# 共有メモリ型マルチプロセッサを対象とした レイヤ単位の並列処理による通信プロトコルの実装方式

佐藤 友実[†] 加藤 聰彦[††] 鈴木 健二[††]

伝送路の高速化にともない通信プロトコルの高速処理方式が必要となっている．最近では，共有メモリ型マルチプロセッサ構成のワークステーションが広く普及しているため，このようなワークステーション上において，並列処理方式によりプロトコル処理を高性能化するための検討が重要であると考えられる．本稿では，OSが提供するスレッドを用いて1つのレイヤのプロトコル処理を個別のプロセッサに割り当てる，レイヤ単位の並列処理方式を提案する．本方式は，並列処理のオーバヘッドを少なくするために，レイヤ間のインタフェースに並行して読み書きが可能なキューを用い，また，レイヤ間で共有するプリミティブやPDU用のバッファのアクセスや，確保/解放処理において，レイヤ間の競合を減らす方法を用いている．さらに，この方式を用いて，データの受信確認やフロー制御の機能を有するコネクション型プロトコルのプログラムを作成し，HIPPIを介し性能評価を行った．その結果，並列度が11の場合に最大10.4倍の性能向上を実現できるという評価を得た．

# Implementation Method of Communication Protocols Using Processor-per-layer Parallel Processing for Shared Memory Multiprocessor Systems

TOMOMI SATO,[†] TOSHIHIKO KATO[††] and KENJI SUZUKI[††]

As the network transmission bandwidth increases, the high performance implementation of communication protocols is required. Because shared memory multiprocessor workstations are widely used, it is important to study the implementation of the high performance parallel processing of communication protocols on such workstations. This paper proposes an implementation method using processor-per-layer parallel processing, which assigns one processor to one layer protocol. Our approach introduces a mechanism to reduce parallel execution overheads among processors for the interaction between protocols executed in parallel, the access to data of protocol data units and service primitives, and the allocation and deallocation of global buffers. We have evaluated the performance of our method using an experimental parallel protocol program through HIPPI, and shown that our scheme provides an effective parallel protocol execution whose efficiency is more than 80 % when 10 processors are used.

## 1. はじめに

近年，100 Mbpsを超える超高速ネットワークの導入にともない，それらに接続されるコンピュータにおいて，高性能な通信プロトコル処理の実現が求められている．一方，最近では，処理の高速化を目的として，メモリを共有するマルチプロセッサ構成を採用したワークステーションが広く普及している．したがって，市販のワークステーション上で高速な通信プロトコル処理を行うためには，共有メモリ型マルチプロセッサを用いた通信プロトコルの並列処理方式の研究が重要であると考えられる．

このような研究は従来からも行われており[1),2)]，これらにおいては，1つのメッセージに対する複数レイヤの処理を，1つのプロセッサに割り当てるメッセージ単位の並列処理方式が有効であると考えられてきた．その理由としては，並列化にともなうオーバヘッドを考慮すると，複数レイヤのプロトコル処理が並列化の単位として適当な粒度であると判断できること，プロセッサ数に適応した負荷の配分が容易であることなどがあげられている[2)]．しかしこの方式では，コネクション管理テーブルなどを，プロセッサ間で共有する必要がある．そこで，それらの情報にアクセスする

† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications
†† 株式会社KDD研究所
KDD R&D Laboratories, Inc.

ために排他制御が必要となり，その処理オーバヘッドやアクセス可能となるまでの待ち時間などにより，プロセッサ数を増加してもスループットが向上しないという結果が報告されている[2]．

これに対し筆者ら[3),4)]は，各レイヤのプロトコル処理を並列に実行する，レイヤ単位の並列処理方式を採用している．この方式では，レイヤ構造に対応した並列化を導入することにより，プロセッサ間で共有する情報を減少させ，上述のような共有情報に対する排他制御の必要性を減少させることが可能であると考えられる．一方，1つのメッセージに対する1つのレイヤのプロトコル処理が並列化の単位であるため，メッセージ単位の並列処理方式に比べて並列化の粒度が小さくなる．したがって，並列処理のためのオーバヘッドが，より大きく性能に影響すると予想される．このため，レイヤ単位の並列処理方式を用いて高いスループットを得るためには，レイヤ間のインタフェースやバッファ管理における排他制御などの，並列処理の実現にともなうオーバヘッドを極力小さくする必要がある[3]．そこで筆者らは，オーバヘッドの小さいレイヤ単位の並列処理方式を提案するとともに[3]，提案方式を実現する並列処理用ライブラリ，および，複数レイヤを積み重ねた評価用のプロトコルプログラムを実装し，その性能評価を行った[4)~6)]．本論文では，これらの結果について報告する．

以下本論文では，筆者らが提案している並列処理方式について2章で述べ，3章において作成した並列処理用ライブラリと，それを用いた通信プログラムの実装方法について述べる．4章で，評価用のプロトコルプログラムを用いて，HIPPI（High Performance Parallel Interface）を介した通信による性能評価について示し，5章で考察を行う．

## 2. 並列処理方式

### 2.1 設計方針と概要

レイヤ単位の並列処理を実現するために，以下のような設計方針を採用した．

- 複数レイヤの通信プロトコルを実行するプロトコルプログラムにおいて，1つのレイヤのプロトコル処理を，オペレーティングシステムでサポートされているスレッドにより実現し，各スレッドを個別のプロセッサに割り当てる．以下，1つのレイヤのプロトコル処理を行うスレッドをレイヤスレッドと呼ぶ．
- レイヤ間インタフェースのための制御情報であるプリミティブや，回線上を転送される各レイヤのPDU（Protocol Data Unit）は，ヒープ領域に確保されたバッファ領域上に作成し，複数のレイヤスレッドから参照可能とする．
- 各レイヤスレッドが他のレイヤとできる限り並列に動作できるようにするために，できるだけレイヤスレッド間で排他制御を行わない方法を採用する．
- レイヤスレッド間において，共有情報に対する排他制御や，他のスレッドに対するイベント通知を行う場合は，1つのスレッドが繰り返しロックを要求するスピンロックや，スレッドのスリープや起動を実現するセマフォなどのカーネル機能を使い分け，処理のオーバヘッドをできるだけ小さくする．

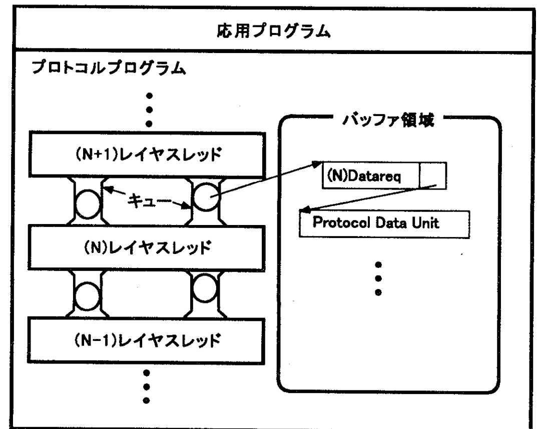



図1 プログラムの全体構成
Fig. 1 Structure of protocol program.

