいことになる．

解の収集方法

スレーブからマスターへの解の送信する時のタイミングについて以下のように幾つかの方法が考えられる．

1. 一定数の解を見つける度にマスターへ送信
2. 解をマスターへ送信するたびに発見しなければいけない解の数を半分に減らしてゆく．
3. 1 回送信するたびに発見しなければいけない解の数を 2 倍に増やしてゆく．

これらの内 1 のポリシーについて，

- 一つ解を見つける毎にマスターへ送信
- 目標とする数値 $Nanswer$を PE 数 $Nproc$ で割った，$N = Nanswer/Nproc$ 個の解を見つけるたびにマスターへ送信

のような条件で実行した．
また，解の送信には PVM の 1 対 1 通信を行う pvm_send() を用いた．

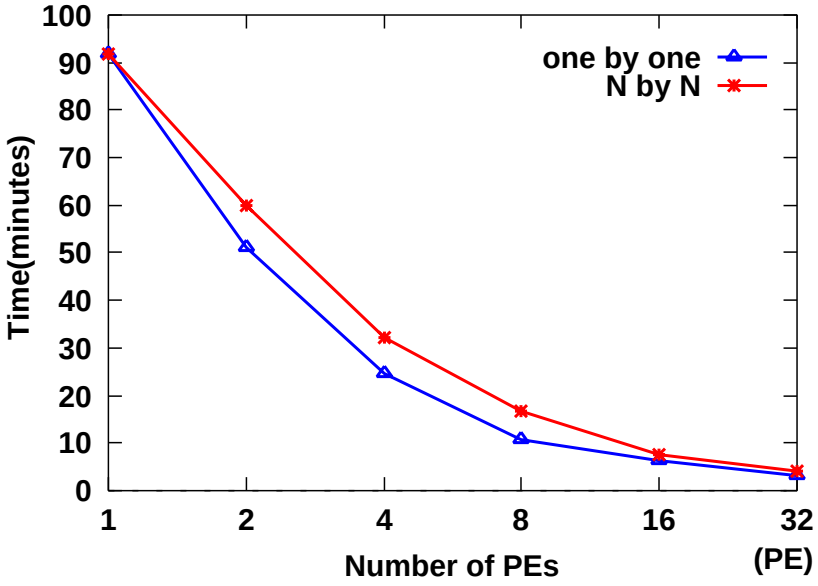


図 4.15: マススレーブ型における TSP の実行時間

マススレーブ型での実行結果

マススレーブ型での実行結果を図 4.15〜図 4.20に示す.

グラフの”one by oen” は，一つの解を見つける度にマスターへ送信した結果，“N by N” は $Nanswer/Nproc$ の解を見つける度にマスターへ送信した結果を示す．また、図 4.17と図 4.18の総メッセージ通信回数と通信量は TSP プログラムを実行し、64 個の解を得るまでに PE 全体で行われた通信回数と通信量を表す。

全体的に $NbyN$ の方が速度向上比が低い結果となった．これは各スレーブ PE が発見した解が重複していた場合，1 度の送信ではマスターが 64 個の解を得て終了できないためである．そのため，各スレーブ PE は再び TSP の実行を行い，再び解を送信しなければならないことになる．

図 4.19図 4.20の表より $NbyN$ の方が TSP の実行回数や解を見つけた数が多いことからもこのことが示される。

PE 数が少ない場合は，重複して計上されなかった分の解をマスター PE 自身が発見しすぐに終了しているため，差が殆んど見られないが 32PE 時の様に PE 数が多ければスレーブからの送信でも十分終了が可能であるが，$NbyN$ では 2 つの解を見つけるまで送信が出来ず，余計な時間がかかってしまうためであると考えられる．

