社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－HPC－107 (37)
2006／8／2

PACS-CS のための高性能通信ライブラリインターフェイスの設計

住元真司[†] 大江和一[†] 久門耕一[†]
高橋大介[††] 朴泰祐[††] 佐藤三久[††]
吉江友照[††] 宇川彰[††]

PACS-CS 上での主要アプリケーションではメモリバンド幅を節約しながら高い通信性能を実現することが重要であるため、コピーを利用しない通信機構の実現が重要である。本稿では、PACS-CS 上での主要アプリケーションにおいて、通信におけるプロセッサによるコピーオーバヘッドを削減するための通信ライブラリインターフェイスの設計に述べる。

A Design of High Performance Communication Library for the PACS-CS System

Shinji Sumimoto,[†] Kazuichi Ooe,[†] Kouichi Kumon,[†]
Daisuke Takahashi,[††] Taisuke Boku,[††] Mitsuhisa Sato,[††]
Tomoteru Yoshie[††] and Akira Ukawa [††]

In some of the main target applications on the PACS-CS system, it it very important to realize a high performance communication which does not require processor memory copy processing because the applications require much of memory bandwidth.

This paper discusses a design of high performance communication library interface to eliminate processor copy overhead on the PACS-CS system.

1. はじめに

筑波大の次期スーパーコンピュータである PACS-CS[1] システムは Gigabit Ethernet を 6 系統用いた 3 次元 Hyper Crossbar ネットワークを採用した 2560 ノードの PC クラスタである。PACS-CS のために、専用の高性能通信機構 PM/Ethernet-HXB[2] が開発され、SCore クラスタ システムソフトウェア[3] が利用されている。

我々は、現在、PACS-CS システム上でアプリケーションの高速化に取り込んでいる。PACS-CS での主要アプリケーションは格子 QCD と実空間-DFT(RS-DFT) であるが、一般の MPI アプリケーションも稼働する。格子 QCD と RS-DFT はそれぞれ通信パターンに特徴がある。格子 QCD は 3 次元 6 方向の隣接同時双方向転送、RS-DFT は全ノードを跨る連続的にデータサイズが変わる大規模 allreduce 計算に特徴がある。

† 富士通研究所
FUJITSU LABORATRIES
†† 筑波大学
University of Tsukuba

それぞれのアプリケーションは MPI で書かれており、これを評価したところ、格子 QCD については、MPI 層におけるメモリコピー処理時間がボトルネックとなり、実行性能向上が頭打ちになることがわかった。また、RS-DFT についても大規模 allreduce 演算を効率的に実行する必要があることがわかった。これを解決するため、コピー処理を最小限にする低レベルの通信 API とそれを実現する通信機構を PM/Ethernet-HXB の拡張 API として開発を進めている。

PM/Ethernet-HXB では既存の Ethernet ネットワークインターフェイスカード（NIC）を用いて高い通信性能を実現するため、受信データのバッファアドレスはメッセージを受信して始めてわかる構造となっている。この考え方を更に進めて、アプリケーションを含めて最もオーバヘッドが少ない通信の実現を目指している。

本稿では、PACS-CS 上でアプリケーションの高速化を実現するための API の設計について述べる。まず、各アプリケーションの動作と PMv2 API の動作について説明し、PMv2 API 上の MPI 処理におけるコピーがどのように行われているかを整理し、問題点をまとめる。次にこの問題点を解決する API の設計について述べる。

2. PACS-CS システムについて

PACS-CS[1] は、筑波大学で開発中の PC クラスタシステムで、3 次元 Hyper Crossbar ネットワークを持つ。システムの概要を表 1 に示す。

表 1 PACS-CS システムの概要

ノード計算機	Intel LV Xeon 2.8GHz(EM64T) (Intel E7520, 2GB DDR2 SDRAM 4 x 64bit 133MHz PCI-X Bus)
計算ネットワーク	3 次元 Hyper Crossbar Gigabit Ethernet(E1000)x6 JUMBO FRAME 利用
管理 IO ネットワーク	Gigabit Ethernet x2
ホスト OS クラスタ OS	Linux Fedora Core 3 for x86_64 SCore5.8.3[3]
ノード数	2,560 (16x16x10)

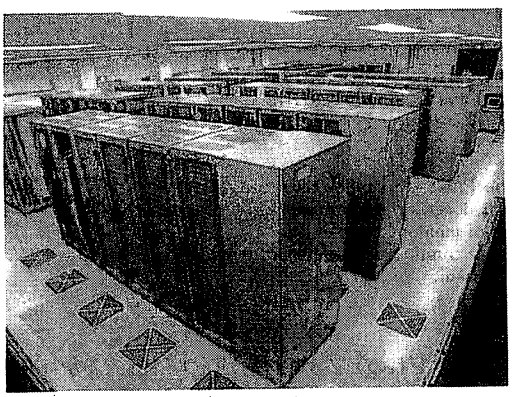

図 1 PACS-CS システム外観

PACS-CS の特徴は、計算処理と通信処理のメモリバンド幅バランスを考慮した単一プロセッサノードと 3 次元 Hyper Crossbar 結合 (16x16x10) を採用している点である。