以上のような設計方針に基づいて，**図1**に示すような構成により，プロトコルの並列処理を行うプログラムを実現することとする．プロトコルプログラム全体はメインプログラムとリンクされるライブラリとして実現され，その中で各レイヤがスレッドとして実行される．レイヤ間でやりとりされるプリミティブや各レイヤで処理されるPDUは，バッファ領域上に作成される．また各レイヤスレッドの間には，バッファ領域上のプリミティブへのポインタを通知するキューが設けられる．

この方式の処理の流れの概要を**図2**に示す．各レイヤスレッドは通常，キューにつながれたのプリミティブを読み出し，プリミティブに対応したプロトコル処理を行い，必要に応じて出力のプリミティブをキューに書き込む．これらの動作は複数のレイヤスレッドで並列に実行され，その結果，各プリミティブの処理は図2に示すように，パイプライン的に実行される．た

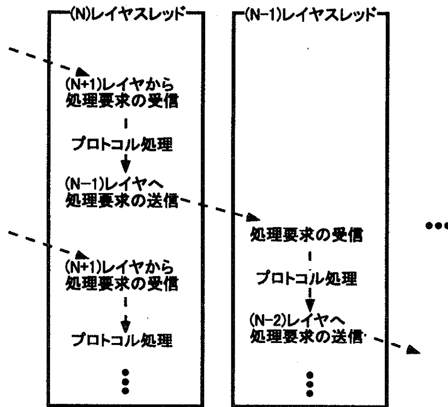



図2 並列処理の流れの概要
Fig. 2 Overview of parallel processing flow for sending data.

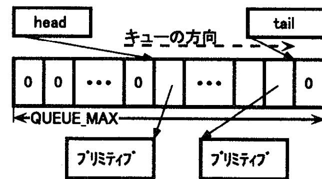



図3 キューの構造
Fig. 3 Structure of a queue.

だし，キューが空でプリミティブを読み込めない場合や，キューがフルで書き込めない場合には，そのスレッドは，他のレイヤスレッドが書き込み/読み出しを行い，それを通知するまでスリープして待つ．

この方式において複数のレイヤスレッドの間で排他制御またはイベント通知を行う必要性がある場合としては，以下が考えられる．

- レイヤ間のキューへアクセスする場合
- 空またはフルのキューに，他のレイヤスレッドが書き込み/読み出しを行いスリープしたスレッドを起動する場合
- バッファ領域から新たにバッファを確保する場合と，使用したバッファを解放する場合
- バッファ領域上に作成されたプリミティブやPDUにアクセスする場合

そこで，レイヤ単位の並列処理方式を実現するためには，これらの各々に対して，オーバヘッドの少ない実現方法を採用する必要がある．以下に，それぞれ実現方法について述べる．

### 2.2 レイヤ間のキューの実現方法

レイヤ間のキューについては，レイヤスレッドがキューに，排他制御のためのロックをかけることなしに，並列に書き込みと読み出しができるように，以下のような方法を用いる．

(1) キューは，**図3** に示すように，プリミティブへのポインタを持つ固定長（QUEUE_MAX）の配列と，読み出す要素と次に書き込む要素を示す変数（head と tail）から構成される．プリミティブがつながれていない配列の要素は‘0’となっている．

(2) 書き込む側のレイヤスレッドは tail のみにアクセスし，tail の要素が‘0’である場合は，キューが書き込み可能であると判断し，tail の要素にプリミティブのポインタを代入し，tail を1進める（tail に (tail+1)%QUEUE_MAX を代入する）．一方，tail の要素にすでにプリミティブのポインタが代入されている場合は，キューがフルであると判断し，書き込み失敗とする．なお，キューに書き込めなかったプリミティブはレイヤ内部で保持する必要がある．

(3) 読み出す側のスレッドは，head の要素が‘0’でなければそのポインタを読み出し，head の要素を‘0’とし，head を 1 進める（(head+1)%QUEUEMAX を head に代入する）．

この実現方法では，読み書きを行うスレッドをそれぞれ1つずつに限定し，ポインタを用いたリストではなく配列を使用し，キューの制御情報を操作する際に，全体にロックをかける必要をなくし，さらに，配列の要素にプリミティブがつながれているか否かの情報を持たせることにより，読み出しと書き込みを並列に行うことを可能とした．

### 2.3 レイヤスレッドのスリープ/起動方法

前述のように，各レイヤスレッドは，空のキューへのプリミティブの到着またはフルとなったキューからの読み出しを待つ必要がある．さらに，各レイヤスレッドは上位と下位に対して2組のキューを持つ．このため，レイヤスレッドのスリープ/起動を，セマフォを用いて以下のように実現することとした．

(1) 各レイヤスレッドは，セマフォを外部変数として持つ．

(2) 各レイヤスレッドは，上位と下位のキューが空でないか，内部で保持したプリミティブをキューに書き込むことが可能である場合は，つねにキューからのプリミティブの読み出し，または，保持したプリミティブの書き込みを行う．

(3) このとき，スリープしている隣接するレイヤスレッドを起動するために，そのレイヤスレッドのセマフォにV操作を行う．

(4) 自身のレイヤの処理がなくなった場合は，その

スレッドはセマフォにP操作を行う．そのセマフォ値が0になるとそのスレッドはスリープする．

### 2.4 バッファ領域の管理方法

プリミティブ，PDU，コネクション管理情報などのためのバッファを確保/解放するために，バッファ領域の管理情報にアクセスするスレッドの間で，排他制御を行う必要がある．そこで，このオーバヘッドを減らすために，次のようにしてバッファ管理を行うこととした．

#### 2.4.1 ローカルバッファ領域の管理

コネクション管理情報など，1つのレイヤ内のみで使用される情報のために，ローカルバッファ領域を設ける．ローカルバッファ領域は，スレッドごとに用意され，他のスレッドからのアクセスや確保/解放はない．

#### 2.4.2 グローバルバッファ領域の管理

プリミティブやPDUなどのためのグローバルバッファは，複数のレイヤスレッドにより確保/解放される必要がある．本並列処理方式では，その際の排他制御によるスレッド間の競合をなるべく避けるために，グローバルバッファの管理方法に対して，以下の方針を採用した．

- レイヤごとにグローバルバッファ用の領域を用意し，1つの領域からバッファを確保するスレッドを特定する．
- グローバルバッファ領域上には，空き領域と確保されたバッファとが混在するが，現在使用している空き領域内でバッファを確保できる場合は，確保の処理を解放の処理と同期をとることなく実行する．
- バッファの解放の処理においては，管理情報に対して排他制御を行う．

これらの方針に基づいて，以下のような管理方法を採用した．

(1) 1つのスレッドの管理するグローバルバッファ領域においては，空き領域は図4に示すように空き領域管理リストにより管理される．このリストはバッファ領域中の空き領域を表すFACB（Free Area Control Block）から構成され，FACBは空き領域の先頭と最後部へのポインタ（**TopPtr**と**BottomPtr**），他のFACBとの間で双方向リンクを構成するためのポインタを含む．ここで，バッファ領域上の空き領域には，バッファ制御用の情報は含まない．

(2) 確保されたバッファは，先頭に制御情報として，そのバッファのサイズを含む．

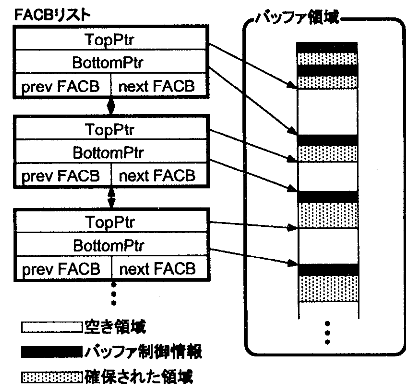



図4　グローバルバッファ領域の構成
Fig. 4　Structure of global buffer area.