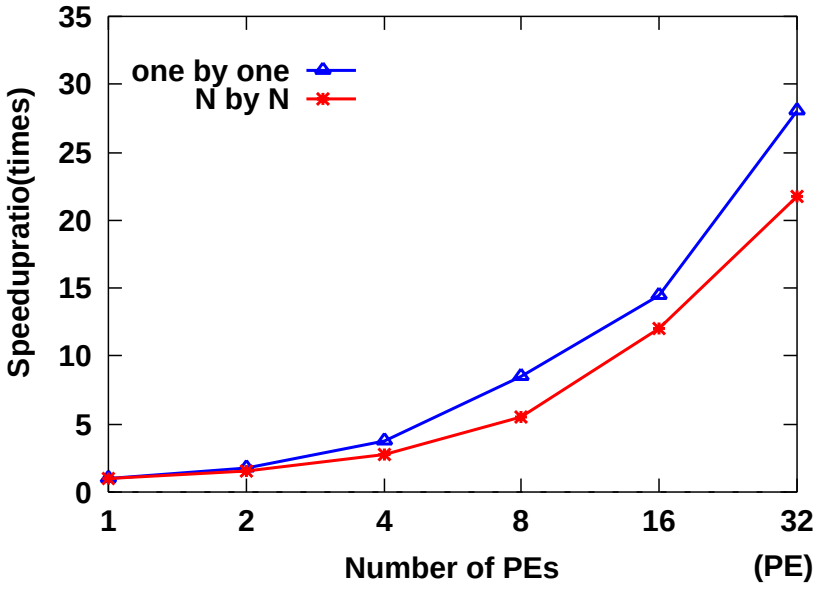


図 4.16: マスタースレーブ型における速度向上比

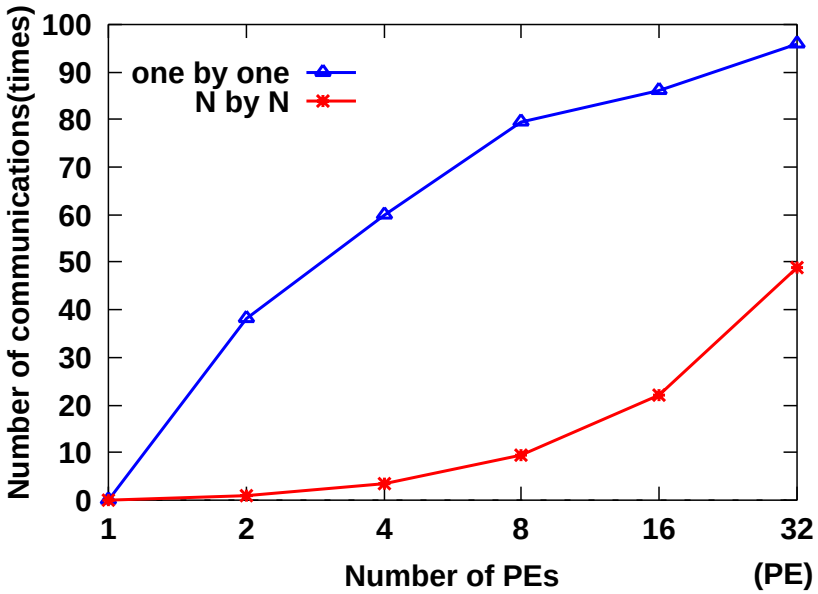


図 4.17: マスタースレーブ型における総メッセージ通信回数

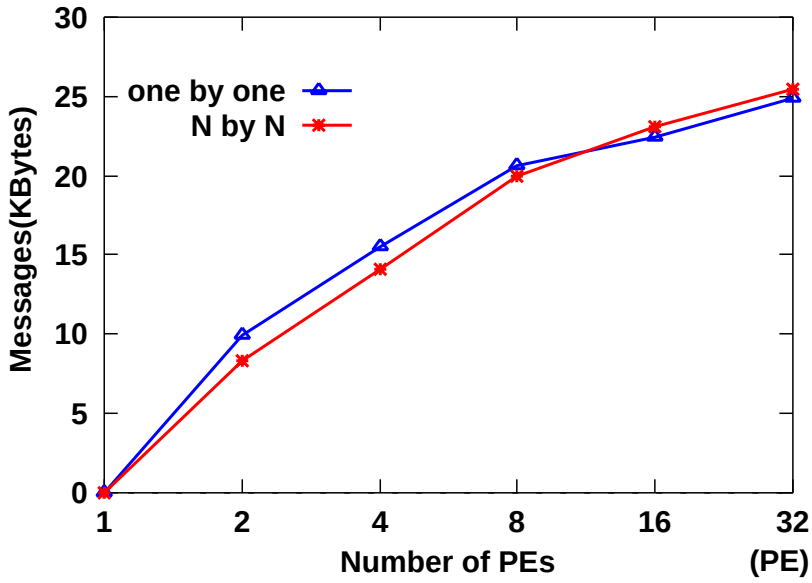
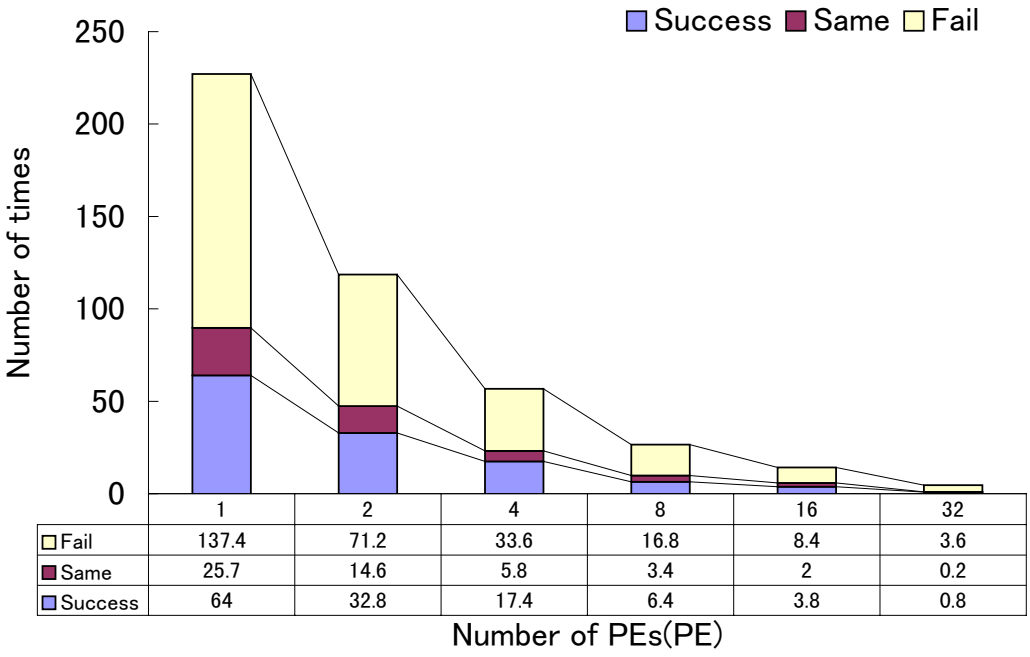
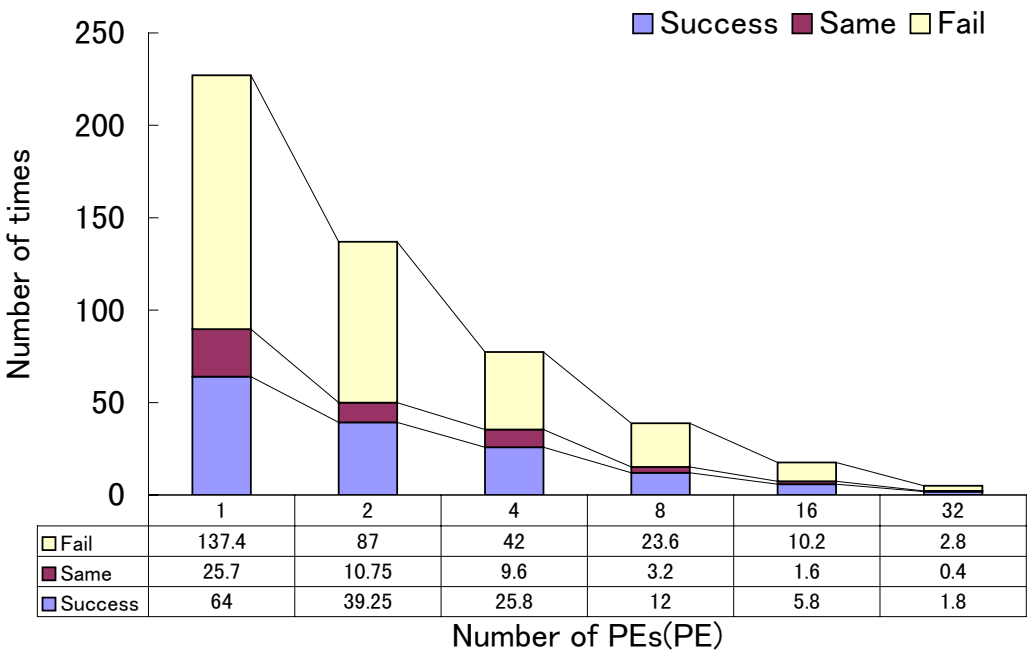


