# UDPを利用したＤＳＥ向けプロトコルの設計と実装

手塚忠則[†,††] 末吉敏則[†††] 有田五次郎[††]

我々は，LAN接続されたワークステーション（WS）を利用して並列処理を行う共有メモリ型並列処理環境DSE(Distributed Supercompting Environment)の研究を行っている．これまでのDSEでは，通信手段としてTCPを用いていた．しかし，TCPでは，各WS上で動作するDSEの仮想プロセッサ同士を１対１で接続するため，仮想プロセッサ数の増加に伴いOSの資源であるポートを大量に消費してしまうという問題があった．そこで，本研究では，通信プロトコルとしてUDPを用いて利用ポート数の削減を図った．本稿では，UDPを用いて実装したプロトコル機能と，DSEに実装して行った評価実験の結果について述べる．実験の結果，実装したプロトコルでは，少ないポート数で信頼性のある通信を，TCPと同等以上のパフォーマンスで実現できることが確認できた．

# Implementation and Evaluation of a UDP-based Communication Protocol for DSE

TADANORI TEZUKA [†,††] TOSHINORI SUEYOSHI [†††]
and ITSUJIROU ARITA [††]

We have conducted studies of a distributed shared memory-based parallel and distributed computing environment on a cluster of workstations, called DSE. Current DSE utilized TCP to communicate between virtual processors. TCP needs a socket port for each connection. Due to the limitation of OS resource, it restricts the number of virtual processors that can be connected. So, we proposed a UDP-based reliable communication protocol for DSE. In this paper, we describe the detail of our protocol and show experimental results of this.

## 1. はじめに

複数の計算機が高速ネットワークで接続された分散環境は，企業や大学を含め多くの場所で構築され利用されている．近年，このような分散環境を利用して並列処理を行うクラスタコンピューティングが注目されており，MPI (Message Passing Interface)[1] やPVM (Parallel Virtual Machine)[2] といったクラスタコンピューティング環境が研究開発され，多くの大学などで利用されている．

† 松下電器産業株式会社 九州マルチメディアシステム研究所 KYUSHU MULTIMEDIA SYSTEMS RESEARCH LABORATORY, MATSUSHITA ELECTRIC INDUSTRIAL CO., LTD.
☆ 現在，
Presently with
†† 九州工業大学 情報工学部 知能情報工学科 DEPARTMENT OF ARTIFICAL INTELLIGENCE, KYUSHU INSTITUTE OF TECHNOLOGY
☆☆ 現在，
Presently with
††† 熊本大学 工学部 数理情報システム工学科 DEPARTMENT OF COMPUTER SCIENCE, KUMAMOTO UNIVERSITY

このような背景のもと，我々は，ローカルエリアネットワーク（ＬＡＮ）接続された複数のワークステーション（ＷＳ）を分散共有メモリ型の並列計算機とみなして利用するための基本ソフトウェアDSE (Distributed Suptercomputing Environment) に関する研究を行っている[3][4][5]．DSEは，仮想的なプロセッサ要素（UNIXプロセス）間の通信をTCPを用いた１対１の通信により実現しており，デーモンなどを介してWS間の通信を行う他のクラスタコンピューティング環境に比べて高速な通信を実現していることが特徴である．

しかし，TCPを用いた通信では，１つの仮想プロセッサ間の通信に対して１つのポートを消費してしまう．このため，多数の仮想プロセッサを利用しようとすると多くのシステムリソース（ポート）を消費してしまうという問題がある．例えば，仮想プロセッサの数を60とした場合，DSE全体で3,540ポートが必要となる．また，多くのオペレーティングシステムでは，１つのプロセスが利用できるポート数に制限があるため，制限以上の仮想プロセッサの接続ができないという問題も生じている．

現在のDSEでは，この問題を回避するために，LAN上にリング網やトーラス網などの仮想的なネットワークを構築し，直接接続されていない仮想プロセッサ間の通信は，他の仮想プロセッサを経由して行うという方法をとっている．しかし，この方法では，仮想プロセッサを経由した通信が発生するために，直接接続された場合に比べて通信速度が低下してしまうという問題が生じる．

