分散メモリ型並列計算機における並列入出力システム

松原正純[†] 沼 寿隆[†,☆]
板倉憲一[††] 朴 泰祐[†]

超並列計算機における大規模科学技術計算によって生じる膨大な量の計算結果を，実時間的に保存・表示するためには，現在よりも高性能かつ柔軟な入出力システムが望まれる．本研究では，その要件を満たすシステムを実現するために開発を進めている，入出力システムについて報告する．ネットワーク媒体としては，すでにコモディティ化されていてコストパフォーマンスに優れた 100Base-TX イーサネットを採用し，またこのネットワークを多重化することで，ネットワークのバンド幅を増強する．これにより，安価ではあるが全体として高速な環境を構築可能である．そして，この上に並列ネットワークの性能を効率良く引き出し，かつユーザにとって利便性の高い並列入出力システムを開発する．また，多重化したネットワーク間の動的負荷分散について，分散メモリ型と共有メモリ型の並列計算機の協調動作により，これを解決する手法を提案・実装する．実装した並列入出力システムを性能評価した結果，複数ある入出力装置を有効に利用でき，ネットワークの並列度に見合った性能向上が得られることが確認された．

Parallel I/O System on Distributed Memory Parallel Computers

Masazumi Matsubara,[†] Hisataka Numa,[†,☆] Ken'ichi Itakura[††] and Taisuke Boku[†]

Through and after a large scale computation on a massively parallel processor (MPP), a large quantity of data will be produced. It is desired to provide a fast and flexible Input/Output (I/O) system to process such a large amount of data. In order to improve the bandwidth of the external network of an MPP, we utilize multiple network interfaces that are based on commodity network like 100Base-TX Ethernet. Building a parallel I/O system on that facility, it is possible to reduce the cost and to get better performance. We also implement a dynamic load balancing technique based on a cooperated work among distributed and shared memory parallel processors. As a result, the parallel I/O system could offer users fast and flexible I/O system.

1. はじめに

超並列計算機（Massively Parallel Processor: MPP）における科学技術計算の過程で生じる，大量のデータに対する入出力処理の高速化は，超並列計算機をより利用しやすくするための重要な技術的要件である．たとえば，超並列計算機の外部環境として存在する，ビジュアリゼーション・サーバやファイル・サーバ等との間で実時間的にデータ処理することを考慮すると，現在の入出力機構では能力不足といえる．そこで，その解決策の 1 つとして，ネットワークを多重化することにより総バンド幅を増強するという方法が考えられる．この並列入出力に関しては，これまでもいくつかの研究がなされている[1)~8)]．

BEOWULF プロジェクト[1)]では，PC ベースの科学技術計算向け並列ワークステーションを作成しており，ノード間を複数のイーサネットで結合することによりデータ転送の高速化を図っている．

Sang[2)]は，ワークステーション間を並列ネットワークで接続した場合の性能に関して研究している．BEOWULF プロジェクトが同種 PC，ネットワーク媒体を用いているのに対し，文献 2) では異種なマシン，ネットワーク媒体でもマルチスレッドを用いて負荷分散することにより，効率良く並列ネットワークの

† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
☆ 現在，農業生物資源研究所
Presently with National Institute of Agrobiological Resources
†† 筑波大学計算物理学研究センター
Center for Computational Physics, University of Tsukuba

性能を引き出せることが報告されている．

HPSS[3]はディスク，テープ・デバイス等の Storage system を含めた統合的な並列入出力システムで，デバイス–マシン間でのファイル転送の高速化を図っている．HPSS の特徴の 1 つとして，並列ネットワーク上でペタバイトクラスのデータ転送を行うためのプロトコルとして提案された PTP[4]を用いていることがあげられる．

また，分散メモリ型並列計算機における並列入出力システムとして，文献 5) がある．このシステムでは，OS の拡張，新規関数の追加を行うことにより並列ネットワークを活かしたデータ転送を実現している．ただし，クライアント側は単一プロセスによってのみ動作している．

分散メモリシステムでの標準的なメッセージパッシングライブラリである MPI-2[6]にも並列入出力の機能（MPI-IO）が盛り込まれている．MPI-IO は，ファイル入出力に関してもメッセージパッシング・モデルをあてはめることができるという思想のもと，複数プロセスが共有ファイルをアクセスするための効率的かつ柔軟性の高いインタフェースの提供を目指している．MPI-IO の実装例としては上記の HPSS ほか，PMPIO[7]，ROMIO[8]等があげられる．

我々は，超並列計算機 CP-PACS[9]と共有メモリ型並列ワークステーションである Origin-2000 および Onyx2 を用いて，高性能かつ柔軟で，しかも安価な超並列計算機向け入出力システムの開発を目指している．特に並列ネットワーク，および MPP がかかえている多数の I/O プロセッサを有効利用することを考え，MPP のデータ生成能力に見合った外部との入出力環境を構築する．上述した他の並列入出力に関する研究の多くは，ファイル転送の高速化に主眼が置かれている．つまり，データ生成システム（超並列計算機）によって生成されたデータは，いったん内蔵ディスク等にファイルとして格納されていることを想定しており，この点がシステムにおけるデータ逐次処理を生み出し，ボトルネックとなる可能性が高い．そこで，本システムではこのような逐次化を排除するために，アプリケーション・プログラムから直接，データを並列入出力チャネルを用いて外部環境に送出し，このような逐次化フェーズをなくす．つまり，入出力を行う両端がともに並列計算機上で走っている並列アプリケーションであることを想定している．また，静的なファイル転送においては，データ量があらかじめ分かっているため，後述する負荷分散問題が比較的容易であるが，アプリケーションから直接データ送出する場合は，各プロセスから送出されるデータ量は変動するため，チャネルの負荷に不均衡が生じやすい．よって，並列入出力システムには動的な負荷分散機構は必須であり，本研究においても動的負荷分散機能について検討する．

本稿では，現在我々が開発を進めている並列入出力システムの実装と性能評価について述べる．

2. 背　　景

過去の超並列計算機プロジェクトにおいては，多くの場合，入出力系のために専用ハードウェア・ソフトウェアが導入・開発されてきた[10),11)]．これに対し，現在の汎用のバスあるいはネットワーク技術は，これらの研究において目標とされてきた水準に十分達しており，今後の超並列計算機用入出力系としては，むしろそのような汎用の製品（commodity）をハードウェア面で積極的に利用していき，ソフトウェア的な手法でその利用効率を高める方向で専用化を進めるほうが開発期間・コストに対する効率の点で有利であると考えられる．本研究ではこのような指針に基づき，コモディティ・ベースのハードウェアを利用していく．特に，100 Mbit/秒程度の通信性能を持つ 100Base-TX イーサネットや ATM は，その汎用性と簡便さからパーソナルユースのレベルにまで普及しているため，コストパフォーマンスの点で上記ネットワーク技術を大きく上回っているのが現状である．また，イーサネットに関しては Gigabit クラスが実用化されており，そのコストも急激に低下しつつある．