図 4.18: マスタースレーブ型における総メッセージ通信データ量



	1	2	4	8	16	32
Fail	137.4	71.2	33.6	16.8	8.4	3.6
Same	25.7	14.6	5.8	3.4	2	0.2
Success	64	32.8	17.4	6.4	3.8	0.8

図 4.19: マスタースレーブ型における TSP の実行回数 (one by one)



	1	2	4	8	16	32
Fail	137.4	87	42	23.6	10.2	2.8
Same	25.7	10.75	9.6	3.2	1.6	0.4
Success	64	39.25	25.8	12	5.8	1.8

図 4.20: マスタースレーブ型における TSP の実行回数 (N by N)

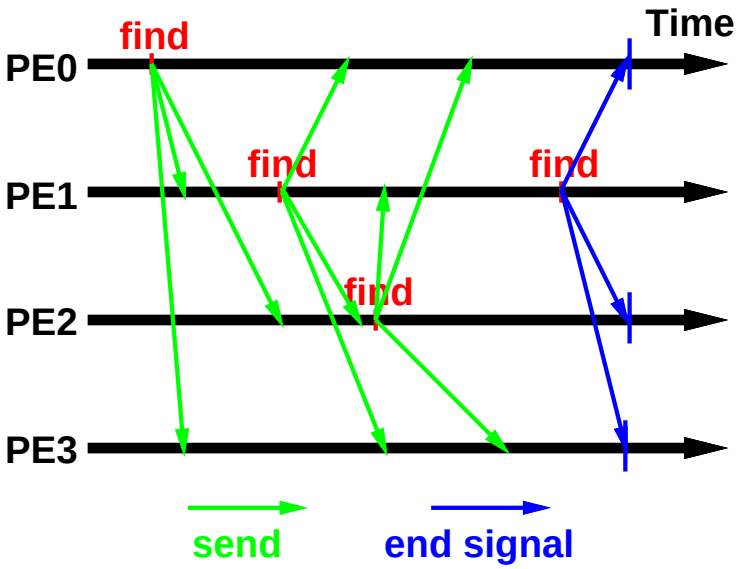


図 4.21: 協調型

次節では協調型実行を行い，その処理性能について検討する．

4.4.5 協調型による TSP の実行

協調型実行は図 4.21に示される様に、複数の PE で TSP プログラムを同時に実行し，見つけた解を順次全ての PE に送信し，最も最初に規定数の解を見つけた時点で終了とするものである．

発見した解の送信には pvm_bcast() を用いた．

この方法でも，各PE が発見すべき解の数はマスタースレーブ型と同様 $N = Nanswer/Nproc$ となる．

協調型での実行結果

図 4.22〜図 4.27に協調型での実行結果を示す．

図 4.23を見ると、マスタースレーブ型以上に実行性能の向上が見られる．32PE での *onebyone* における速度向上はほぼ 32 倍で投入台数と同じ速度向上が得られたている。また他の PE でも同様である。協調型を *onebyone* で実行するとこの様な結果を得られることが分かった。