そこで，本研究では，これまでのTCPに代わり，通信プロトコルとしてUDPを用いて通信を行うことを試みた．UDPは，コネクションレス型のプロトコルで，TCPとは異なり1つのポートで複数の計算機との通信が可能である．しかし，UDPでは，データの到着が保証されていないため，TCPのような信頼性を備えていない．これまでに行った予備実験の結果から，UDPはTCPに比べて双方向データ通信の場合で約5%～16%程度高速ではあるが，片方向への連続転送では半分以上のデータがバッファオーバフローにより喪失してしまうことが分かっている[6]．

これらを踏まえ，我々はUDPとTCPの処理速度の差異の範囲内の処理量で信頼性を実現するプロトコルを設計し，ユーザレベルプロセスとして実装した．具体的には，メッセージの分割・組み立て／エラーリカバリ／フロー制御を備えたプロトコルをUDPの上位に実現した．実装したプロトコルでは，送信・受信のために各1ポートしか必要としない．このため，仮想プロセッサ数が60の場合には，全体で120ポートしか消費しない．このように，60台の場合で消費ポート数を約1/28に押さえながらも，各仮想プロセッサ間の直接通信を可能にしている．

本稿では，新たに実装したプロトコルの概要および，プロトコル単体での予備実験および，プロトコルをDSEに組み込んで行った実験の結果について報告し評価結果について述べる．

## 2. プロトコル概要

### 2.1 プロトコルヘッダ

実装したプロトコルのヘッダを**図1**に示す．ヘッダ中の**Src CPU No.**および**Des CPU No.**は，受信したパケットの送信元および受信先の仮想プロセッサ番号である．仮想プロセッサ番号とは，DSEのプロセッサ要素となる計算機上のプロセスにつけられるDSE全体で一意な番号であり，これを用いることで，どのプロセッサ要素からのメッセージであるのかが一意に識別できる．本プロトコルでは，これを利用して各仮想プロセッサから送信されたパケットを識別および振り分けを行う．また，**Src Seq No.**および**Des Seq No.**は，各仮想プロセッサ間の通信ごとに割り当てられるシーケンス番号である．

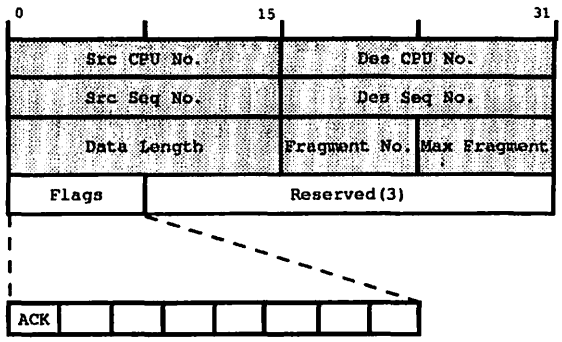


**図1** ヘッダフォーマット
Fig. 1 A Headder Format

**Data Length**と**Fragment No.**および**Max Fragment**は，DSEの通信単位であるメッセージをパケットに分割したり，パケットからメッセージを組み立てるために利用される情報である．また，**Flags**には，通信制御のためのいくつかのフラグビットが格納されており，現在のところ**ACK**パケットかどうかを識別するためのビットのみが割り振られている．

### 2.2 メッセージ分割・組み立て処理

本プロトコルでは，同一ポートで受信した複数の仮想プロセッサからのパケットを，個別のDESのメッセージに組み立てなおすために，**図2**に示すようなメッセージの分割・組み立て処理を行っている．図2中の N-m/n という表記は，Nがパケット番号を，nがメッセージの分割数を，mが分割されたメッセージの何番目であるかを示している．例えば，$a-1/2$という表記は，パケット番号がaで，分割数が2，そしてこのパケットが2つに分割されたパケットの始めの(1番目の)パケットであることを示している．以下，この受信処理の流れを例に，メッセージの組み立て処理について簡単に説明する．