その一方で，多くの分散メモリ型超並列計算機では，入出力処理のためのインタフェースを持つ専用プロセッサが多数用意されており，大容量の内蔵ディスクはもちろん，多数の外部入出力チャネルをサポートすることが可能になっている．また，外部環境として存在する，並列ワークステーションのような他の計算機資源においても，その計算機性能に応じた並列なネットワーク環境が提供されつつある．したがって，超並列計算機とこれらのシステムを 1 対 1 に結合する際に，並列ネットワークを用いて多数のデータ流の同時転送を行うことが可能である．

以上の環境を統合することにより，超並列計算機上で生成された複数のデータ流は，並列ネットワークを介して，途中で逐次化されることなく並列のまま外部マシンに送ることが可能となり，高性能で安価な並列ネットワーク環境を構築することができる．しかし，このような並列ネットワーク環境下でのアプリケーション開発は，効率良く並列ネットワークを利用するために相手側のアプリケーションやチャネル数等を考

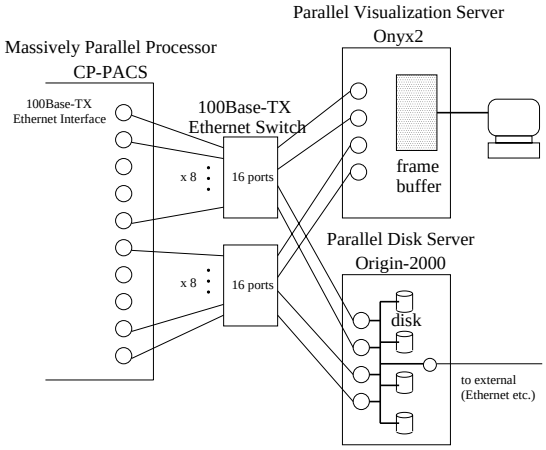

図 1 並列入出力実験システム
Fig. 1 Experimental environment for parallel I/O system.

慮し，最適化されるようにプログラミングしなければならず，ユーザにとって大変な重荷となる．そこで，このような複雑なシステム情報を内部で吸収し，ユーザからは簡便なアプリケーション・インタフェースによって利用可能な入出力システムの開発が望まれる．

3. ハードウェア構成

上述した入出力システムを実現するために，**図 1** に示すような環境を構築し，その上に並列入出力プログラミングを行うための並列入出力システムを設計・実装する．

図中の超並列計算機 CP-PACS [9]ではデータ生成が行われ，そのデータがフロントエンドマシン Origin-2000 のディスクにファイルとして格納されたり，または Onyx2 上で可視化処理が施されて frame buffer に書き込まれた結果を実際にディスプレイを通して見たりすることができるようになる．

CP-PACS は演算プロセッサが 2048 台，そしてそれらの入出力を処理するプロセッサが 128 台，計 2176 プロセッサから成る．また，入出力プロセッサのうち 16 台については，100Base-TX イーサネット・インタフェースが実装されている．図中の○が 100Base-TX イーサネット・インタフェースを表す．これら複数のチャネルを 2 台の 100Base-TX スイッチに等分して接続し，並列ディスク・サーバである Origin-2000（8 プロセッサ）および並列ビジュアリゼーション・サーバである Onyx2（4 プロセッサ ＋ 1 ラスタマネージャ）と結合している．これらのワークステーションにも，各々 4 ポートの 100Base-TX イーサネット・インタフェースが装備されており，各入出力チャネルは 2 つずつに分けて 2 台の 100Base-TX スイッチに接続されている．この結果，各 100Base-TX スイッチは CP-PACS 側からのリンク 8 本とフロントエンドマシン側からのリンク 4 本が接続され，16 ポートのうち 12 ポートが使用されていることになる．

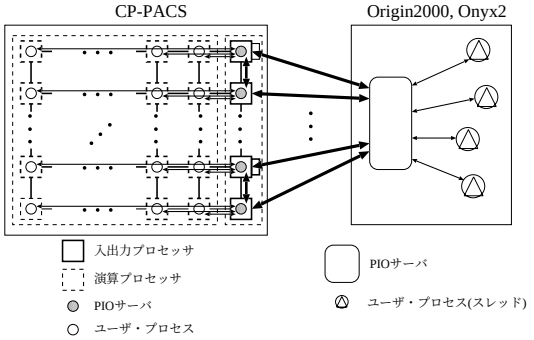

図 2 並列入出力システムの構成
Fig. 2 Process configuration of parallel I/O system.

なお，本稿ではチャネルとは各マシンの概念的な通信の端点を意味し，具体的には 100Base-TX インタフェースのことを指す．また，以降の説明で出てくるコネクションとは，両端のマシンのチャネルを結んだリンクのことを指す．実際には，図 1 に示すように 100Base-TX スイッチを介して接続されている．

4. 並列入出力システム

現在我々が開発を進めている並列入出力システム（“PIO システム” と呼ぶ）の構成を，**図 2** に示す．本システムは，マシン間でのデータ入出力を行うサーバ・プロセス（PIO サーバ）と，ユーザ・プロセスと PIO サーバ間との通信を行うための API（Application Program Interface）から成る．

本章では，まず PIO システムにおけるデータ転送方式および各チャネルの負荷分散方式について説明した後，PIO サーバ，API の特徴について述べる．

4.1 データ転送方式

PIO システムでは，データ転送は必ず PIO サーバを介して行う．マシン間（PIO サーバ間）でのデータの送受信は，**図 3** のように 3-way 転送方式によって行われる．PIO サーバ間で確立されたコネクションごとに固定サイズのバッファを割り当てれば，データの送信とそれに対する Ack という 2-way 方式を採用することも可能である．しかし，我々は膨大な量のデータ転送を行うアプリケーションを想定しているので，なるべく有効にバッファを利用したい．そこで単一のバッファを用意し，それを共有することにする．この手法によりプロトコル・オーバヘッドが多少増大するが，大量データの転送を目的とした場合，その影響は

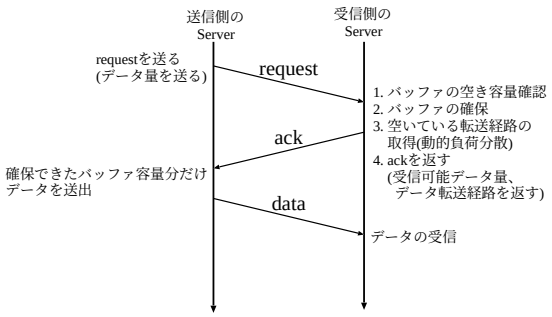

図 3 マシン間でのデータ転送プロトコル
Fig. 3 Data transfer protocol between machines.