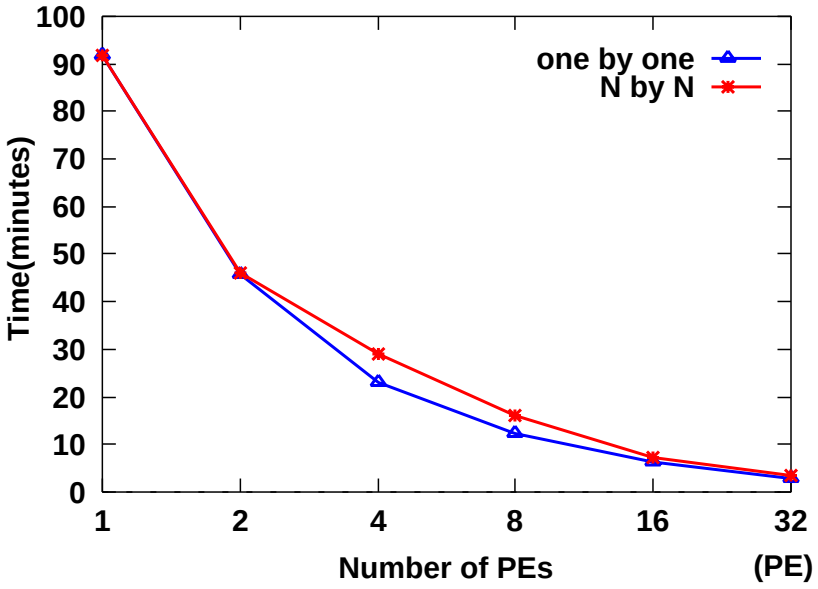


図 4.22: 協調型における TSP の実行時間

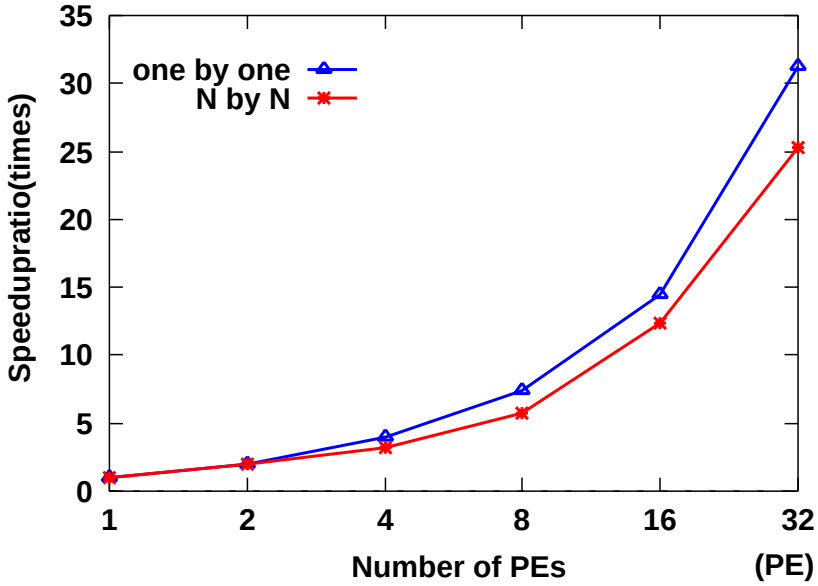


図 4.23: 協調型における速度向上比

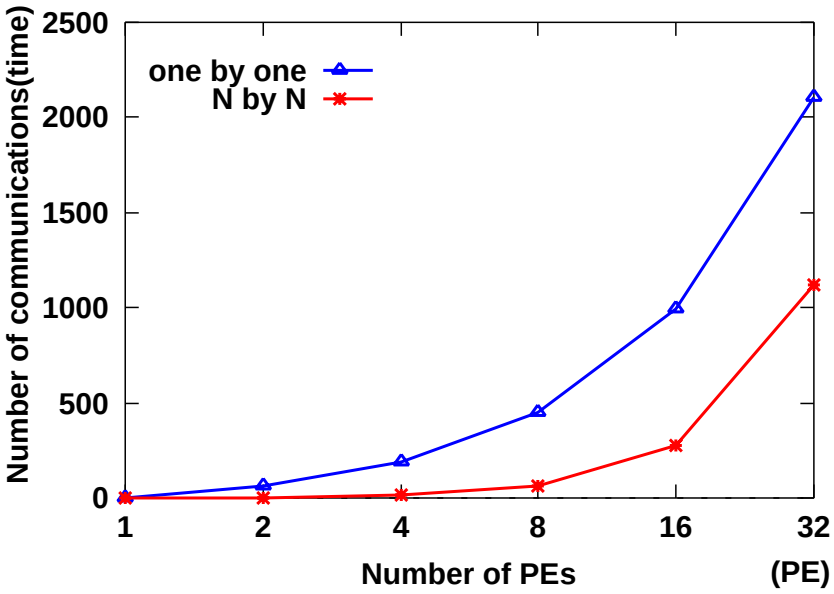


図 4.24: 協調型における総メッセージ通信回数

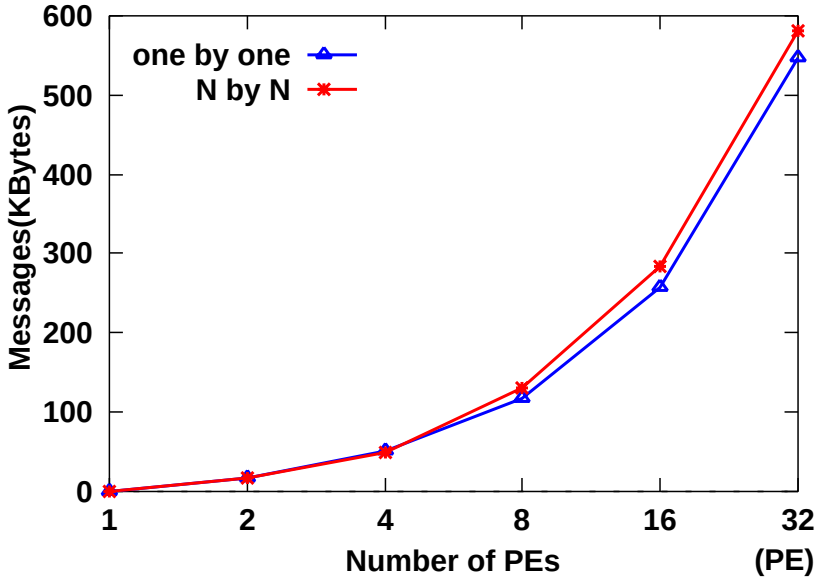
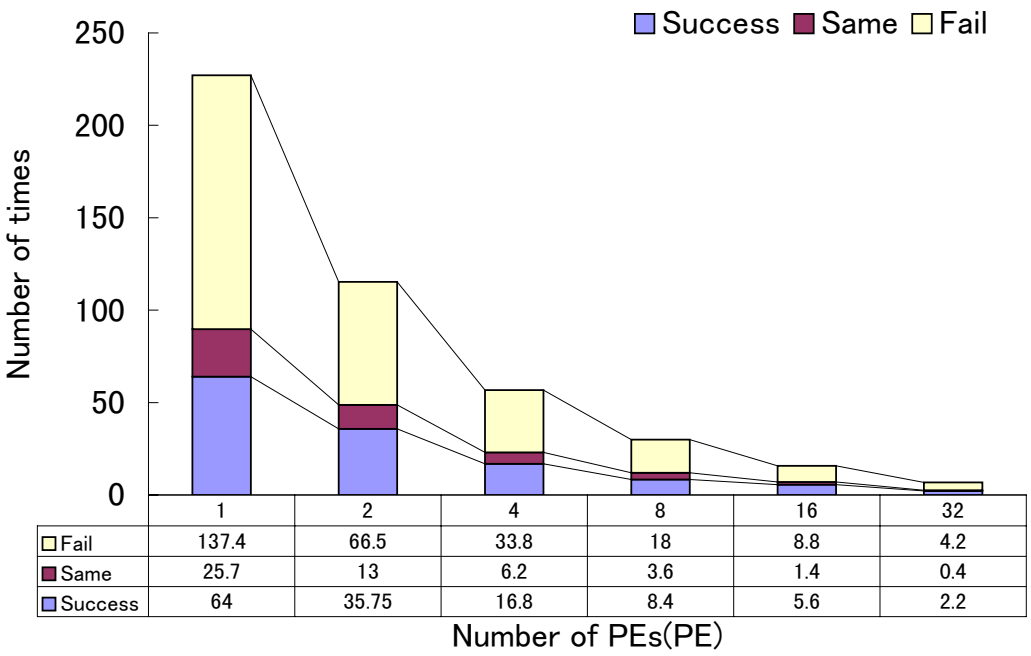
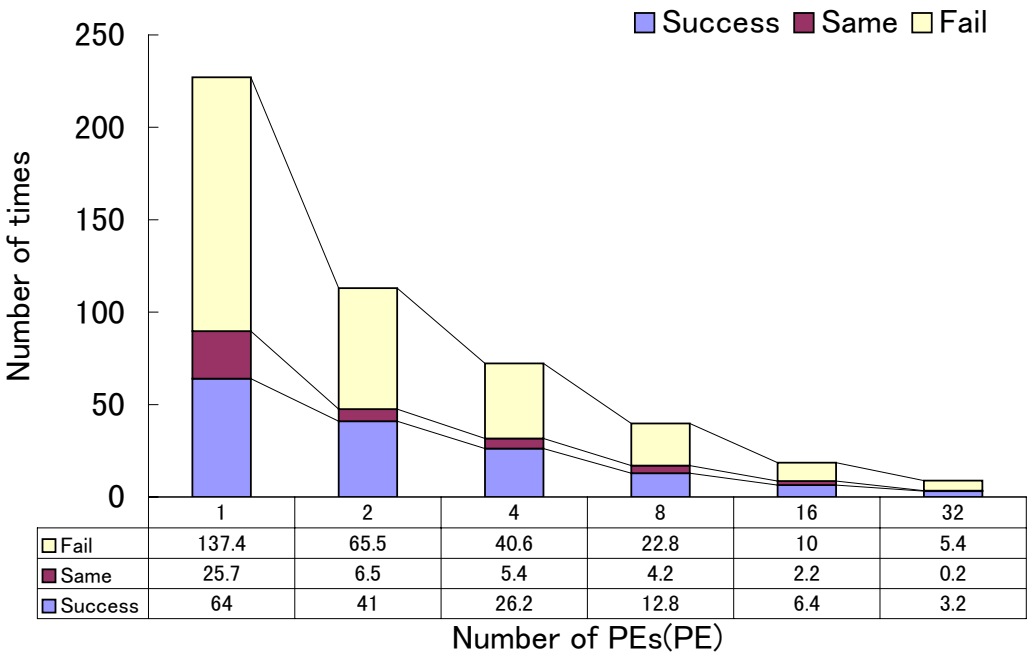


図 4.25: 協調型における総メッセージ通信データ量



	1	2	4	8	16	32
Fail	137.4	66.5	33.8	18	8.8	4.2
Same	25.7	13	6.2	3.6	1.4	0.4
Success	64	35.75	16.8	8.4	5.6	2.2

図 4.26: 協調型における TSP の実行回数 (one by one)



	1	2	4	8	16	32
Fail	137.4	65.5	40.6	22.8	10	5.4
Same	25.7	6.5	5.4	4.2	2.2	0.2
Success	64	41	26.2	12.8	6.4	3.2

図 4.27: 協調型における TSP の実行回数 (N by N)

4.4.6 各実行方法の比較

本項では、今まで述べてきた各実行方法についての比較を行う．まずマスタースレーブ型の実行と協調型の実行について比較を行い、

マスタースレーブ型と協調型の比較

TSP の処理速度を向上させるため，並列処理としてマスタースレーブ型と協調型の 2 種類の方法を実行した．ここではそれぞれの方法の比較を行う．また，解の送信ポリシについての検討を行う．