送信（1）では0～3番のパケットを，送信（2）ではa～dのパケットをそれぞれ順番に送信し，受信側では，a,0,1,b,c,2,3,dの順で受信している．パケットaを受信した場合，このパケットは2つのパケットに分割されたメッセージの1つめであるので，送信（2）に対応する一時バッファ(Buf2)に蓄積しておき，パケットbが受信された時点で受信バッファへ格納する．次に受信したパケット0は分割されていないメッセージであるので，そのまま受信バッファに格納する．また，次に受け取ったパケット1は2つに分割されたメッセージの1つめのパケットであるため，送信（1）

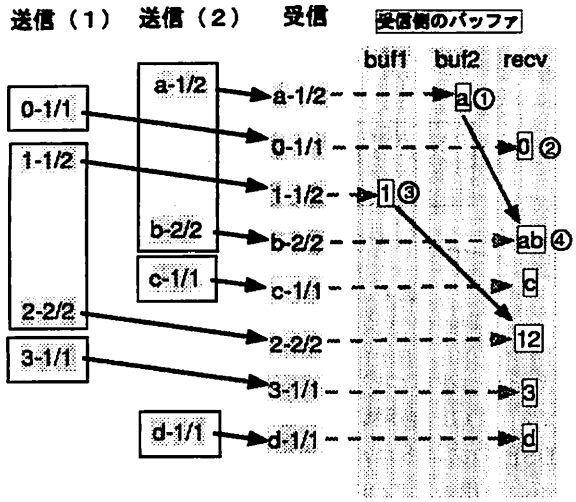


図 2 メッセージの分割・組み立て（例）
Fig. 2 A fragmentation and defragmentation of messages (Example).

のために一時バッファ(Buf1) に格納され，パケット 2 が受信されるまで一時的に保存される．

以上のように，メッセージが複数のパケットに分割される場合は，送信元別の一時バッファに格納しておき，最後のパケットを受信した時点でメッセージへの組み立てを行い，受信バッファに格納する．また，メッセージが分割されていない場合には，受信バッファには一時バッファを通さずにそのまま受信バッファへ格納する．

このように，パケットサイズより小さいメッセージは直接受信バッファに格納するようにしたことで，小さなメッセージの受信時のコピー処理を削減し，高速化を図っている．表 1 は，DSE 上で，巡回騎士問題，オセロ，行列積，偏微分方程式 (SOR)，画像に対する離散コサイン変換 (DCT) の 5 つのアプリケーションを実行した時の実行ログからメッセージのサイズとその割合を計算したものである．表 1 を見ると，32 バイト以下のメッセージが全体の 80%,128 バイトまでのメッセージであれば全体の 98%を占めることがわかる．したがって，UDP を下位プロトコルとして利用する本プロトコルの場合には DSE のメッセージの 98%以上が 1 つのパケットとして送信され，受信される．このため，一時バッファに格納されるメッセージの数も全体の 2%以下となり，ほとんどのメッセージが一時バッファを通さずに受信バッファに格納されるため，小さいメッセージを直接受信バッファにコピーする速度的効果が期待できる．

### 2.3 エラー処理および再送処理

本プロトコルでは，エラー処理とエラー時の再送処理については，LAN をターゲットとした場合の必要最低限の機能だけを盛り込むことにした．このようにターゲットを LAN に限定することで，TCP に比べて簡単な処理でエラー処理を行うことが可能となった．具体的には，タイマ処理による再送，および NACK による再送要求処理の送信という 2 つの処理によりエラー処理および再送処理を実現している．なお，エラー発生時の再送は，エラー発生の少ない LAN 環境を前提としているので，処理コストの小さな Go-Back-N プロトコルを用いた．

表 1 アプリケーション実行時のメッセージサイズの割合
Table 1 The percentage of message size in the DSE applications

| メッセージサイズ | 割合 (%) |
|---|---|
| 1 ～ 16 バイト | 60% |
| 17 ～ 32 バイト | 20% |
| 33 ～ 128 バイト | 18% |
| 129 ～ 1024 バイト | 0.3% |
| 1025 バイト以上 | 1.7% |