少ない．また，3-way 転送方式にしたことにより，後述の動的負荷分散方式を効率的に実装できる．

マシン間のデータ転送は，TCP/IP を用いる．したがって，チャネルあたりの最大性能は，100Base-TX インタフェースの TCP/IP 性能に等しいか，もしくはそれ以下ということになる．しかし，TCP/IP という一般的な通信プロトコルを用いることにより，本システムに可搬性を持たせ，他プラットフォームへの移植が容易となる．

両端のマシン上でそれぞれ稼働しているアプリケーションのユーザ・プロセス間では，以下のようにしてデータが交換される．

一方のユーザ・プロセスから送出されたデータは，いったんそのマシンの PIO サーバに送られる．その PIO サーバでは，データに付加されたヘッダ情報に基づき，通信相手となるマシン上で走っている PIO サーバに対してデータを転送する．そして最終的に，他方のユーザ・プロセスにデータが送られる．ここで，各マシン内でいったん，PIO サーバによるデータのバッファリングを行うため，一般の TCP/IP 通信と比べてレイテンシの増加が余儀なくされる．しかし，各マシンに PIO サーバを置くことは，並列ネットワークの性能を引き出すためには必須であり，それぞれのユーザ・プロセス間で独自に TCP/IP コネクションを確立するよりも全体的な性能は向上するものと期待できる（詳細については，4.3 節にて述べる）．

また，少しでも通信のレイテンシを抑えるために，各マシン内部での PIO サーバとユーザ・プロセス間のデータ通信には「ゼロコピー通信機能」を利用する．ここでいうゼロコピー通信とは，通信を行う 2 プロセス間でデータのバッファリングを行うことなく，直接相手側のメモリにデータを書き込む通信のことを指す．超並列計算機 CP-PACS においては，ユーザ・レベルでのゼロコピー通信，すなわち，システムが提供するバッファを介することなしに，直接データの転送を行うことが可能である [9]．CP-PACS 側の PIO システムでは，この機能を利用し，演算プロセッサ（ユーザ・プロセス）と入出力プロセッサ（PIO サーバ）の間の通信において余計なデータ・コピーを排除することにより，処理の高速化を図る．また，Origin-2000 および Onyx2 上においては，共有メモリ型ワークステーションの特性を利用し，PIO サーバとユーザ・プロセスの間で共有メモリ空間を用いることにより，同様に余計なデータ・コピーを排除する．

4.2 負荷分散方式

本システムでは，アプリケーションに応じて適当な負荷分散方式を適用できるよう，動的負荷分散を含めた以下の 3 方式を提供する．

(1) ユーザ指定による負荷分散

これは，ユーザが直接プログラム内で使用するチャネルを指定する方式である．本方式は，故意にチャネルの負荷バランスを変えたい場合に有効である．たとえば本システムを利用したアプリケーションを複数走らせていた場合，優先度の高いアプリケーションにより多くのチャネルを割り当てるといった使用法が可能である．

(2) システムによる静的負荷分散

PIO サーバによって各ユーザ・プロセスが使用するチャネルを一意に決める方式である．MPP の一般的な利用方法は，問題領域の空間分割法等に代表される SPMD 的な処理を行うアプリケーションが多いものと考えられる．そのようなアプリケーションでは，各ユーザ・プロセスから送出されるデータ量はほぼ等しく，また適度にユーザ・プロセス間で同期をとっているためデータが送出されるタイミングも揃う．したがって，システム側であらかじめ各ユーザ・プロセスごとに使用するデータ転送経路を一意に決定しておいても自然に負荷分散されることが期待できる．

(3) システムによる動的負荷分散

PIO サーバがそのときの通信負荷状況から判断して最適なチャネルを選択する方式である．この方式は，各ユーザ・プロセスから送られるデータ量がまったく異なったり，予測不可能なアプリケーション，たとえばマスター・スレーブ方式のアプリケーション等には非常に有効である．また，計算負荷の分散を動的に行うアプリケーションにおいても，各プロセスが処理するデータ量は刻々と変化するため，本方式を用いるのが適当であると考えられる．

静的なファイル転送の場合は，あらかじめ転送量が

分かるため，チャネル数に応じてデータを等分し転送することで，均等な負荷分散が可能である．しかしながら，我々が開発しているシステムは，アプリケーションから直接データが転送されることを想定している．アプリケーションによっては，各ユーザ・プロセスから送出されるデータ流に空間的および時間的粗密が生じ，各チャネルにかかる負荷に不均衡が生じてしまうため，動的な負荷分散の実装は必須である．

現存する MPP システムの主流は CP-PACS のように分散メモリ型である．分散メモリ型の並列計算機では，各入出力サーバが独自に転送データを管理しているため，動的に負荷を分散するためには何らかの協調動作をとる必要がある．1 つの方法としては，一定周期ごとにすべての入出力サーバ間でチャネルの負荷および各サーバがかかえる転送データの情報を交換する方法がある．この場合，一定周期ごとに同期をとって負荷情報を交換する必要があるため，入出力サーバ数の増加にともない同期のオーバヘッドも大きくなってしまう．また，負荷情報を交換中にユーザ・プロセスから後続データが送られてくると，またそこで負荷バランスが変わってしまう．よって，この方法の場合は同期のタイミングを慎重に設定しなくてはならない．別の方法としては，特定ノード（たとえば複数ある入出力サーバの中の 1 つ）で負荷情報を集中管理する方法がある．負荷情報が 1 カ所に集中するため，全データ転送を均等に複数チャネルに割り振ることができる．ただし，相手マシンにデータを送る前に，上記ノードに使用すべきチャネルを問い合わせる必要がある．

一方，共有メモリ型の計算機で並列ネットワークを利用する場合は，共有メモリを用いることにより，各チャネルの負荷情報を一元管理することができ，また余計な通信は生じない．したがって，共有メモリ型のマシンのほうが分散メモリ型のマシンよりも負荷分散の問題を容易に解決できる．

本研究では，分散メモリ型の超並列計算機（CP-PACS）の外部マシンとして共有メモリ型のワークステーション（Origin-2000，Onyx2）を置くことを想定している．さらに，4.1 節で説明したように，マシン間では 3-way 転送プロトコル（図 3）を用いてデータを転送している．したがって，CP-PACS 側からのデータ送信時にはこのプロトコルに負荷情報を追加し，そして共有メモリ型のマシン上で分散メモリ型マシンの通信負荷についても一元管理することで，余計な通信コストをかけずに負荷を分散することが可能である．そこで，我々はこの方針に従い，動的負荷分散機能の実装を行う．つまり，CP-PACS 側からのデータ送出

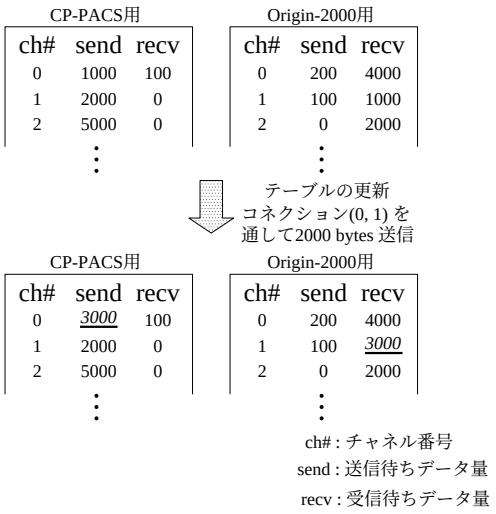

図 4 負荷分散情報記録用テーブル
Fig. 4 Table to record channel-load.