マスタースレーブ型と協調型で大きく事なる点は，PE 間の通信である．マスタースレーブ型はマスターとスレーブの 1 対 1 通信であるのに対し，協調型ではブロードキャストでデータを全ての PE へ送信するため，図 4.30や図 4.4.6に示されるように $(Nproc - 1)$ 倍の通信が処理全体で発生することになる．メッセージ通信のベンチマーク測定の結果や，ソーティングの実行結果より，10Base-T のような低速なネットワークで構成されるワークステーションクラスタではメッセージ通信が処理における大きなボトルネックとなるため，この差は大きいものであると予想した．しかし実際は図 4.28,4.29 の通り，協調型の方が実行時間が短いという結果になった．この結果より，低速なネットワークであっても TSP のように小さなメッセージがやりとりされるような問題では有効な処理性能の向上を見込めることが考えられる．

解の送信方法の検討

マスタースレーブ型と協調型の実行より，1 つ解を発見する度に送信する方法が N 個ずつ送る方よりも実行速度が優れていることが分かった．

しかし，これは TSP の実行条件が 64 種類の巡回路を見つけるととしていることに起因しており，32PE でも 1 回目の送信で 64 個の解が見つからなければ再び計算を行い，2 つの解を探さなければならないためである．

これより，解の送信個数を $Ncity/Nproc$ 個からはじめ，送信するたびに値を減らして行く方法がマスタースレーブ型の実行や少数の PE による実行時に有効であると考えられる．例えば，2PE での TSP などでは送った解の中に同じ解があった場合は 32 個の解を発見しなければならないところが半分の 16 個で送信できるようになるのでよりはやく終了することが出来るようになる．