(3) バッファの確保は，次の手順で行う．

- バッファの確保を要求するスレッドは，まず空き領域管理リストにロックをかけることなく，その先頭のFACBの**TopPtr**と**BottomPtr**を参照し，参照した時点での空き領域のサイズ（**BottomPtr-TopPtr**）が，要求されたバッファサイズよりも大きい場合は，その空き領域内にバッファを確保し，バッファの先頭にそのサイズを設定し，**TopPtr**をずらして，バッファの先頭ポインタを返す．
- 空き領域のサイズが，要求されたバッファサイズよりも小さい場合は，空き領域管理リスト全体にロックをかけ，現在参照しているFACBを，空き領域管理リストの最後に並べかえるとともに，その次の制御ブロックを参照し，それが管理する空き領域からバッファを確保するよう試みる．空き領域管理リストの各FACBに対して，バッファが確保されるまでこの処理を続ける．
- 空き領域管理リストの管理する空き領域内にバッファが確保できない場合は，さらに別のバッファ領域を割り当てる．
- 空き領域管理リストのロックについては，スピンロックを用いる．

このような処理により，先頭のFACBの管理する空き領域内にバッファが確保できる限りは，空き領域管理リストをロックする必要がない．

(4) バッファの解放は，次の手順で行う．

- 解放を要求するスレッドは，空き領域管理リスト全体にロックをかける（スピンロック）．FACBを先頭からたどり，解放するバッファを挿入すべき位置を求める．

- 解放される領域が，前後のFACBの管理する空き領域と隣接しているかを判断し，この領域を管理するFACBの登録，前後のFACBのBottomPtrまたはTopPtrの更新，前後の空き領域の統合のうちのいずれかを行う．

このバッファ管理方法においては，空き領域管理リストの先頭のFACBのTopPtrを参照するスレッドを1つと限定している．そのため，スレッドが要求するバッファが，先頭のFACBが管理する空き領域内で確保できるときには，TopPtrに要求するサイズを加算するだけでバッファを確保でき，他のスレッドの解放処理との排他制御が不要となっている．一方，解放処理には空き領域管理リストをロックすることとしているが，確保処理を並列に実行可能とするために，先頭のFACBのTopPtrについては操作しない．

#### 2.4.3 ヒープ領域の制御

ローカルバッファおよびグローバルバッファのためのバッファ領域は，プログラム起動時および空き領域が不足した時点で，ヒープ領域から確保される．このためにレイヤスレッドが，ヒープ領域にアクセスするためのロック（スピンロック）を用意している．

### 2.5 プリミティブおよびPDUへのアクセス制御方法

プリミティブとPDUについては，以下の方法を用いることにより，複数のレイヤのスレッドがそれらにアクセスするための排他制御を最小限としている．

#### 2.5.1 プリミティブへのアクセス制御

あるレイヤスレッドがプリミティブを作成し隣接レイヤに渡した後は，そのレイヤスレッドにはそれにアクセスさせないこととする．これにより，グローバルバッファ領域上のプリミティブへのアクセスに対する排他制御は不要となる．

#### 2.5.2 PDUへのアクセス制御

グローバルバッファ領域上のPDUとしては，レイヤ内におけるユーザデータのコピーを避けるために，PDUのデータ自身を格納するバッファ本体と，バッファ本体を指し示すバッファ・ディスクリプタから構成されるPDUバッファを採用する[7]．バッファ・ディスクリプタには，バッファ本体中の作業領域とPDUデータの位置，バッファ長とPDUデータ長，複数のバッファ本体を連結するためのディスクリプタへのポインタなどが含まれる．またバッファ本体には，それを参照しているディスクリプタの数を示すリファレンス・カウントと，リファレンス・カウントにアクセスするためのロックが含まれる．

プロトコル処理においては，送信処理において再送用に保持する場合を除き，1つのレイヤがPDUの処理を終了し，隣接するレイヤに渡した後は，そのレイヤはそのPDUにはアクセスしない．また，1つのレイヤは，PDUのうち，そのレイヤのヘッダのみをアクセスし，他の領域はユーザデータとして扱い内容を参照/更新することはない．プロトコル処理のこれらの性質を考慮し，複数のスレッドがPDUにアクセスする場合の競合を避けるために以下の方法を用いることとした．

- あるレイヤで再送用にPDUを保持する場合には，そのPDUに対するディスクリプタを新たに作成する．これにより，すべての場合で，バッファ・ディスクリプタはスレッド間で共有されることはなくなる．
- バッファ本体はスレッド間で共有される．バッファ本体に格納されたPDUのデータについては，各レイヤのスレッドがアクセスする箇所が定まっているため，ロックによる排他制御は行わない．一方，スレッドがバッファ本体中のリファレンスカウントを操作するときは，ロックをかける．この場合リファレンスカウントの処理時間は非常に小さいと考えられるため，スピンロックを用いる．

## 3. 実装方法

前章の設計に基づいて，プロトコルの並列処理方式を実装することとした．実装にあたっては，並列処理方式を実現する並列処理用ライブラリを作成し，それを用いてプロトコルプログラムを開発するというアプローチを採用した[4]．

### 3.1 並列処理用ライブラリ

#### 3.1.1 レイヤ間キュー

レイヤ間インタフェースに使用するキューの領域は，プログラム起動時に確保する．その領域をキューとして使用するために，以下の関数を設けた．

(1) **p_cremsgq_r**（引数：キューの名前）/ **p_cremsgq_s**（引数：キューの名前）
2.2節で示したデータ構造を用いて，プリミティブ受信用および送信用キューをそれぞれ作成する．戻り値として，作成したキューのIDを返す．

(2) **p_putmsg**（引数：キューID，プリミティブ）/ **p_getmsg**（引数：キューID）
2.2節のアルゴリズムにより，プリミティブの書き込みと読み出しをそれぞれ行う．書き込み時にキューがフルの場合は $-1$ を，読み出し時にキューが空である場合はNULLを，それぞれ返す．

#### 3.1.2 レイヤスレッドのスリープ/起動

(1) **p_wait**（引数：レイヤ番号）
指定したレイヤスレッドに対応するセマフォに対して，P操作を行いスリープさせる．

(2) **p_wakeup**（引数：レイヤ番号）
指定したレイヤスレッドがスリープしている場合に，対応するセマフォに対してV操作を行い，起動させる．

#### 3.1.3 バッファ管理

各レイヤスレッドは，起動時に`malloc()`を用いて，グローバルとローカルのバッファ領域のためのメモリブロックを確保する．その後は，各スレッドは以下の関数を用いて，バッファの確保/解放を行う．

(1) **p_galloc**（引数：レイヤ番号，サイズ）
2.4.2 項に示したアルゴリズムに従って，要求されたサイズのグローバルバッファを確保し，確保したバッファのポインタを返す．

(2) **p_malloc**（引数：レイヤ番号，サイズ）
対応するローカルバッファ領域からバッファを確保し，確保したバッファのポインタを返す．

(3) **p_free**（引数：解放するバッファのアドレス）
確保したバッファを解放する．指定されたバッファがグローバルバッファである場合は，2.4.2 項のアルゴリズムに従って解放処理を行う．ローカルバッファの場合は通常の解放処理を行う．