に関する負荷分散に関しても，相手側の Origin-2000（Onyx2）の PIO サーバが管理するようにする．

本システムは並列ネットワークを利用しているので CP-PACS と Origin-2000，Onyx2 間には複数のコネクションが張られており，動的負荷分散を行う場合はこれらのコネクションの中で最も負荷が軽いものを選択すればよい．そこで，各コネクションの両端のチャネルにかかる負荷情報を登録しておくテーブルを用意する．上述したとおり，CP-PACS 側から届くデータの情報についても Origin-2000（Onyx2）側 PIO サーバのテーブルに登録する．

このテーブルは，各マシンごとに用意され，それぞれのマシンに設置されたチャネル数だけのレコードを持つ．各レコードには，そのチャネルを通して送受信されるデータ量が記録される．たとえば，CP-PACS から Origin-2000 に対してコネクション (0, 1)（CP-PACS 側のチャネル 0 番と Origin-2000 側のチャネル 1 番を通した転送経路）を使用して 2000 bytes のデータが送られるとすると，**図 4** のようにテーブルの更新が行われる．実際には，データの送受信が行われようとしている PIO サーバ間で確立しているコネクションの中から，(送信元チャネルの送信待ちデータ量) ＋ (送信先チャネルの受信待ちデータ量) という値が最小のものを最も負荷の軽いコネクションとして選択し，テーブルの更新を行う．この負荷計算は，CP-PACS 側からデータ送信 request が届いたときにバッファ容量の確保と同時に行われ，コネクション選択結果は ack に含まれて CP-PACS 側 PIO サーバに返される（図 3）．そして，Origin-2000 側にデータが

届いた時点で，再度テーブルの更新が行われる（そのデータ転送に使用されたコネクションに対応する項目から，転送データ量分だけ減算する）．

なお，CP-PACS 側の複数の PIO サーバから同時に request が届く可能性があるが，テーブルの参照・更新は排他的に行われるため，特定のチャネルに対する一時的な負荷の集中もなく，つねに各チャネル間で均等な負荷の分散が行える．また，Origin-2000（Onyx2）側から送出されるデータに関しては，そのマシンの PIO サーバで負荷分散の管理をするため，マシン間でのデータ転送プロトコル中に負荷に関する情報は含まれない．

動的負荷分散を選択すればすべてのアプリケーションで均等な負荷分散を行うことができるが，転送経路を選択するために若干の遅延が生じるものと考えられる．したがって，アプリケーションに合わせて，適切な負荷分散方法を適用するべきである．

4.3 PIO サーバ

分散メモリ型並列計算機である CP-PACS では，各入出力プロセッサ上に PIO サーバを立ち上げ，PIO サーバ間で協調動作させることでシステムとしての一貫性を保つ．各 PIO サーバは，それぞれに割り当てられたユーザ・プロセス群のデータ送受信を仲介する．

一方，共有メモリ型の並列計算機である Origin-2000，Onyx2 では，並列入出力は 1 つの PIO サーバ・プロセスによって一元管理される．ただし，複数の 100Base-TX インタフェースおよびユーザ・プロセスの入出力を効率良く管理するために，PIO サーバ・プログラムはマルチスレッド化し，処理の多重化を図っている．

CP-PACS の各 PIO サーバと Origin-2000，Onyx2 の PIO サーバ間では，複数本のコネクションが確立されている．データ送信に使われるコネクションは，負荷分散方式に従って選択される．

PIO サーバの機能をまとめると，以下のようになる．

(1) 両端のマシンの PIO サーバ間におけるデータ転送すべての管理
(2) 余分なシステム情報の隠蔽
(3) 入出力データのバッファリング
(4) 複数チャネル間での負荷分散

図 2 のような環境のもとで，並列チャネルの存在を意識しつつ通信負荷の分散を行うことは，ユーザ・プログラムを繁雑化し，またシステム構成の変更等に対する適用性を損ねる恐れがある．たとえば，並列チャネル数の増減や，同時に実行されている他のユーザ・アプリケーションとの負荷バランス，あるいは接続先の相手計算機のチャネル数といった，データ転送負荷の制御に直接関係する要因をユーザ・プログラム側に意識させると，ユーザから見て非常に使い難いシステムになってしまう．さらに，大規模な超並列計算機では 10^3〜10^4 オーダものプロセスが通信に参加する．したがって，各々のユーザ・プロセスが相手側マシンのユーザ・プロセスと TCP/IP コネクションを確立したのではファイル・ディスクリプタ等のリソース不足に陥ることが容易に推測される．そこで本システムでは，各マシン上に稼働している PIO サーバがマシン間のデータ転送をすべて管理することで上記の問題を解消する．ユーザ・プロセスは，後述の簡便な API を用いて，PIO サーバとのみ交信する．これにより，上述したチャネル数や他のユーザ・アプリケーション等をユーザは考慮する必要がなくなり，プログラミングの負担が減る．

さらに，PIO サーバは適度なサイズのデータバッファを用意している．これにより，PIO サーバにデータがすべて送られた時点でユーザ・プロセスは送信処理から復帰することが可能である．また，複数チャネル間の負荷分散を図ることも，PIO サーバの重要な役割である．負荷分散については，4.2 節で述べたとおりである．

なお，本システムはオペレーティングシステムに依存しないよう完全にユーザレベルで実装しているため可搬性が高い．

4.4 並列入出力のための API

PIO システムを利用するための API（Application Program Interface）として，以下の機能を提供する．

- データ入出力操作
- システムで利用可能な並列入出力チャネルの構成要素情報の提供
- 接続先システムの情報の提供
- 手動または自動による通信負荷の均等化（データ生成プロセッサごとの負荷分散）

PIO システムの入出力関数は，UNIX 標準の read(), write() システムコールを拡張したものととらえることができる．ただし，read(), write() では通信の端点としてファイル・ディスクリプタを用いるが，本システムでは通信相手の（マシン ID，プロセス ID）という組によって指定される．つまり，通信の端点は個々のプロセスということになる．本システムは並列アプリケーション間での利用を想定しているので，ユーザは相手アプリケーションのどのプロセスとの間でデータ送受信するか明示する必要はある．しかし，通信の端点がプロセスであるため，データ転送経路上のネッ

表 1 PIO システム用関数群
Table 1 Functions in library of PIO system.