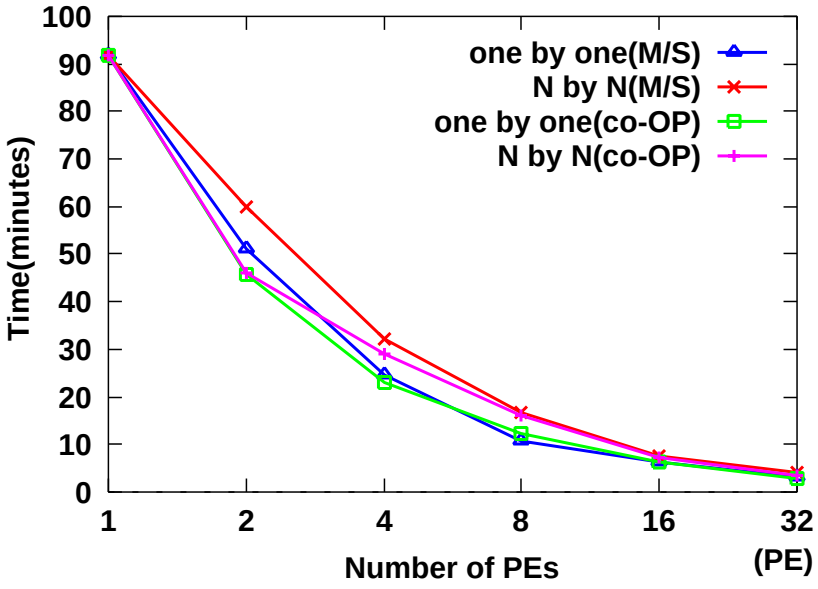


図 4.28: マスタースレーブ型と協調型の実行時間

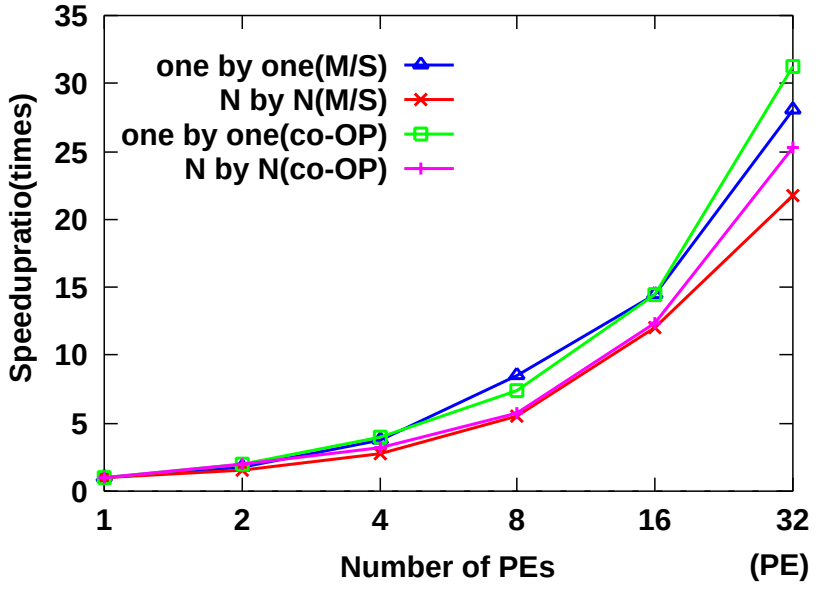


図 4.29: マスタースレーブ型と協調型の速度向上比

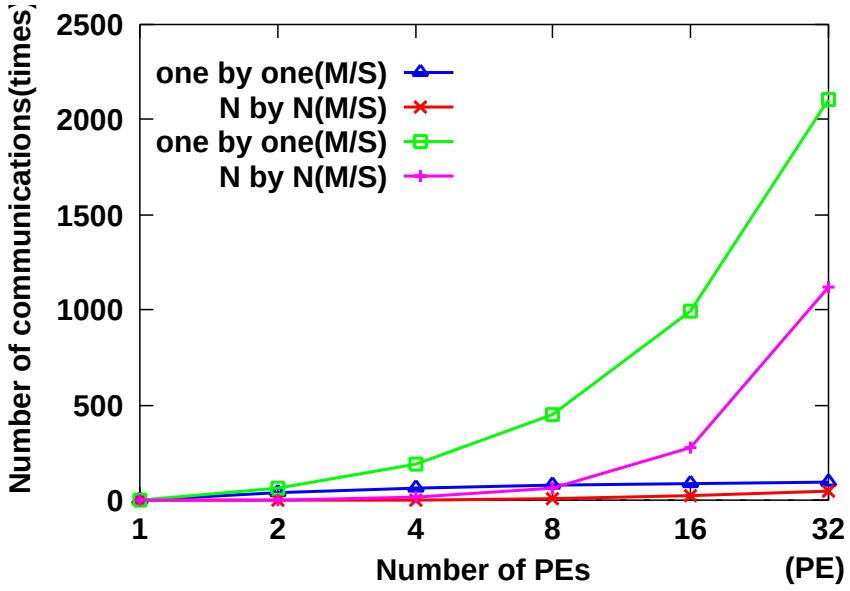


図 4.30: マススレーブ型と協調型の総メッセージ通信回数

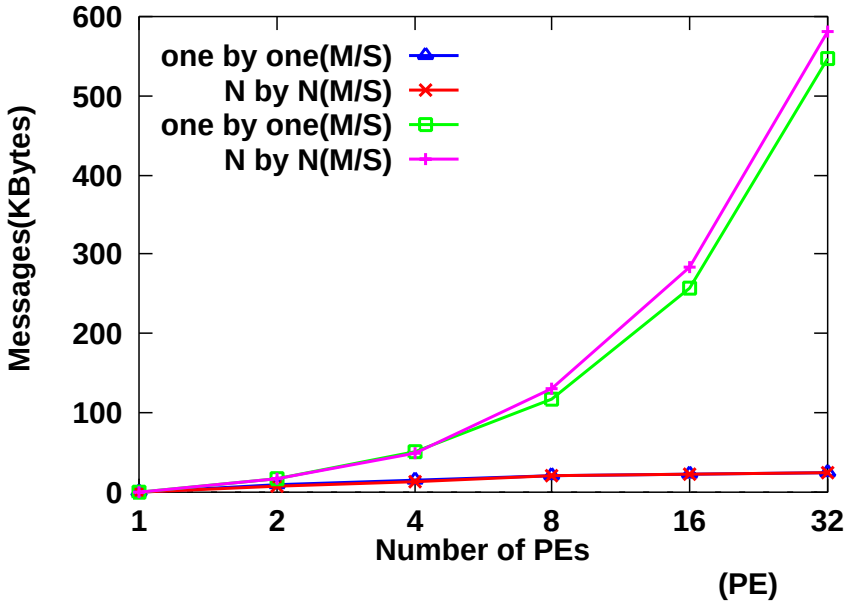


図 4.31: マススレーブ型と協調型の総メッセージ通信量

この方法を更に拡張して，マスタースレーブ型の実行ではマスターからスレーブに残りの解を元に，送信する解の単位を変更し効率良く解の送信を行わせ，処理速度の向上を計ることが考えられる．

4.4.7 PVM による変数共有システム

ワークステーションクラスタなどのような，分散メモリ型の並列計算機ではメッセージのやりとりに関する正確な指定が欠かせず，プログラムインタフェースが難しいという問題点がある．TSP の場合のように解を共有するために adsmith[7] のようなのもあるが，PVM 本体だけでなく専用のライブラリを入手する必要があり，TSP のように解を共有するだけのためにこのようなシステムを導入することはコストが大きい．

そこで PVM を利用して変数を共有化させ，発見した最短巡回路の保存に使用する変数共有システムを提案する．

共有システムは PVM のメッセージ通信ライブラリを利用する．共有させる変数はデータの共有のために以下のような機能を備える．

- 自分が更新をおこなうときはすでに到着しているデータを破棄する．
- 受け取ったデータは古いものから新しいものへと順次更新されて行く
- 更新は共有している変数を引数とする関数の実行とする．

以下にこのシステムで考えられる利点と欠点を挙げる．

利点

- 標準の PVM が使用できれば特別なソフトウェアを必要としない
- メッセージ通信のためのプログラムインタフェースが容易になる．

  共有システムを用いる変数は全ての PE へ送信/受信が可能なので，1 対 1 通信やブロードキャストのためのプログラムインタフェースを学ぶ必要がない．

欠点

- 相手の PE にデータが到着してもすぐに受け取られるわけではない.

  Active Message[5] ではメッセージの到着と同時にデータを受信することが出来るが, 本手法では実現できない. 適当な所でユーザーの手による更新データの受信を行うことが必要である.

- 更新するデータは無条件に受け入れてしまう.

  更新と称して送信された解のデータが所持している解よりも少ない解のデータの場合でも受信してしまい、処理時間がかえって増大してしまうという可能性が考えられる.

  これを回避するためには変数共有システムに TSP の解の保存に関する処理を組み込み，解の少ないデータは更新しない，などの機能を特別に付加しなければならない.

- メッセージ通信回数が多くなる