#### 3.1.4 PDUのアクセス制御

PDUバッファにアクセスするための関数としては，筆者らが先に開発したOSIプロトコル用のPDUバッファ制御用関数[7]を，以下のように変更して使用することとした．

- グローバルバッファ領域上にPDUバッファを作成するために，制御用関数のバッファ確保とバッファ解放の関数として，それぞれ`p_galloc()`と`p_free()`を使用する．
- 2.5.2 項に示したように，PDUバッファのバッファ本体にロックを設け，リファレンス・カウントにアクセスする場合は，スピンロックを用いて排他制御を行う．

### 3.2 プロトコルプログラムの構造

前節の並列処理用ライブラリ関数を使用したプロトコルプログラムは，メイン関数でレイヤのスレッドを起動する．各レイヤのスレッドは，1つのプロセッサに割り当てられた後は，以下の処理を繰り返す．スレッドは，メモリブロックの確保，キューの作成などの初期化処理を行った後，**図5**のような構造を持つメインループに入る．メインループでは，上位/下位レイヤスレッドからのキューを監視し，プリミティブがある

```
for( ;; ) {
    while( キューにプリミティブを送信可能 || キューから受信可能 ) {
        /*** 送信処理 ***/
        p_getmsg( 上位レイヤへのキュー );
        if( プリミティブ取得成功 ) {
            p_wakeup( 上位レイヤ )
            ....プロトコル処理.... }
        if( 下位レイヤへのプリミティブあり ) {
            p_putmsg( 下位レイヤへのキュー )
            if( 下位レイヤに送信成功 ) {
                p_wakeup( 下位レイヤ ) }
        }
        /*** 受信処理 ***/
        ....送信処理と同様に受信処理を行う.... }
    p_wait( ) }
```

図5 レイヤプログラムのメインループ
Fig. 5 Main loop in layer program.

場合，それを取得しプロトコル処理を行う．処理が終了すると，必要なプリミティブを作成して隣接レイヤへ送信する．このとき，隣接レイヤへのキューがフルである場合は，レイヤ内に保持しておき，キューにプリミティブが送信可能となった時点で送信を行う．

### 3.3 使用した計算機環境

上記のような並列処理用ライブラリとプロトコルプログラムを実装するために，Silicon Graphics 社のONYXワークステーション（OS: IRIX R5.3 System V, 12 CPU）を用いることとした．ここで，IRIXにおいては，アドレス空間を共有するプロセスをスレッドとして用いた．また，ライブラリの作成においては，以下に示すIRIXシステムコールを使用した．

- **ussetlock()** スピンロックを行う．競合したときはロックを獲得するまでビジーウエイトする．
- **usunsetlock()** **ussetlock()**により獲得したロックを解除する．
- **uspsema()** セマフォに対してP操作を行う．セマフォ値が0となった場合は，本関数を呼び出したプロセスがスリープする．
- **ustestsema()** セマフォ値を取得する．
- **usvsema()** セマフォに対してV操作を行う．

## 4. 性能評価

### 4.1 性能評価の方針

提案する並列処理方式の性能評価を行うにあたり，以下のような方針を採用することとした．

- 3章に示した実装方法に基づいて，評価用のプロトコル並列処理プログラムを作成し，実際のネットワークを介して通信実験を行った場合のスループットを測定する．ネットワークとしては，800 Mbps

の転送速度を有する HIPPI（High Performance Parallel Interface）を用いる．

- レイヤ数を変化させた性能評価を容易に実行可能とし，また，各レイヤのプロトコル処理量を同一とし，結果の解析を容易にするために，評価用プログラムでは，同一のプロトコルを複数レイヤ積み重ねた構成を用いる．
- 評価用プログラムのプロトコルとしては，データの受信確認，フロー制御，誤り再送などを機能を有するコネクション型のプロトコルを採用する．
- 以下の 2 つの観点から性能評価を行う．
  – セグメンティングを用いない場合と用いる場合など，レイヤのプロトコル機能を変更して，スループットを測定する．
  – CPU の増加にともなうプロトコルの並列処理方式のオーバヘッドを評価するために，レイヤ数（CPU 数）を変化させてスループットを測定する．
- 比較のために，プロトコル処理を行わない，HIPPI を介したデータ転送のスループットも計測する．
- さらに，同様のプロトコルプログラムをメッセージ単位の並列処理方式を用いて並列実行させた場合の性能評価を行い，提案した並列処理方式の有効性を示す．この評価においては，HIPPI を用いた通信実験に加えて，内部折返しの形態も用いる．
- 加えて，提案する並列処理方式において，ボトルネックとなる可能性のあるグローバルバッファの確保および解放のオーバヘッドについて，詳細な評価を行う．

### 4.2 通信実験形態および評価用プログラムの構成

通信実験においては，**図 6** に示すように，HIPPI を有するワークステーションの折返し通信を行う．すなわち，ワークステーションの HIPPI の Destination（DST）ポートと Source（SRC）ポートに，1 本の HIPPI ケーブルの両端を接続し，データを折返しで転送する[5]．

また，評価用プログラムでは，各レイヤスレッドにおいて，データの送信時における PDU のヘッダの付加と，データ受信時における PDU のヘッダの除去，およびそれらに対応するプロトコル処理を行う．最上位のレイヤスレッドにおいては，これに加え，送信するユーザデータ用のバッファの確保と，受信したユーザデータのバッファの解放を行う．さらに，最下位のレイヤスレッドとして，HIPPI へのデータの送信と HIPPI からのデータ受信のみを行うスレッドを起動する．

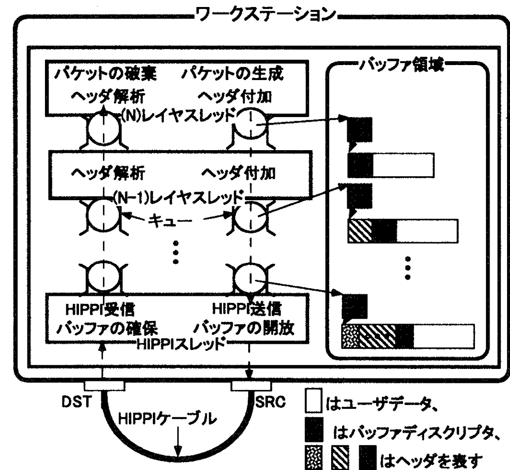



**図 6**　通信実験形態
Fig. 6　Configuration of experiment for performance evaluation.

### 4.3 評価用のプロトコル機能

評価用のプロトコルとして，OSI のトランスポートプロトコルクラス 4（TP4）[8]を採用した．実装した TP4 の機能を以下に示す．

- コネクションが確立された後のデータ転送フェーズのみをサポートした．
- TP4 で定められている手順要素[8]のうち，セグメンティングとリアセンブリング，TPDU のトランスポートコネクションへの関連付け，TPDU の番号付け（拡張フォーマットを使用），TPDU の確認と保持（AK TPDU を使用），クレジットによる明示的フロー制御，再順序付けを実装した．

### 4.4 評価方法

- スループットの測定においては，ユーザデータの連続的な転送を行い，受信側の最上位レイヤにおいて，ユーザデータの受信開始から終了までの時間で，転送したユーザデータの総量を割った値をスループットとして使用する．
- フロー制御によりデータ転送の性能が制限されないように，クレジットとして十分大きな値（50 TPDU）を採用する．また，受信確認のための AK TPDU の送信オーバヘッドの影響を少なくするために，AK TPDU はクレジットの半分の DT TPDU を受信した時点で送信する．また，TP4 で規定されているチェックサムについては使用しない．
- プロトコル機能を変更した性能評価においては，以下のような方法を用いる．
  – 4 つの CPU を用いて，レイヤ数を 1 から 3