機能	関数名
PIO の初期化・終了	PIO_Init(), PIO_Exit()
データ入出力	PIO_Send(), PIO_Recv()
各種情報取得	PIO_Getappinfo() PIO_Gethostinfo() PIO_Getconinfo()
通信相手の変更	PIO_Setpartner()

トワークが並列であるかどうかはまったく意識せずにプログラミングできる．具体的には，PIO システムの送信関数では，引数として与えられたアドレスから，上述の組によって指定されたプロセスに向けてデータが送出される．一方，受信側では，指定したプロセスから送られてきたデータを受信関数の引数として与えたアドレスに格納する．

また，本 API ではシステム構成のうち，最低限の情報のみをユーザに提供，あるいは操作させ，大部分を自動化することを目的としている．明示的な情報操作をしない限り，ユーザプログラムからの入出力要求は，並列チャネルのいずれかに自動的に割り振られる．ユーザ・プログラム上で明示的な負荷分散を行いたい場合は，両マシン間に張られているコネクション情報等を引き出すことにより，使用するチャネルを指定することも可能である．そのうえ，両者の方法を混在することも可能である．

さらに，基本的にどの計算機上でも同一な API が利用できるように設計されている．通常，PIO システムを利用する 2 台以上の計算機システム上では，双方の端点における PIO プログラミングが必要となる．API が統一化されていることにより，このプログラミングは非常に容易となり，また作成されたプログラムの可搬性も向上する．

表 1 に本システムが提供する API を示す．システムの初期化・終了そしてデータ入出力関数以外に，ユーザが通信の最適化を行いたい場合に必要となるであろう各種情報を取得するための関数を用意する．また，PIO システムを利用している最中に通信相手（ホスト）を変更することができるように，PIO_Setpartner() 関数を用意した．本関数は，たとえば MPP で生成されたデータをファイル・サーバに蓄えつつ，リアルタイムにビジュアリゼーション・サーバ上でデータの可視化を行うといった 3 者間での利用を可能にする．

なお，共有メモリ型並列計算機である Origin-2000, Onyx2 用のユーザ・アプリケーションは，一般的にマルチスレッド化されている可能性がある．したがって，Origin-2000，Onyx2 上の PIO システムはそのようなアプリケーションにも考慮して，スレッドセーフとなるように実装されている．

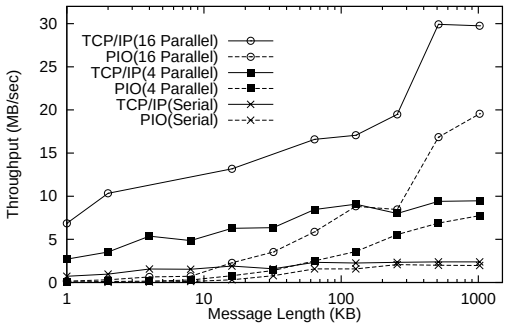

図 5 ping-pong 転送プログラムにおける PIO の性能
Fig. 5 Performance of ping-pong transfer.

5. PIO システムの性能評価

本章では，実装した PIO システムの性能評価を行う．実験システムとして，CP-PACS および Origin-2000 を用いる．ただし，5.4 節リモート・ファイル転送の評価に関しては，CP-PACS の代わりにそのプロトタイプ機である Pilot-3 を用いる．

Pilot-3 は CP-PACS とまったく同じアーキテクチャで，規模を縮小したマシンである．今回は，その中でも計算専用ノード 16 台，入出力用ノード 8 台，計 24 台のノードが使えるパーティションを使用する．なお，Pilot-3 は 4 つの 100Base-TX イーサネット・インタフェースを備えている．これら Pilot-3 側の 4 つのチャネルそして Origin-2000 が持つ 4 つのチャネルはすべて 1 台の 100Base-TX Switch に接続されている．

5.1 ping-pong 転送

PIO システムを用い，CP-PACS と Origin-2000 間で簡単な ping-pong 転送を行った場合の通信性能を **図 5** に示す．また，同図上にネイティブな TCP/IP 関数群をユーザアプリケーションから直接利用した場合の結果も載せる．それぞれ入出力チャネル数を 1，4，16 本と変えて測定を行った．ただし，Origin-2000 側は最大 4 チャネルなので，CP-PACS 側が 16 チャネルの場合は 16：4 という非対称な並列ネットワークとなる．なお，ユーザ・プロセス数も CP-PACS 側のチャネル数に応じて増やしている．また，単純なデータ転送しか行わないので，ユーザによる負荷分散方式を用いている．

図 5 より，PIO システムを用いた場合も TCP/IP 関数群を直接用いた場合もどちらもチャネル数の増加にともなった性能向上が得られていることが読みとれ

る．ただし，16 チャネルに関しては，ネットワークが非対称であるため必ずしも 16 並列に見合った性能向上は得られていない．また，Origin-2000 側において，ユーザプロセス数（16 プロセス）がプロセッサ数（8 プロセッサ）を超えていることも性能低下の要因と考えられる．特に，PIO システムを用いた ping-pong 転送では PIO サーバ（16 スレッド）も起動するため，その影響は大きい．

また，PIO システムを用いた ping-pong 転送は，TCP/IP 関数群を用いた場合より性能が低く，最大でも TCP/IP 関数群を用いた場合の 8 割の性能しか発揮していない．PIO システムでは PIO サーバを介してデータを送受信しているため，PIO サーバで一時的にバッファリングすることによるオーバヘッドが性能低下の主因であると考えられる．

さらに，TCP/IP 関数群を直接用いた場合であっても，各入出力チャネルあたりのスループットはたかだか 2.5 MB と，100Base-TX のピーク性能の 1/5 程度であることが分かる．これは，CP-PACS におけるオペレーティング・システムレベルで潜在的な性能低下となる要因があることを示唆している．したがって，100Base-TX イーサネット・インタフェースの性能を極限まで引き出すためには，オペレーティング・システムに含まれる TCP/IP プロトコル自体を改良する必要がある．