### 2.4 フロー制御

先に述べたように，これまでの予備実験 [6] より，UDP を用いた連続データ転送では，パケット消失が頻繁に発生することが分かっている．本プロトコルでは，前述した再送処理によりエラー発生時にも問題なくデータの転送を行うことができるが，これだけでは，頻繁にエラーが発生する場合に再送処理のオーバヘッドが非常に大きくなってしまう．そこで，送信側の連続転送パケット数を制限し，ACK が戻ってから続きを送信するという簡単なフロー制御を組み込み，バースト的なデータの転送時のエラー発生を抑制した．また，前述の実行ログの結果から，送信と送信に対する返信がペアになったメッセージが多いことも分かっているので，ACK を返信パケットにパッキングすることにより，この制御に伴う ACK のみのパケットの数の削減した．

なお，今回の実装では，連続パケット送信数は 10 パケット，最大パケットサイズは 1024 バイトとした．

## 3. DSE への組み込み

### 3.1 DSE 概要

DSE は，分散共有メモリ型並列計算機の機能を LAN 接続された WS クラスタ上で実現するためのソフトウェアであり，UNIX のユーザプロセスとして実装されている．DSE では，UNIX プロセスが 1 つの仮想的なプロセッサ要素に対応し，この仮想的なプロセッサ要素上でアプリケーションを実行する．UNIX プロセスが仮想プロセッサに対応するため，1 つの WS 上

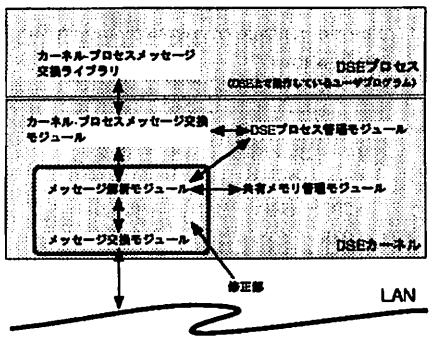


図 3 DSE の機能ブロック
Fig. 3 A block diagram of DSE

に複数の仮想プロセッサをマッピングすることも可能であり，少ない WS 数でも多くのプロセッサ要素を持つ並列計算機の環境が実現できる．

**図 3** に DSE のソフトウェアの機能ブロックを示す．図 3 のように，DSE は大きく分けて 2 つのブロックから構成されている．1 つは，DSE プロセスで，これは，ユーザの記述したプログラムを実行する部分である．もう 1 つは，DSE カーネルで，これはユーザプログラムからライブラリを通して送られた要求を解析し，必要であれば他の WS 上で実行されている DSE カーネルと通信を行って処理を行う部分である．この DSE カーネルでは，DSE プロセスからの要求に対する処理だけなく，仮想的な分散共有メモリの管理なども行っている．

また，DSE カーネルは，カーネル-プロセスメッセージ交換モジュール，DSE プロセス管理モジュール，共有メモリ管理モジュール，メッセージ解析モジュール，メッセージ交換モジュールの 5 つの機能モジュールから構成される．今回のプロトコルの実装にあたっては，この 5 つのモジュールのうち，メッセージ解析モジュールとメッセージ交換モジュールの 2 つを修正した．

### 3.2 プロトコルの組み込み

**図 4** は，修正が必要となる 2 つのモジュールの機能と関係を詳しく示したものである．新たに実装したプロトコルは図 3 および図 4 のメッセージ交換モジュールに相当する部分を行うもので，今回は，このメッセージ交換モジュールを新規のプロトコルのものと入れ替えることで実装を行った．また，この入れ替えで，メッセージ交換モジュールとのデータのやり取りを行っていたメッセージ解析モジュール中のメッセージ生成部とメッセージ解析・実行部の修正が必要となったので，この 2 つの部分も修正した．なお，DSE は機能ブロック毎に入れ替えが可能な構造であるため，DSE の他