まで変更させる．

– 各レイヤで使用する機能がすべて同一の場合と，一部のレイヤにおいて異なる機能を使用した場合の双方に対して，ユーザデータサイズを変更してスループットを測定する．

– レイヤの機能を変更して評価を行うために，コピーをまったく行わない場合と，ヘッダの付加と除去において，memcpy() 関数を使用してユーザデータをコピーする場合，および，セグメンティングとリアセンブリングを行わない場合と行う場合をそれぞれ使用する．

- 並列処理方式のオーバヘッドの評価においては，以下のような方法を用いる．

– 12 個の CPU を用いて，レイヤ数を 1 から 11 まで変化させる．

– 各レイヤにおいて，ユーザデータのコピー，および，セグメンティングとリアセンブリングを行わない場合に対して，スループットを測定し，レイヤ数の増加にともなう全体の処理能力の増加の度合を評価する．なお，評価をユーザデータサイズを変更して行う．

- メッセージ単位の並列処理方式との比較においては，以下のような方法を用いる．

– 同様なプロトコルをメッセージ単位に並列処理するプログラムを作成する．

– 12 個の CPU を用いて，レイヤ単位の並列処理では，レイヤ数を 1 から 9 まで変化させ，メッセージ単位の並列処理では，メッセージの並列処理を行うスレッド数を 1 から 9 まで変化させる．各レイヤにおいては，並列処理方式のオーバヘッドの評価と同様に，ユーザデータのコピー，および，セグメンティングとリアセンブリングを行わない場合に対して，ユーザデータサイズを変更し，スループットを測定する．

– 前述のように，評価は，HIPPI を用いた通信実験と，内部折返しの形態の双方で行う．

- グローバルバッファの確保および解放のオーバヘッドの評価については，提案する並列処理方式のオーバヘッドの評価における 11 レイヤの場合で，確保時のロックの要求数，ロックの競合回数を求める．

### 4.5 評 価 結 果

#### 4.5.1 プロトコル機能を変更した性能評価

まず，すべてのレイヤが同一の機能を使用する場合として，ユーザデータのコピー，および，セグメンティングとリアセンブリングを行わない場合に対して，レイヤ数およびユーザデータサイズを変化させてスループットを測定した．その結果を図 7 に示す．

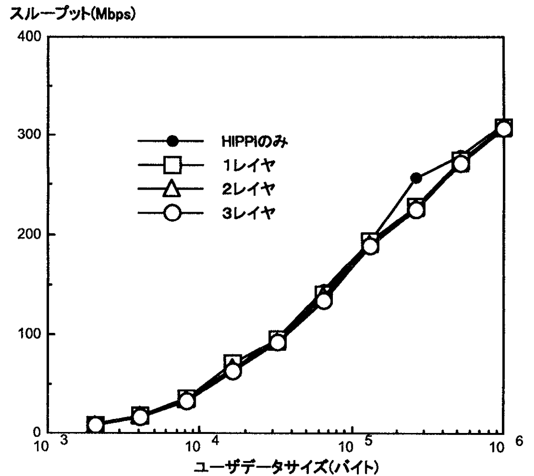



図 7 HIPPI を介した TP4 のスループット
(すべてのレイヤが同一機能の場合)
Fig. 7 Throughput of TP4 Program over HIPPI (with the same functions for every layer).

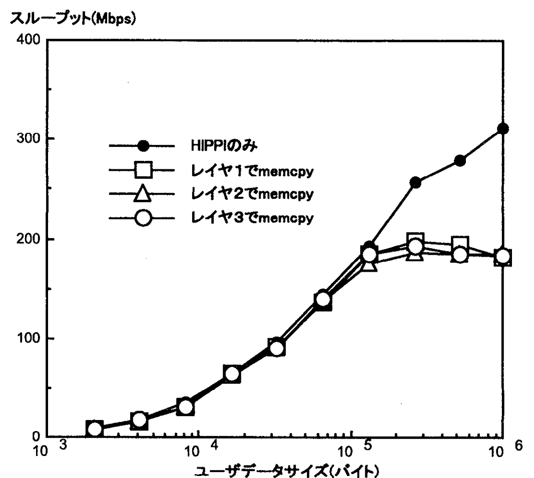



図 8 HIPPI を介した TP4 のスループット
(特定のレイヤでユーザデータコピーを行った場合)
Fig. 8 Throughput of TP4 Program over HIPPI (with user data copying in one layer).

次に，レイヤ数を 3 とし，そのうちの特定レイヤのみに対して，プロトコル機能を増加させプロトコル処理速度を低下させて，スループット測定を行った．特定の 1 つのレイヤ（レイヤ 1 からレイヤ 3）において，ユーザデータのコピーを行った場合のスループットの測定結果を図 8 に示す．また，1 つの最大 TPDU サイズを 66,000 バイトに設定し，それを超えるサイズのユーザデータの送信が要求された場合は，セグメン

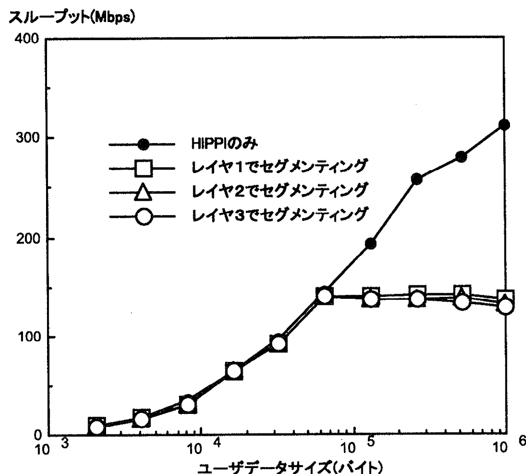



図9 HIPPIを介したTP4のスループット
(特定のレイヤでセグメンティングを行った場合)
Fig. 9 Throughput of TP4 Program over HIPPI
(with segmenting and reassembling in one layer).

ティングとリアセンブリングを行わせた場合の結果を，**図9** に示す．

これらのスループット測定より以下の結果を得た．

- 図7より，レイヤ数が1，2，3と増加した場合でも，全体のスループットはほぼ同一であることが分かる．さらに，このスループットは，プロトコル処理を行わない，HIPPIを介したデータ転送のスループットとほぼ一致している．
- 図8より，1つのレイヤにユーザデータのコピーを行わせた場合は，ユーザデータサイズが128Kバイト（131,072バイト）を超えた場合は，それ以上ユーザデータサイズを増加しても，約200Mbpsでほぼ一定となっている．またこの結果は，ユーザデータのコピーを行うレイヤの位置には依存していない．
- 図9より，あるレイヤでセグメンティングとリアセンブリングを行った場合は，セグメンティングが行われると，すなわち，ユーザデータサイズが64Kバイト（65,536バイト）を超えると，約140Mbpsで一定となっている．またこの結果は，セグメンティングとリアセンブリングを行うレイヤの位置には依存しない．

#### 4.5.2 並列処理方式のオーバヘッドの評価

この評価においては，8,192バイト，32,768バイト，131,072バイトのそれぞれのユーザデータサイズに対して，レイヤ数を1から11まで変化させてスループットを測定した．さらに，レイヤ数の増加にともなう全体の処理能力の増加の度合を評価するために，以下の2つの指標を計算した．