5.2 一方向転送

上記の ping-pong 転送実験のように，双方がデータを送信し合うというよりも，一方（データ生成系）から他方に多量のデータが送られることのほうが本システムにおいては一般的なケースであると考えられる．このとき，データ出力部分よりも演算部分がボトルネックとなるようなデータ生成系の場合，データ送信時間をできる限り短くして，早急に演算処理に復帰できるようにすることが望ましい．そこで次に，データ生成系（CP-PACS）の 1 プロセスからデータ処理系（Origin-2000）の 1 プロセスに対して一方向転送を行った場合の，送信関数を呼び出してから復帰するまでのレイテンシを求める．**図 6** は，1 チャネルだけを用いてデータ転送を行った場合の測定結果である．

図 6 より，先ほどの結果とはまったく異なり，どのメッセージ長においても PIO システムを用いた結果のほうが TCP/IP 関数群を用いた結果よりもレイテンシが小さい．1 MB のメッセージ転送にいたっては，TCP/IP 関数群を直接用いた場合の 1/100 以下の時間で完了している．これは，さきほどの ping-pong 転送のときは性能低下の主因となった，PIO サーバで

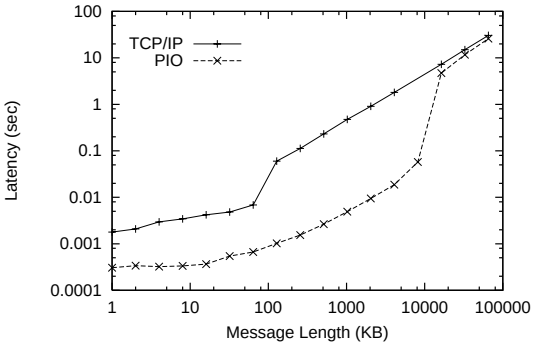

図 6 一方向転送プログラムにおける PIO の性能
Fig. 6 Latency of unidirectional transfer.

のバッファリングが有効に働いていると考えられる．つまり，PIO システムの場合は入出力プロセッサ上で走っている PIO サーバにデータが送られた時点で，ユーザ・プロセスは演算処理に復帰するからである．また，ユーザ・プロセス，PIO サーバ間のデータ転送はゼロコピー通信であるリモート DMA 転送[9]を用いているのでピーク 300 MB/sec というリンクあたりのスループットを最大限に利用できる．

一方，TCP/IP 関数群を用いた場合は，オペレーティング・システムが用意した通信バッファが溢れるとそこで通信バッファが空くまでブロックされる．実験した結果，CP-PACS のオペレーティング・システムが各ソケット用に確保できるバッファサイズは，送信用，受信用共最大で 63550 (Bytes) であった．したがって，このサイズ以上のデータを送る場合は，100Base-TX ネットワークのスループットがボトルネックとなる．さらに，TCP/IP 用のパケット作成にも多少の時間を要するため，メッセージ長が短い場合でもオペレーティング・システムの関与を極力排除したリモート DMA 転送を用いた PIO システムよりもレイテンシが大きい．

なお，PIO を利用した場合でも，PIO サーバのバッファが一杯になるとユーザ・プロセスはバッファが空くまでブロックしてしまう．つまり，サーバのバッファにたまっているデータがネットワークを介して外に送出されない限り，ユーザ・プロセスからの後続データを受け取ることができない．このような状況は，図中の 16 MB 以上のデータを転送した場合に現れてきている．また，多数のノードが通信に参加するほどバッファは溢れやすくなり，同様の状況に陥る．本システムは TCP/IP スタック上に実装されているので，PIO サーバから外にデータが送出されるスピードは “TCP/IP” とまったく同等である．したがって，図に示されるよ

うに，“PIO” の性能は “TCP/IP” とほぼ等しくなる．

5.3 負荷分散

続いて，各ユーザプロセスからの送出データ量が異なるアプリケーションを用いて，負荷分散方式による性能の違いを比較評価する．

本実験では，CP-PACS 上で走っている 256 プロセスから Origin-2000 上の 1 プロセスに対して一方的に総量 512 MB のデータ送信を行う．ここで，CP-PACS 上の各プロセスから送出されるデータの転送回数は等しいものの，1 転送あたりのデータ量はプロセスごとに異なるようにする．具体的には，静的な負荷分散を行った場合，CP-PACS，Origin-2000 間を結ぶ 16 本のコネクション（実際には，間に 2 台の 100Base-TX switch が設置してあるため，$2 \times (8 \times 2) = 32$ 通りのコネクションが考えられるが，ここでは CP-PACS 側のチャネルが重ならないような組合せ 16 通りのみを使用した）のうち，データ流量が最大のコネクションと最小のコネクションとでは 16 倍の差が生じるようにする．そこで，静的負荷分散の場合は各コネクションに流れるデータ量が 4.27 MB，17.07 MB，38.40 MB，68.27 MB と異なるようにし，同様の組合せを 4 組用意した．総データ量は，上述のとおり 512 MB である．動的負荷分散を行うことにより各コネクションに流れるデータ量は均等化され，つまり 512/16 = 32 MB のデータがそれぞれのコネクション上を流れる．静的負荷分散の場合は，データ流量が最大となるコネクションがボトルネックとなるので，この結果，動的負荷分散は静的負荷分散の 68.27 MB/32 MB= 2.13 倍の性能が得られるものと推測される．

図 7 に測定結果を示す．横軸は平均メッセージ長，縦軸は全メッセージ転送が終了するまでに要した時間である．各折れ線は，ユーザによる負荷分散（**USER**），システムによる静的負荷分散（**STATIC**），システムによる動的負荷分散（**DYNAMIC**）の結果である．ここで USER とは，ユーザが最適なデータ転送パターンになるようにハンドチューニングしたプログラムを用いたものであり，こちらも理論的には STATIC の 2.13 倍の性能が得られるはずである．

実測（図 7）では，動的負荷分散（DYNAMIC）は静的負荷分散（STATIC）よりも 1.75〜2.60 倍速いという結果が得られた．理論値 2.13 倍よりも高速になっているのは，STATIC ではネットワーク上での競合によるウェイトが生じているため，STATIC の実測値が同理論値を下回っていることが原因と推察される．また，DYNAMIC は USER よりも数%遅いが，これは DYNAMIC には動的にデータ転送経路を選択することによるオーバヘッドが加わることが原因であると推測される．ただし，動的負荷分散処理（データ転送経路の選択）を行う Origin-2000 側の PIO サーバはマルチスレッド化されているため，ある送信データの動的負荷分散処理と，先行して送出された別データの受信はそれぞれ異なるスレッドを割り当てることで同時に処理することができる．よって，処理のオーバラップがうまくいかない全データ転送の開始および終了間際のオーバヘッドのみが性能差として現れていると考えられる．また，このことはメッセージ長が異なる場合でも DYNAMIC と USER との性能差がほぼ一定である理由でもある．

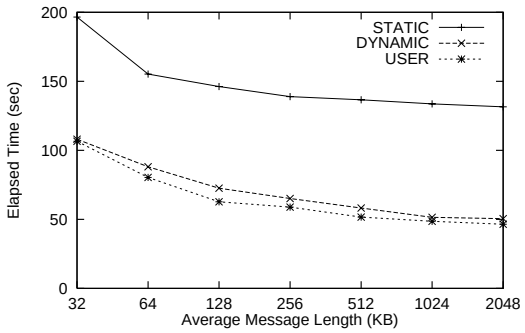

図 7 負荷分散方式の性能比較
Fig. 7 Performance comparison of three load-balancing methods.