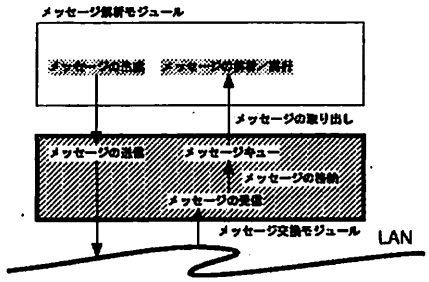


図 4 モジュールの機能詳細
Fig. 4 A detail of module's functions

のモジュールについて修正を行う必要はなかった．

### 3.3 TCP を用いた場合との比較

DSE のプロトコルを新規に実装したプロトコルに変更した場合，TCP に比べてメッセージ受信に対して UNIX カーネルを呼び出す回数が少なくなる．

TCP の場合，ストリームを基本とする通信であるため，送受信されるメッセージのサイズが可変長であれば，送信側が送出したメッセージのサイズを受信側では知ることができない．このため，従来の DSE では，まず DSE のメッセージのヘッダを読み出してメッセージ長を調べ、メッセージ長が分かってからメッセージの本体を読み出すという UNIX カーネルに対して 2 回の Recv 処理を行う必要があった．しかし，UDP の場合は通信はパケット単位を基本とするため，受信は送信されたパケット単位で行うことができる．前述したように，DSE メッセージの多くは UDP の 1 パケットに納まるサイズであるため，ほとんどのメッセージに対して 1 回の Recv で受信を行うことが可能である．

このように，新たに実装したプロトコルを用いた場合，UNIX の Recv システムコールの発行回数を削減することができる．

## 4. 実験結果

### 4.1 予備実験

DSE への実装を評価する前に，実装したプロトコルと TCP との評価を行ったのでその結果を報告する．実験を行った環境は，SPARC station20 × 3 台，SunOS 4.1.4，Ethernet(10Base-T) という環境である．

実験では，受信 WS 対送信 WS の数を 1 対 1 または 1 対 2 に変更して，送信側から受信側へのバースト的な転送およびピンポン転送を連続して 10,000 回行った．この結果が，**図 5** および**図 6** である．これらの結果を見ると，バーストおよびピンポン転送のどちらの場合も実装したプロトコルの方が処理に時間がかかっていることがわかる．バースト転送の場合に TCP の方が高速な理由としては以下のことが考えられる．

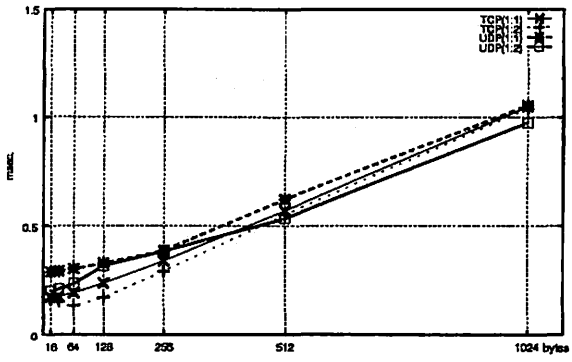


図 5 一方向への連続転送の結果
Fig. 5 One-way transmission results

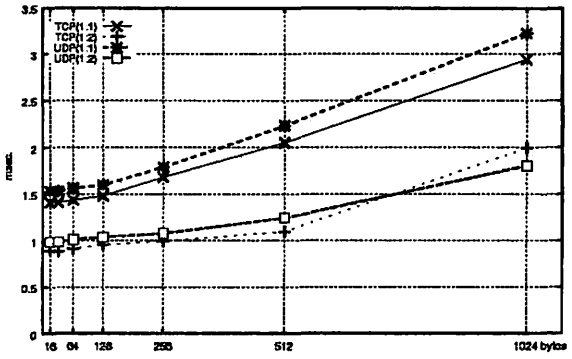


図 6 ピンポン転送の結果
Fig. 6 Two-way transmission results

TCP では，小さな連続したデータ転送を 1 つのパケットにまとめて送り転送効率を上げるという機能を組み込まれている．したがって，小さなデータのバースト転送の場合は，複数の送信要求が 1 つのデータにまとめられ送信されるため，転送効率が上がっている．実際，図 5 を見ると，小さなデータサイズの方が両者の差が大きいことがわかる．