- プロトコルの処理能力の相対値，すなわち，スループット × レイヤ数を，レイヤ数1の場合のスループットで割った値．
- 並列化の効率，すなわち，相対処理能力をレイヤ数で割った値．

これらの結果を，**表1**，**表2**，**表3** に示す．これらの表より，レイヤ数が5の場合で，並列度（使用されたCPU数）の90%以上，レイヤ数が9から11の場合で，約80%以上の効率を保っていることが分かる．また，ユーザデータサイズを増大させると，効率は向上した．

#### 4.5.3 メッセージ単位の並列処理方式との比較

上述と同一のプロトコル機能を有し，メッセージ単位の並列処理を行うプログラムを作成し，スループットを測定した．そのプログラムの概要を以下に示す（**図10** 参照）．

- プログラムはユーザスレッド，メッセージスレッド，HIPPI/内部折返しスレッドから構成される．メッセージスレッドは複数存在し，各メッセージスレッドはユーザスレッドとHIPPI/内部折返しスレッドとのインタフェースとしてキューを持つ．さらに，各スレッドはグローバルバッファ領域とローカルバッファ領域を持つ．ここで，バッファ領域制御とキュー制御の実装方法は，提案するレイヤ単位の並列処理方式のための評価プログラムと同一である．
- ユーザスレッドは送信するユーザデータを作成し，その送信要求プリミティブを，メッセージスレッドへ順番に与える．すなわち，N個のメッセージスレッドが起動されている場合は，i番目に送信されるユーザデータは `((i%N)+1)` 番目のメッセージスレッドで処理される．また，ユーザスレッドは順番にメッセージスレッドのキューからプリミティブを読み込み，受信したユーザデータのバッファを解放する．
- メッセージスレッドは，4.3節で述べたTP4の機能を複数レイヤサポートする．各レイヤに対して，それぞれのコネクションの状態をコネクション管理情報（CCB: Connection Control Block）を用いて管理する．CCBはグローバルバッファを用いて実現され，複数のメッセージスレッドからアクセスされる．このため，各メッセージスレッドは，1つのレイヤの処理を行う場合は，その間，対応するCCBにロックをかける．
- メッセージ単位の並列処理方式では，同一のコネクションのPDUが異なるスレッドで処理されるため，スレッド間でPDUの順番の入れ替わりが

表1 並列処理方式のオーバヘッド（ユーザデータサイズ：8,192バイト）
Table 1 Parallel processing overhead (User data: 8,192 bytes).

| レイヤ数 | 1 | 3 | 5 | 7 | 9 | 11 |
|---|---|---|---|---|---|---|
| スループット（Mbps） | 35.6 | 33.9 | 33.0 | 31.4 | 30.5 | 28.2 |
| 相対処理能力 | 1 | 2.86 | 4.63 | 6.17 | 7.71 | 8.71 |
| 効率 | 1 | 0.95 | 0.93 | 0.88 | 0.86 | 0.79 |

HIPPI回線のスループット：35.8 Mbps

表2 並列処理方式のオーバヘッド（ユーザデータサイズ：32,768バイト）
Table 2 Parallel processing overhead (User data: 32,768 bytes).

| レイヤ数 | 1 | 3 | 5 | 7 | 9 | 11 |
|---|---|---|---|---|---|---|
| スループット（Mbps） | 95.6 | 92.7 | 91.4 | 89.9 | 87.9 | 84.0 |
| 相対処理能力 | 1 | 2.91 | 4.78 | 6.58 | 8.28 | 9.67 |
| 効率 | 1 | 0.97 | 0.96 | 0.94 | 0.92 | 0.88 |

HIPPI回線のスループット：98.2 Mbps

表3 並列処理方式のオーバヘッド（ユーザデータサイズ：131,072バイト）
Table 3 Parallel processing overhead (User data: 131,072 bytes).

| レイヤ数 | 1 | 3 | 5 | 7 | 9 | 11 |
|---|---|---|---|---|---|---|
| スループット（Mbps） | 192.4 | 188.6 | 187.5 | 185.9 | 184.0 | 181.9 |
| 相対処理能力 | 1 | 2.94 | 4.84 | 6.76 | 8.61 | 10.4 |
| 効率 | 1 | 0.98 | 0.97 | 0.97 | 0.96 | 0.95 |

HIPPI回線のスループット：193.5 Mbps

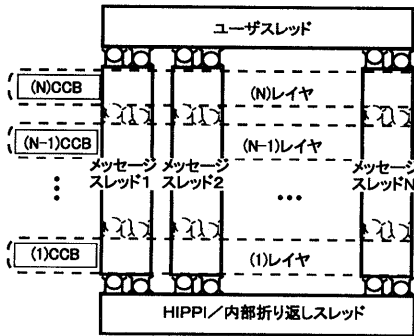



図10 メッセージ単位の並列処理の評価のためのプログラム構成
Fig. 10 Program structure for evaluating processor-per-message parallel processing.

発生する場合がある．今回の評価においては，入れ替わりが発生した場合はTP4の最順序付けにより対応した．

- レイヤ単位の並列処理方式との比較のため，メッセージスレッドの数とレイヤ数を同一とした．
- HIPPI/内部折返しスレッドは，HIPPIへの送受信処理または内部折返し処理を行う．受信されたHIPPIデータパケットはそれぞれメッセージスレッドへ分配される．内部折返しについては，ユーザデータのコピーは行っていない．

レイヤ数とメッセージスレッド数を変化させ，ユーザデータサイズが32,768バイト，131,072バイトの場合について，HIPPIを用いた通信実験と内部折返しによるスループットを計測した．その結果を，それぞれ**表4**，**表5**に示す．

メッセージ単位の並列処理方式では，ユーザデータサイズやHIPPI通信実験または内部折返しにかかわらず，すべての場合に，スレッド数（レイヤ数）の増加とともにスループットが低下している．たとえば，HIPPI通信実験において，ユーザデータサイズが131,072バイト，レイヤ数が9の場合は，メッセージ単位の並列処理方式では，スループットが約30%減少している．これは並列化の効率が約0.7であることを意味しており，レイヤ単位の並列処理方式の効率の0.95に比べ小さい値となっている．

特に内部折返しの場合で，メッセージ単位の並列処理方式のスループットの低下が目立つ．この理由は，あるメッセージスレッドが，1つのレイヤのプロトコル処理を行おうとした場合，他のスレッドがそのレイヤを処理中であると，その処理が終了するまで，そのスレッドは待たなければならず，並列度が増加すると，処理停止回数が増加しスループットが下がるためであると考えられる．

なお，内部折返しにおいてスレッドが1つの場合で，レイヤ単位とメッセージ単位の並列処理方式のスループットが異なるのは，レイヤ単位では最上位のプロトコルレイヤ内でユーザデータを作成しているのに

表4 並列処理方式の比較（ユーザデータサイズ：32,768バイト）
Table 4 Comparison of parallel processing schemes (User data: 32,768 bytes).

| | レイヤ数 | 1 | 3 | 5 | 7 | 9 |
|---|---|---|---|---|---|---|
| HIPPI 通信実験 | レイヤ単位 (Mbps) | 95.6 | 92.7 | 91.4 | 89.9 | 87.9 |
| | メッセージ単位 (Mbps) | 95.5 | 85.4 | 71.1 | 62.2 | 57.0 |
| 内部折返し | レイヤ単位 (Mbps) | 1990 | 1130 | 896 | 636 | 483 |
| | メッセージ単位 (Mbps) | 1910 | 379 | 220 | 163 | 136 |