さらに，全体的な性能をみるとたかだか 10 MB/sec であることが分かる．5.1 節の pingpong 転送では PIO システムを利用して 16 チャネルで 20 MB/sec の性能が出ているのに対し，その半分の性能しか発揮できていない．本システムでは，各ユーザプロセスがデータを送る PIO サーバは一意に決められているため，本実験で使用したアプリケーションのように各 PIO サーバから送出されるデータ量が大幅に異なってしまうケースが出てくる．現在の PIO システムでは，並列ネットワークの各コネクションの負荷を分散するための機能は実装されているものの，同一マシン上に起動している PIO サーバ間でデータ流量の分散は行っていない．したがって，この問題はどの負荷分散方式をとっていても起こりうる．つまり，最もデータ流量の多い PIO サーバがボトルネックとなり，図 7 のように全体的に性能が低くなってしまったものと考えられる．PIO の次期 version では，PIO サーバ間で処理データ量の分散も行うようにする予定である．

本実験では，プロセスごとのデータ送出量の偏り具合が明白であったためユーザによる負荷分散方式が最も良い性能を示した．しかし，動的負荷分散機能を用い

表 2 リモート・ファイル転送（40 MB）に要する実行時間（NFS，rcp では 1 本，PIO では 4 本のチャネルを使用）
Table 2 Latency for remote transfer of 40 MB file (using 1 channel for NFS and rcp, 4 channels for PIO).

(1) Output data from an application program	
via NFS	69.39 sec (0.58 MB/sec)
via PIO	5.95 sec (6.72 MB/sec)
(2) Copy a file from Pilot-3 to Origin-2000	
using rcp	22.43 sec (1.78 MB/sec)
using PIO	7.34 sec (5.45 MB/sec)

た場合もそれとほぼ同等な性能が得られている．データ量が完全にランダムなアプリケーション，たとえば計算の負荷分散を動的に行うようなアプリケーションでは，各プロセスが扱うデータ量は予測不可能である．そのようなアプリケーションでは，動的負荷分散方式を選択することにより最小の実行時間で済むものと期待される．

5.4 リモート・ファイル転送

PIO システムを利用したより現実的なアプリケーションとして，リモート・ファイル転送の測定を行う．Pilot-3 側では，4 プロセスによって生成もしくは読み込まれたデータを Origin-2000 上で稼働しているファイル・サーバに転送する．Origin-2000 側のファイル・サーバは，Pilot-3 から送られてきた 4 つのデータ流を単一流にまとめ，そしてローカルのファイルシステムに 1 つのファイルとして書き込む処理を行う．

今回は 2 種類のファイル転送実験を行う．なお，転送するファイルのサイズは 40 MB である．まず最初に，ファイル・サーバに送るデータを Pilot-3 側のプログラム内で直接生成する場合である．つまり，データを読み込むためのディスクアクセスはいっさい発生しない．また，PIO を用いたファイル転送性能と比較するために，NFS を介したファイル転送の性能も測定する．NFS は version 3 を使用し，プロトコルとしては TCP を用いている．また，PIO を用いたファイル転送では，1 KB を転送単位としている．

結果は**表 2** (1) のようになった．これによると，PIO を用いたデータ転送は，NFS を利用した場合の約 9%の実行時間で済んでいる．これは，PIO が 4 本の 100Base-TX イーサネットを効率良く利用できているのに対し，NFS はそのうちの 1 本しか用いていないことが主因である．さらに，NFS の場合はリモートのファイルシステムをあたかもローカルのファイルシステムであるかのように見せるために，様々なソフトウェア・オーバヘッドが生じていることが本測定結果に影響しているものと考えられる．

そこで次の実験として，NFS よりもソフトウェア・オーバヘッドの小さい UNIX の “rcp” コマンドと PIO を利用したファイルコピー・プログラムの性能を比較する．対等な比較評価を行うために，PIO を用いたプログラムもファイルからデータを読み込むようにする．PIO を利用したプログラムでは，4 つのプロセスが同時に 1 つのファイルからデータを読み込み，そしてある程度のサイズに分割して 4 つのチャネルからデータを送るようにする．一方，“rcp” コマンドの場合は 1 プロセスがファイルからのデータ読み込み，チャネルへのデータ出力を行う．また，使用するチャネル数も 1 本のみである．

結果は表 2 (2) のようになり，PIO を利用したファイルコピー・プログラムは “rcp” コマンドの約 33%の転送時間を要した．今回の実験では，4 倍のチャネル数を利用しているにもかかわらず，それほど効果がでていないという結果になった．これは Pilot-3（Origin-2000）におけるファイル読み込み（書き込み）のところで逐次化されてしまうところがボトルネックとなり，それほど差が生じなかったのではないかと推測される．

6. PIO の実用例と今後の課題

現在，PIO システムを利用した実アプリケーションとして，PIO システムと同時並行して並列可視化システムの開発が進められている[12]．

従来のデータ可視化では，MPP 上で生成されたデータをいったんファイルとして保存し，それを外部のグラフィックワークステーションにファイル転送してから可視化するというのが一般的であった．それに対して，本可視化システムでは内部のデータ可視化処理を並列に行うことで高速化を図る一方，PIO システムを利用した MPP 上のアプリケーションからの並列データ入力をサポートしている．さらに，並列チャネルからデータを受け取る部分は，可視化処理プロセスとは別のプロセスとして実装しているため，並列チャネルからのデータ入力と可視化処理を並列に実行できるようになっている．このようにデータ生成，データ転送，データ可視化すべてを空間並列化，機能分割・並列化できており，その結果，データ生成と並行してリアルタイムに中間データ・最終データを連続的に可視化することが可能となった．

PIO システムを利用することにより，MPP でのデータ生成から外部並列システム環境でのデータ処理に至るすべてを継ぎ目なく並列化することが可能となる．これにより効率の良いデータ処理が可能となり，また本システムはそのようなアプリケーション開発を促進するものと期待される．

我々は，上記の並列可視化システム以外にも並列ファイルシステム等の実用アプリケーションを充実させていく予定である．そして，それと同時に以下のような改善を今後行っていく必要があると考えている．