  全ての PE で解を共有するため 1 回の解の更新で $(Nproc-1)$ 回の通信が必ず発生してしまう. しかし，協調型の実行結果より，この TSP ではメッセージ通信に関する処理時間の増大が殆んど確認されなかったので，この問題は回避できると考えられる.

これらの仕様を実装し，その性能の測定と評価を行う。

4.5 まとめ

本章ではソーティング，N クィーン問題，TSP を実行しワークステーションクラスタの処理性能について検討した. ソーティングの結果より，多量のデータを通信するようなアプリケーションは 10Base-T のような低速なネットワークでは投入台数に見合う実行性能の向上が殆んど得られないことが分かった.

N クィーン問題の結果より，メッセージ通信を伴わない処理で，計算負荷の分散が均等にできれば投入台数に見合う線形的な処理速度の向上を得られることが分かった.

また，マスタースレーブ型と協調型との比較より TSP で行われるメッセージ通信は最大で協調型の $NbyN$TSP の実行でわずか 600KB 程度であるため，3 章において議論され

たメッセージ通信の測定結果と検討しても処理時間に殆んど影響を与えず，かなり高い処理速度の向上が確認することが出来た．

これよりPVMに対して変数を共有するシステムを作り，プログラミングインタフェースを殆んど変えずに，ワークステーションクラスタの処理性能に殆んど影響を与えないシステムを提供することが出来ると考えられる．

第 5 章

まとめ

ワークステーションクラスタは比較的容易に並列処理の環境を得ることが出来るシステムである．しかし、ワークステーションを接続するネットワークに高速なものを用いなければ殆んどのアプリケーションはメッセージ通信に大きな時間がかかり十分な性能が得られない．

2章ではワークステーションクラスタに対して現在取り組まれている高速化の手法についてハードゥエア、ソフトゥエアの 2 面から検討を行った．ハードウェアはネットワークを高速化し通信速度そのものを高速化する手法が盛んに行われ、現在では Myrinet の様にギガビットレベルの通信速度を発揮するものも登場している．ソフトゥエアでは既存の PVM や MPI などのライブラリを拡張し、メッセージ通信のための手続きに要する時間を削減したり、新しいインタフェースを提供するものがある．しかし Myrinet などの高速ネットワークや PVM や MPI に対する拡張は入手がまだ困難であったり、プログラミングインタフェースの変更により過去のソフトウェアが使用出来なくなるといった問題があることが分かった．

3章では，10Base-T スイッチハブを用いたワークステーションクラスタを構築し，PVM と MPI のメッセージ通信性能ついて評価を行った．その結果，PVM のメッセージ通信機能は TCP を用いるとよい性能が出るということ確認出来た．PVM と MPI では、MPI の方が通信性能が良いという結果が得られた．通信性能の評価より、PVM のブロードキャストにおける通信速度の向上を図る手法として，階層型ブロードキャストを提案した．その結果、8PE において 256KB のブロードキャストを行う場合と、4PE において 1MB のブロードキャストを行う場合について階層型ブロードキャストの性能が PVM のブロード