表5 並列処理方式の比較（ユーザデータサイズ：131,072バイト）
Table 5 Comparison of parallel processing schemes (User data: 131,072 bytes).

| | レイヤ数 | 1 | 3 | 5 | 7 | 9 |
|---|---|---|---|---|---|---|
| HIPPI 通信実験 | レイヤ単位 (Mbps) | 192 | 189 | 188 | 186 | 184 |
| | メッセージ単位 (Mbps) | 190 | 182 | 160 | 146 | 136 |
| 内部折返し | レイヤ単位 (Mbps) | 6750 | 4400 | 3550 | 3600 | 1900 |
| | メッセージ単位 (Mbps) | 5300 | 950 | 328 | 279 | 238 |

表6 グローバルバッファの管理におけるロックの状態（11プロトコルレイヤ，ユーザデータサイズ131,072バイト，ユーザデータを10,000個送信した場合）
Table 6 Locks in global buffer area management (11 protocol layers with user data size is 131,072 bytes, and sending 10,000 user datas).

| スレッド | HIPPI | レイヤ1 | レイヤ2 | レイヤ3 | レイヤ4 | レイヤ5 |
|---|---|---|---|---|---|---|
| ロック付き確保回数/確保回数 | 1,433/22,889 | 1/11,669 | 1/11,441 | 1/11,112 | 1/11,001 | 1/10,890 |
| 競合回数/ロック回数 | 3/24323 | 0/11,654 | 86/11,441 | 0/11,065 | 0/10,957 | 0/10,849 |
| スレッド | レイヤ6 | レイヤ7 | レイヤ8 | レイヤ9 | レイヤ10 | レイヤ11 |
| ロック付き確保回数/確保回数 | 1/10,779 | 1/10,668 | 1/10,557 | 1/10,446 | 1/10,335 | 1,511/29,226 |
| 競合回数/ロック回数 | 0/10,742 | 0/10,636 | 0/10,530 | 0/10,423 | 0/10,320 | 201/31,686 |

対し，メッセージ単位の場合では，ユーザスレッドが別に存在しているための処理量の違いから生じると考えられる．

#### 4.5.4 グローバルバッファの管理方法に関する評価

提案する並列処理方式においては，グローバルバッファの確保と解放を行うために，レイヤスレッド間において排他制御の必要がある．特に，グローバルバッファの解放時には，必ず空き領域管理リストのロックをかける必要がある．このため，グローバルバッファ領域の管理におけるロックの競合について詳細に検討した．具体的には，HIPPI回線を介した通信実験において，以下の2つの回数を測定した．

- グローバルバッファの確保において，空き領域管理リストのロックが行われた回数
- グローバルバッファの確保および解放において，空き領域管理リストのロックを行ったうちで，競合が生じた場合の回数

レイヤスレッドの数が11の場合に，131,072バイトのユーザデータを10,000回転送した場合の結果を，**表6**に示す．

この表の第1行より，グローバルバッファの確保の要求のほとんどが，空き領域管理リストへのロックを要求することなしに，先頭のFACBから確保されていることが分かる．レイヤ1からレイヤ10までのレイヤスレッドは，PDUバッファのバッファ・ディスクリプタを確保するのみであるため，空き領域管理リスト全体をロックする必要があるのは，最初の1回のみであった．また，レイヤ11のレイヤスレッドと，HIPPIの送受信を行うスレッドは，それぞれ，ユーザデータのためのグローバルバッファの確保と，HIPPI回線から受信したデータのためのグローバルバッファを確保するために，より多くのロックが必要であったが，それでも，ロックが必要となった確保要求の回数は，要求全体の10%以下であった．これらの結果は，提案する並列処理方式において，グローバルバッファの確保の多くが，空き領域管理リストのロックなしで行うことが可能であることを示している．

表6の第2行は，ロックの競合回数，すなわち，各

レイヤのスレッドにおいて定義されたロックに対して，スピンロックが要求された回数と，その要求が失敗した回数を示す．この結果から，グローバルバッファの確保および解放においては，空き領域管理リストのロックを要求した場合，99%以上が成功したことが分かる．このように，提案する並列処理方式においては，グローバルバッファの確保および解放におけるロックが，性能低下を生じさせていないと考えられる．

## 5. 考 察

### 5.1 並列処理方式に関する考察

(1) 提案する方式では，レイヤ単位の並列処理を採用したため，並列に動作するスレッドの間でのオーバヘッドを減少できたと考えられる．たとえば，スレッド間のデータのやりとりはキューを介したプリミティブの送受だけとなり，2.2 節で述べたキューにより，そのために同期をとることは不要となる．また，レイヤ単位の並列の特徴により，2.5 節に述べたように，グローバルバッファ領域上のプリミティブやPDUのためのバッファへのアクセスは，PDUバッファのリファレンス・カウントにアクセスする以外は，排他制御を行う必要はない．したがって，レイヤスレッドの間において並列処理にともなって生ずるオーバヘッドは，グローバルバッファ領域上のバッファを確保/解放する際の排他制御のみとなる．

(2) グローバルバッファ領域から確保されるプリミティブ/PDU用バッファは，そのプリミティブやPDUの処理が行われている間のみ確保され，処理が終わると解放される．すなわち，コネクション管理テーブルなどと比較すると，比較的短時間で解放される．このため，2.4.2 項で検討したバッファ領域の管理方式を用いると，多くの場合先頭の空き領域からロックなしでバッファを確保でき，バッファの確保における排他制御のオーバヘッドを減らすことが可能であると考えられる．これらは4.5.4 項に示したグローバルバッファの確保/解放におけるロックの回数の評価により，明らかとなっている．

(3) 従来から検討されているメッセージ単位の並列処理方式では，各レイヤの処理を行うに際して，並列に動作する各スレッドがコネクションの管理情報を排他制御する必要がある．このため，複数のスレッドがあるコネクションの特定のレイヤの処理を並列に行おうとした場合は，1つのスレッドのみが処理可能となるのみで，他のスレッドはその処理の終了を待つ必要がある．この結果，並列に動作するスレッド数が多くなっても並列処理の効果が上がらない．4.5.3 項に示した評価結果では，並列度が9（メッセージスレッド数が9）の場合には，効率は0.7程度となっている．また，文献 2) では，メッセージ単位の並列処理方式においては，TCP のようなコネクション型のプロトコルを並列に実行させた場合は，全体の性能は，並列度が6を超えた場合はそれ以上向上せず，その時点の効率は0.6程度であると報告している．

(4) 提案する方式では，レイヤスレッド間の同期のためのカーネル機能を，以下のように使い分けている．PDU用バッファのリファレンス・カウントや，グローバルバッファ領域の制御情報へのアクセスなどのように，単に他のスレッドの競合を待てばよい場合は，スピンロックを用いる．また，キューが空またはフルの場合の処理のように，他のスレッドの処理（キューの書き込みまたは読み出し）を待つ必要がある場合は，セマフォを使用する．

### 5.2 性能評価の結果に関する考察

(1) 図7の結果は，レイヤ数が1，2，3のすべての場合において，ほぼ同一のスループットを達成している．レイヤ数が3の場合は，1の場合に比べて，全体のプロトコル処理量は3倍となっている．したがって，同一のスループットを得たという結果は，並列処理により処理速度が3倍となったことを意味している．図7の結果より，レイヤ数が3までの範囲では，並列処理によりレイヤ数倍の性能を得たと考えることができる．