- TCP/IP は高い可搬性を持つが，マシンによってはプロトコルネックで性能が低下する場合もある．たとえば，CP-PACS は MPP として高速プロトコルでのプロセッサ間通信は非常に高速だが外部ネットワークを介した絶対通信性能が低いことが本測定からも明らかになった．したがって，必要に応じてある程度システムに手を入れる必要がある．
- 今回，ネットワーク通信レベルでのプロトコルには，本システムの各種プラットフォームへの可搬性を期待して TCP/IP を用いたが，ユーザレベルの API には HPC ユーザが必要最低限の情報をシステムに与えつつ最大の効果を得られるよう，我々独自の PIO-API を設計・実装した．しかし今後，ユーザレベルでの可搬性を考慮した場合，PIO-API だけでなく，より一般的な API，たとえば MPI-IO 等を対象とした並列入出力システムの実装も検討すべきである．
- 可搬性とコストパフォーマンスの点から現在は 100Base さらに Gigabit イーサネットを考えているが，今後の展開として最近注目されている SAN（System/Storage Area Network）に対してこのようなシステムを適用していくことも大事である．

現在の実装・性能評価では実用システムとしての絶対性能よりも，むしろ本システムの拡張性，有効性を検証している．しかし，上述したような性能チューニングを施すことにより 1 チャネルの通信が十分高速になれば，非常にコストパフォーマンスの高いシステムが実現できることを確信している．

7. おわりに

本研究では，現在我々が開発を進めている超並列計算機向けの並列入出力システムを超並列計算機 CP-PACS，そしてその外部マシンである SGI Origin-2000，Onyx2 に実装し，その性能評価を行った．

ネットワークを並列化することにより，両端のマシンに複数ある入出力プロセッサを効率良く利用でき，その結果，複数データ流をいったん逐次化することなく，並列のまま相手側マシンに送ることが可能となった．また，近年高性能化が進んでいるコモディティ化されたネットワークを用いることで，安価ではあるが高速な環境を実現できる．我々は，このようなハードウェア環境下で，並列ネットワークを効率良く利用するための並列入出力システムを開発した．ユーザからは，簡便なインタフェースを利用することで，並列ネットワークの恩恵を享受できる．本システムの性能を評価した結果，入出力を並列化することにより，容易に線形な性能向上が見込めることが確認された．

また，本システムでは，3 つの負荷分散方式を提供している．実験結果から，負荷が局所的に集中してしまうようなアプリケーションであっても，動的負荷分散を行うことにより，負荷の不均衡が解消されることが確認された．

現在のところ並列入出力システムの基盤がほぼ仕上がった段階であり，その有効性については本研究で実証できた．しかしながら，実用性を考えるならば絶対性能の改善，アプリケーション開発はもちろんのこと，標準 API への適用，プラットフォームの拡大等を検討する必要がある．

謝辞 本研究を行うにあたり，ご意見・ご協力いただいた筑波大学計算物理学研究センター未来開拓プロジェクト関係各位に感謝いたします．なお，本研究の一部は日本学術振興会未来開拓学術研究推進事業「計算科学」（Project No.JSPS-RFTF 97P01102）によるものである．

参考文献

1) Sterling, T., Becker, D., Savarese, D, et al.: BEOWULF: A Parallel Workstation for Scientific Computation, *Proc. International Conference on Parallel Processing*, Vol.1, pp.11–14 (1995).
2) Sang, J.: Multithreaded Data Transfer over Parallel Links, *Proc. PDPTA'99 International Conference*, Vol.5, pp.2403–2409 (1999).
3) Watson, R.W. and Coyne, R.A.: The Parallel I/O Architecture of the High-Performance Storage System (HPSS), *Proc. 1995 IEEE MSS Symposium* (1995).
4) Berdahl, L.: *Parallel Transport Protocol Proposal*, Lawrence Livermore National Laboratory (1995).
5) 北井克佳，吉澤 聡，マシエルフレデリコ，鍵政豊彦，稲上泰弘：TCP/IP 通信を用いた並列ネットワーキング方式の検討，情報処理学会論文誌，Vol.39, No.11, pp.3044–3053 (1998).
6) *MPI-2: Extensions to the Message-Passing Interface.* `http://www.mpi-forum.org/`.
7) `http://parallel.nas.nasa.gov/MPI-IO/pmpio/pmpio.html`.
8) Thakur, R., Gropp, W. and Lusk, E.: Data

Sieving and Collective I/O in ROMIO, *Proc. 7th Symposium on the Frontiers of Massively Parallel Computation*, pp.182–189 (1999).
9) 岩崎洋一，中澤喜三郎ほか：計算物理学と超並列計算機—CP-PACS 計画，情報処理，Vol.37, No.1, pp.10–42 (1996).
10) 廣野英雄，松岡浩司，岡本一晃，横田隆史，坂井修一：RWC-1 の入出力機構と基本性能，情報処理学会研究報告，ARC119-34, pp.197–202 (1996).
11) 吉山 晃，中野智行，中條拓伯，金田悠紀夫：超並列計算機 JUMP-1 の入出力システムの実装と評価，情報処理学会研究報告，ARC119-35, pp.203–208 (1996).
12) 沼 寿隆，松原正純，板倉憲一，朴 泰祐：並列入出力機構を用いた可視化システムの提案，情報処理学会研究報告，HPC77-10, pp.53–58 (1999).

(平成 12 年 2 月 7 日受付)
(平成 12 年 6 月 2 日採録)

松原 正純（学生会員）

昭和 48 年生．平成 8 年筑波大学第三学群情報学類卒業．平成 10 年同大学大学院工学研究科前期博士課程修了．現在，同研究科後期博士課程在学中．並列入出力に関する研究に従事．

沼 寿隆

昭和 50 年生．平成 10 年筑波大学第三学群情報学類卒業．平成 12 年同大学大学院理工学研究科修士課程修了．現在，農林水産省農業生物資源研究所にてイネゲノムの研究に従事．

板倉 憲一（正会員）

昭和 44 年生．平成 5 年筑波大学第三学群情報学類卒業．平成 11 年同大学大学院工学研究科博士課程修了．博士（工学）．平成 11 年筑波大学計算物理学研究センターリサーチ・アソシエイト．並列計算機アーキテクチャ，並列入出力・可視化機構の研究に従事．

朴 泰祐（正会員）

昭和 59 年慶應義塾大学工学部電気工学科卒業．平成 2 年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和 63 年慶應義塾大学理工学部物理学科助手．平成 4 年筑波大学電子・情報工学系講師，平成 7 年同助教授，現在に至る．超並列処理ネットワーク，超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，並列処理システム性能評価の研究に従事．電子情報通信学会，日本応用数理学会，IEEE 各会員．