キャストよりも良い結果を得ることが出来た．また、送信するメッセージの大きさによってブロードキャストにPVMのブロードキャストを用いたり階層型ブロードキャストを用いたりする選択型ブロードキャストについての提案を行った．

4章では，構築したワークステーションクラスタにおける並列処理の性能を検討するため，ソーティング，N クィーン問題，TSP を実行し，処理性能の検討を行った．その結果ソーティングでは送信されるメッセージの量が大きく、メッセージ通信に関する時間が全体の処理時間の多くを占め、処理性能があまり向上しないことが分かった．この結果から、10Base-T をネットワークに利用したワークステーションクラスタでは多量のメッセージがやりとりされる問題には向いていないと考えられる．N クィーンの実行によって、メッセージ通信を伴わない処理は負荷の分散が均等に行われればワークステーションクラスタでも投入台数に見合うだけの性能向上を得られることが分かった．TSP の実行では、複数のワークステーション間でメッセージ通信が行われても、小さなサイズのデータであればワークステーションクラスタの処理時間の大きな増加には繋がらないことが分かった．これらの結果より 10Base-T のような低速なネットワークを用いたワークステーションクラスタではメッセージ通信を殆んど行わないか、データサイズの小さなメッセージ通信を行うアプリケーションであれば処理性能の低下が起きないと考えられる．

これらの結果を利用して、PVM を用いた変数共有システムについて提案し、その機能についての検討を行った．既存のインタフェースをそのまま利用可能で、プログラムインタフェースを簡単にし、処理性能には殆んど影響が出ないと考えられる．

今後の課題としては提案した階層型ブロードキャストの改善や，PVM を利用した変数共有システムの実装とその性能の評価などが挙げられる．

謝辞

本研究を進めるにあたり，終始熱心な御指導，御鞭撻をいただいた北陸先端科学技術大学院大学 堀口 進 教授，阿部 亨 助教授に心から感謝いたします.

また，様々な面で御教授頂きました，日比野 靖 教授，横田 治夫 助教授に深く感謝致します.

サブテーマで熱心に御指導をいただいた 中島 達夫 助教授に感謝いたします.

日頃より有意義な御教示，検討，議論をしてくださった山森 一人 助手に感謝いたします.

ワークステーションクラスタの構築に関してアドバイスを頂いた井口 寧 助手に感謝致します.

研究に関して有意義な助言，議論をして下さった林 亮子さん，星芝 貴之さん，加藤 聡さん に感謝致します.

また，日頃よりお世話になった堀口・阿部研究室の皆様に厚く御礼申し上げます.

参考文献

[1] 手塚宏史、堀敦史、石川裕、曽田哲之、原田浩、古田敦、山田努 “PC とギガビット LAN による PC クラスタの構築”, 情報処理学会計算機アーキテクチャ研究会報告 ARC119-7,Aug.1996.

[2] V.S.Sunderam.PVM:A Framework for Parallel Distributed Computing.*Concurrency: Practice and Experience*,2(4):315-339,1990.

[3] G.A.Geist,A.Beguelin,J.J.Dongarra,W.Jiang,R. Manchek and V.S.Sunderam. PVM: A Users'Guide and Tutorial for NetworkedParallel Computing. MIT Press,1994

[4] http://www.epm.ornl.gov/pvm/

[5] T.von Eicken et al. Active Messages: a Mechanism for Integrated Communication and Computation. In Proceedings of the 19th International Symposium on Computer Architecture, March 1992.

[6] G.A.Geist,J.A.Kohl,P.M.Papadopoulous. PVM and MPI: a Comparison for Features. Calculateurs Parallels Vol. 8 No. 2 (1996)

[7] Wen-Yew LIANG, Chung-Ta KING, Peipei LAI. Adsmith: An Object-Based Distributed Shared Memory System for Networks of Workstations. IEICE VOL.E80-D, NO.9, pp.899-908, Sep 1997

[8] Honbo Zhou and Al Geist. LPVM: A Step Towards Multi threaded PVM. Technical report, Oak Ridge National Laboratory, Oak Ridge, TN, 1995.

[9] A.J.Ferrari and V.S. Sunderam. TPVM: A Threads-Based Interface and Subsystem for PVM. Technical Report CSTR-940802, Emory University Department of Mathematics and Computer Science, Atlanta, GA, August 1994.

[10] MPI Forum, "MPI: A Message-Passing Interface Standard", *International Journal of Supercomputer Application Vol. 8 No. 3/4*, 1994

[11] http://www.mcs.anl.gov/Projects/mpi/mpich/index.html

[12] Y.Tanaka,K.Kubota,M.Matsuda,M.Sato and S.Sekiguchi "A Comparison of Data-Parallel Collective Communication Performance and Its Application" IEEE Computer Society Press(1997)

[13] http://www.cs.utk.edu/~mucci/mpbench/mpbench.html

[14] 弘中哲夫, "大規模ワークステーション・クラスタにおける PVM の性能評価", 情報処理学会ハイパフォーマンスコンピューティング研究会報告 HPC55-12,Mar.1995.

[15] N.J.Bode, D.Cohen, R.E.Felderman, A.E.Kulawik, C.L.Seitz, J.N.Seizovic and Wenking Su. "Myrinet — A Gigabit-per-second Local-Area Network". IEEE MICRO, Vol. 15, No.1, pp29-36,February 1995

[16] Scott Pakin Vijay Karamcheti Andrew A. Chien. "Fast Messages(FM):Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors)

研究業績

[1] ワークステーションクラスタにおけるメッセージ通信性能の評価
奥野弘之，堀口進
情報処理学会ハイパフォーマンスコンピューティング研究会報告 HPC68-4 pp.21-26, Oct.1997.

[2] ワークステーションクラスタにおけるメッセージ通信性能の評価
奥野弘之，阿部亨，堀口進
電気関係学会北陸支部連合大会 pp.279, Nov.1997.