しかし，DSE での利用を考えた場合，DSE メッセージの解析から多くのメッセージは要求 (Request) と返答 (Ack) という組で構成されていることが多く，通信パターンとしてはピンポン転送の方が多い．このため，ピンポン転送での性能が重要となる．ピンポン転送を見た場合，両者の差は 10%程度であり，DSE に組み込んだ場合は，前述した受信バッファ制御や UNIX システムコール発行数の削減により両者の差はさらに小さくなると考えられる．

### 4.2 並列アプリケーション

次に，プロトコルを組み込んだ DSE を用いて行った性能の評価について説明する．ここでは，メッセージのパターンが特徴的な 2 つのアプリケーションを利用して実験を行った．実験に利用したアプリケーションは，1 つはコンピュータ対コンピュータで対戦を行うオセロゲームであり，もう 1 つは偏微分方程式の解を求めるプログラム (SOR) である．

オセロゲームは，各仮想プロセッサからのアクセスが 1 台の仮想プロセッサに集中するという特徴があり，多くのプロセス生成操作やバリア同期操作を含むプログラムである．また，SOR は，基本的にはプロセッサ番号が隣接する仮想プロセッサ同士（例えばプロセッサ番号が N の仮想プロセッサであれば，仮想プロセッサ N-1 と N+1 と通信を行うという意味）での通信が主となるプログラムであり，メッセージの送受信は隣接する仮想プロセッサ間に局所化されている．つまり，オセロゲームは一極集中したアクセス，SOR は均等に分散されたアクセスを行うプログラムである．

この実験では，FreeBSD が動作する環境で仮想プロセッサの数を 1 ～ 8 に変化させて実験を行った．また，TCP を用いて従来の DSE についてはポート数を制限する意味で，仮想的に完全網，鎖網，木状網，リング網，トーラス網による接続を行い実験を行った．この実験の結果が**図 7** と**図 8** である．なお，グラフの縦軸は処理時間，横軸は実行回数で，今回の実験では 5 回測定を行いその結果を示している．なお，DMP が今回実装したプロトコル，Cont が完全網，chain が鎖状網，tree が木状網，ring がリング網，torus がトーラス網を示す．

図 7 の結果より，実装したプロトコルは完全網による接続には劣るものの，他の接続に比べてよい結果をえることができることがわかった．なお，接続網順に結果を並べると，完全網，トーラス網，リング網，木状網，差網となり，これは，アクセスが集中する仮想プロセッサ 0 と各仮想プロセッサとのの平均距離の順に一致している．オセロにおいて完全網に比べて実装したプロトコルが劣る理由は，実装したプロトコルでは，1 つのポートですべての仮想プロセッサからのメッセージを受け取るため，アクセス集中が起こると処理性能が低下してしまうこと，および，メッセージの集中するプロセスの UNIX 上でのプロセスの優先順位が低下してしまうことが原因である．

一方，SOR は基本的には隣接する仮想プロセッサ同士での通信しか行わないため，リング網やトーラス網などでもよいパフォーマンスを得ることができると考えられる．実際，図 8 の結果を見ると，完全網，実装したプロトコル，リング網，トーラス網，木状態網，鎖網の順で実行時間が短いことがわかる．完全網がリング網などに比べ結果が良い理由は，一部のメッセージ（途中結果表示など）はユーザコンソールのある仮想プロセッサに集中するため，一部隣接以外の通信が

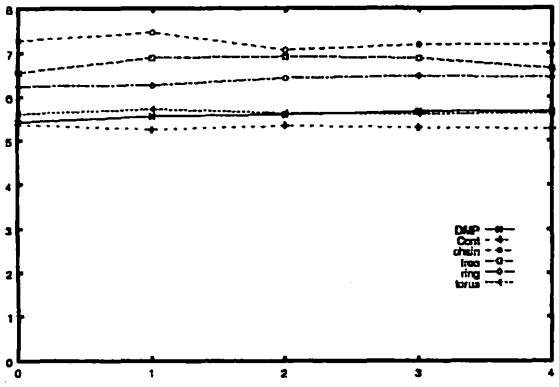

図 7 オセロゲームの実行結果
Fig. 7 The result of othello game.