(2) 図7の結果は，HIPPIのみを介したデータ転送のスループットとほぼ一致している．したがって，この実験においては，通信プロトコル処理のオーバヘッドが，HIPPIの入出力処理のオーバヘッドに比べて小さかったと考えられる．そこで，各レイヤのプロトコル処理を増加させるために，`memcpy()` によるユーザデータのコピーを行い，レイヤ数を1から3まで変化させてスループットを測定する実験を，別途実施した．その結果は，ユーザデータが128 Kバイトを超えた場合は，コピーのほうがHIPPIの入出力処理よりもオーバヘッドが大きくなりスループットが約200 Mbpsで一定となったが，レイヤ数が1，2，3の場合のスループットはほぼ同一であった．したがって，プロトコル処理がボトルネックとなった場合も，(1)と同様に並列処理による性能向上が得られると考えられる．

(3) 図8の結果は，`memcpy()` によるユーザデータのコピーを行うレイヤスレッドのオーバヘッドが，他のレイヤスレッドよりも大きいために，全体のスループットが，そのレイヤによって決定されることを示している．

(4) 図9の場合は，次のように解釈できる．

- この場合は図8と異なり，プロトコル処理のオーバヘッドは大きくない．
- ユーザデータサイズが64Kバイトを超えると，特定のレイヤでセグメンティングが行われ，それ以下のレイヤスレッドおよびHIPPIの送受信スレッドには66,000バイトの送信要求が行われる．
- この場合，ボトルネックはHIPPIの入出力処理となり，したがって，ユーザデータサイズを増加させても，ユーザデータサイズが64Kバイトの場合のスループットとほぼ同一の値が得られる．

## 6. おわりに

本稿では，各レイヤのプロトコル処理をオペレーティングシステムが提供するスレッド機能を用いて並列に実行する通信プロトコルの実装方式を提案するとともに，HIPPIを介した通信実験により，提案した方式の性能評価を行った結果を示した．レイヤ単位の並列処理方式は，従来から研究されているメッセージ単位の並列処理方式に比べて，並列化のオーバヘッドの影響を受けやすい一方で，プロセッサ間で共有する必要のある情報を減少することができる．提案する並列処理方式においては，プロセッサ間の同期のオーバヘッドを少なくするために以下の方法を採用している．

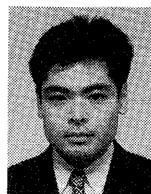
- アクセスの際に排他制御が必要ないキューを使用することにより，レイヤスレッド間で，同期することなく，プリミティブのやりとりを行う．
- グローバルバッファ領域上に作成されたプリミティブやPDUについては，レイヤスレッドは排他制御なしにアクセスを行う．
- 複数のレイヤスレッドが参照するグローバルバッファ領域に対しては，バッファ確保の処理を，バッファ解放の処理と並列に実行可能とする．

この方式を用いて，データの受信確認やフロー制御の機能を有するコネクション型プロトコルのプログラムを作成し，HIPPIを介し性能評価を行い，並列度が11の場合に最大10.4倍の性能向上を実現できるという結果を得た．このことから，提案する方式は，小さいオーバヘッドでレイヤごとのプロトコルを並列に実行させることが可能であると考えられる．今後は，本方式を用いてOSIやTCP/IPなどの現実のプロトコルの実装を進める予定である．

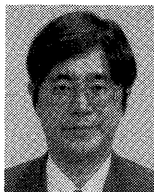
## 参考文献


1) Goldberg, M., Neufeld, G. and Ito, M.: A Parallel Approach to OSI Connection-Oriented Protocols, *Proc. 3rd IFIP Workshop on Protocols for High-Speed Networks* (May 1992).
2) Bjorkman, M. and Gunningberg, P.: Locking Effects in Multiprocessor Implementations of Protocols, *Proc. SIGCOMM '93* (Sep. 1993).
3) 加藤聰彦，鈴木健二：共有メモリ型マルチプロセッサを用いた通信プロトコルの並列処理方式，情処マルチメディア通信と分散処理研究会，69-17 (Mar. 1995).
4) 佐藤，加藤，鈴木：レイヤ単位の並列処理方式を用いた通信プロトコルプログラムの実装に関する検討，第51回情報処理学会全国大会論文集，1E-3 (1995).
5) 佐藤友実，加藤聰彦，鈴木健二：レイヤ単位の並列処理方式を用いた通信プロトコルプログラムの実装，情報処理学会マルチメディア通信と分散処理研究会，73-19 (Dec. 1995).
6) 佐藤友実，加藤聰彦，鈴木健二：レイヤ単位の並列処理方式を用いた通信プロトコルプログラムのHIPPIによる通信実験，第52回情報処理学会全国大会論文集，1Bb-4 (1996).
7) 加藤聰彦，井戸上彰，鈴木健二：OSIプロトコル実装のためのユーザデータをコピーしないバッファ制御方式，情報処理学会マルチメディア通信と分散処理研究会，62-13 (Sep. 1993).
8) CCITT X.224: Transport Protocol Specification for Open Systems Interconnection for CCITT Applications (1988).
9) HIGH-PERFORMANCE PARALLEL INTERFACE – Mechanical, Electrical, and Signalling Protocol Specification (HIPPI-PH), ANSI X3.183-1991 (Nov. 1994).


(平成 9 年8月21日受付)
(平成10年7月 3 日採録)

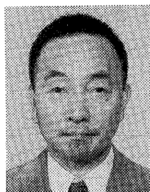


**佐藤 友実**（学生会員）
昭和46年生．平成6年電気通信大学電気通信学部電子工学科卒業．平成8年同大学院情報システム学研究科情報ネットワーク学専攻博士前期課程修了．現在同研究科博士後期課程在学中．通信プロトコルの高速化に興味を持ち，ハードウェアによるプロトコルの実装の研究に従事．



**加藤　聰彦**（正会員）

昭和31年生．昭和53年東京大学工学部電気工学科卒業．昭和58年同大学院博士課程修了．同年国際電信電話（株）入社．現在,（株）KDD研究所高速通信グループリーダ．工学博士．昭和62年から63年まで米国カーネギーメロン大学計算機科学科客員研究科学者．この間，OSI通信プロトコルの実装，通信システムの試験技法，プロトコルの形式記述，分散処理，ATM，高速通信などの研究に従事．昭和60年情報処理学会学術奨励賞，平成元年度元岡賞受賞．平成5年度より電気通信大学大学院情報システム学研究科客員助教授．電子情報通信学会会員．



**鈴木　健二**（正会員）

昭和20年生．昭和44年早稲田大学理工学部電気通信学科卒業．昭和44年から45年までオランダのフィリップス国際工科大学に招待留学．昭和51年早稲田大学大学院博士課程修了．同年国際電信電話（株）入社．現在,（株）KDD研究所代表取締役副所長．工学博士．この間，磁気記録，パケット交換方式，ネットワークアーキテクチャ，高速・分散処理の研究に従事．昭和62年度前島賞，平成4年度電子情報通信学会業績賞，平成7年度科学技術庁長官賞を各受賞．平成5年度より電気通信大学大学院情報システム学研究科客員教授．電子情報通信学会，IEEE各会員．