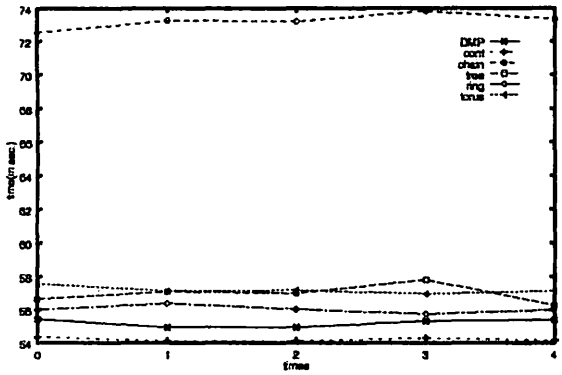

図 8 偏微分方程式 (SOR) の実行結果
Fig. 8 The result of a partial differential equation(SOR).

存在しているためである．なお，この結果でも，実装したプロトコルは完全網以外よりは高速であることがわかった．完全網に比べ実装したプロトコルが遅い理由は，ユーザプロセス上でプロトコル処理を行ったため，DSE カーネルを実行しているプロセスの優先度が低下するためである．実際に，実時間ではなく，プログラム実行中に DSE カーネルに割り当てられた CPU 時間を比較すると TCP を用いた DSE に比べて 3 ～ 5 倍程度 CPU 時間が割り当てられている．

以上の結果より，実装したプロトコルは完全網で接続した場合には劣るが，ポート数を制限した他の接続に比べて短い時間でアプリケーションの実行ができることが示された．また，TCP を用いた仮想的な接続網を利用した場合には，接続網の形状とアプリケーションのアクセスパターンの差異が実行速度に大きく影響するのに対して，本プロトコルでは，アプリケーションのアクセスパターンの影響が少ないということがわかる．

## 5. ま と め

本稿では，TCP を用いた従来の DSE における仮想プロセッサ数の増加に伴い消費するポート数が増大してしまうという問題の解決手段として，UDP を用いたプロトコルを提案し実装を行った．また，実装したプロトコルの性能の評価の結果から，DSE に実装した場合，TCP を用いた完全接続に比べてパフォーマンスの低下はあるが，ポート数を制限した場合に比べるとよい結果を得られることを示した．

以上のように，1 つの仮想プロセッサが送信と受信に対して各 1 ポートづつしか消費しないようにしたため，多数の仮想プロセッサを利用する場合のポート数を大幅に削減でき，これによりシステムのリソースの消費を押さえることができた．なお，利用するポートが少ないため，複数のユーザが同時に DSE を利用する環境下でのポートの自動割り当てが容易になると考えられる．

## 参 考 文 献


1) MPI Forum.: MPI: A message-passing interface standard., *International Jounal of Supercomputer Applications and High Performance Computing*, Vol. 8 (1994).
2) Sunderam, V. S.: PVM: A Framework for Parallel Distributed Computing, *Concurrency: Practice and Experience*, Vol. 2, No. 4, pp. 315–339 (1990).
3) T. Tezuka, K. Ryokai, B. Apduhan and T. Sueyoshi: Implementation and Evaluation of a Distributed Supercomputing Envionment on a Clustor of Workstations, *Proc. of 1992 International Conference on Parallel And Distributed System*, pp. 58–65 (1992).
4) B. Apduhan, T. Sueyoshi, T. Tezuka and I. Arita: The Effect of Communication Processing in Network Supercomputing Environment., *Proc. of 7th International Joint Workshop on Computer Communications*, pp. 373–380 (1992).
5) 手塚忠則, 了戒清, 末吉敏則: 分散システムを利用した並列処理環境における通信処理の影響, 情報処理学会「マルチメディア通信と分散処理」ワークショップ論文集, pp. 249–256 (1993).
6) 江口真, 手塚忠則, 末吉敏則, 有田五次郎: UDP を用いたクラスタコンピューティング向け通信プロトコル設計のための予備実験, 情報処理学会「マルチメディア通信と分散処理」ワークショップ論文集, pp. 203–207 (1998).