PACS-CS システムでは、SCore クラスタシステムソフトウェアが採用され、3 次元 Hyper Crossbar 結合に対応した通信機構である PM/Ethernet-HXB[2] が開発され、利用されている。

3. PACS-CS システムでのターゲットアプリケーション

PACS-CS 上での主要アプリケーションとしては、素粒子物理学における格子 QCD 計算と物性物理学における RS-DFT 計算がある。これら以外にも一般の MPI アプリケーションが実行される。

格子 QCD プログラムの動作： 格子 QCD プログラムにおける計算は、全体のデータ領域を 3 次元に分割し、QCD 計算と 3 次元 Hyper Crossbar 結合上での 6 方向の隣接双方向通信によりデータ交換を繰り返しながら、計算を進めていく。

3 次元 HyperCrossbar 結合上での 6 方向の隣接双方向通信において、転送されるデータは、データ配列上飛び飛びの位置に存在するため、メモリバンド幅で処理性能が律速される。

RS-DFT のプログラムの動作： RS-DFT プログラムの主要処理は、共役勾配法 (CG) もしくは最小残差法 (RMM) によるハミルトニアンの固有値・固有ベクトルの算出処理とグラムシュミット直交化処理を組み合わせた反復計算を行なう。この計算過程で、連続的にサイズが変化する MPI_Allreduce 処理が多用される。計算処理自体に占める通信処理の割合は大きくないが、大規模なデータ転送となるために通信オーバヘッドの削減が重要である。

4. PACS-CS 上での MPI 通信の問題点と実現する通信機構の目標

本章では、PACS-CS 上で実現する通信機構の目標について述べるために、PMv2 通信機構と PMv2 API の動作と MPI 通信の問題点を整理し、この問題点を解決するための通信機構の目標について述べる。

4.1 PMv2 通信機構と PMv2 API の動作

PMv2[4),5)] は、SCore クラスタシステムソフトウェアの低レベル通信機構である。PMv2 は、ネットワークハードウェアを意識することなくシームレスに利用可能にするために、ネットワークデバイスハードウェアに依存しないアプリケーションプログラムインターフェイス (API) を規定している (PMv2 API[6])。

PMv2 においては、メッセージ通信と RMA 通信の 2 種類の通信形式をサポートしており、それぞれの通信において PMv2 API は次のように利用される。

- メッセージ通信：
  - メッセージ通信処理を高速に実行するために、送信バッファ受信バッファともに、ハードウェアの性能を引き出せるように独自に割り当てられるようになっている。このため、PMv2 のメッセージ通信の主要 API は、送信バッファの獲得 (pmGetSendBuffer)、送信 (pmSend)、受信メッセージの確認 (pmReceive)、受信メッセージの解放 (pmReleaseReceiveBuffer) で規定されている。本関数においては、送信バッファ、受信バッファ共に通信ライブラリにより、そのバッファアドレスが提示されるために、ユーザバッファとの間でコピーが必要になる点が問題である。
  - 送信データは PMv2 で転送可能な最大 MTU サイズ毎に転送する必要がある。送信バッファ

不足時はポーリングでバッファ空きを待つ必要があるため、転送相手がデータを受信しないと、転送効率が著しく低下する場合がある。
  - 大規模データ送信時には MTU サイズ毎に pmGetsendbuffer, pmSend を実行する必要があるため、送信バッファ枯渇時のバッファの空き待ちと、関数処理オーバヘッドが問題となる。
- 遠隔メモリアクセス（RMA）通信：
  - RMA 通信は、Myrinet や InfiniBand など、RDMA 通信を持っているハードウェアの利用を前提に提供されている通信である。RMA 通信においては、各ノードのメモリは、pmAddrHandle と呼ばれる構造体で管理されている。そして、この pmAddrHandle を生成する関数として pmMLock があり、同時に pindown を実行する機能を持つ。この pmAddrHandle を相互のノードで交換して pmWrite, pmRead 関数を用いて RMA 通信を実行する。
  - RMA 通信は、ハードウェアサポートを持っていないネットワークについてもエミュレーション的にソフトウェアで実装することが可能であり、例えば、PM/Ethernet[7),8)] では、RMA 通信をデバイスドライバから直接 Read/Write することにより実現している。ハードウェアサポートがない場合には、エミュレーションのため、コピーが必要になる。
  - RMA 通信では、実際の通信を行なうために、pmAddrHandle の交換が必要であるために、この交換のための遅延が通信性能に大きく影響する。予め交換して再利用を徹底するなどの効率化が必要である。

4.2 PACS-CS 上での MPI 通信の問題点

第 4.1 節で述べた PMv2 の持つ API 上で実装された MPI は、メッセージ通信を主体とした実装となっており、RMA を用いる場合には実行時に mpi_zerocopy_on というオプションを用いることに実行可能になっている。しかし、PACS-CS で採用しているネットワークハードウェアでは RMA をハードウェアでサポートする機構をもっていないため、MPI を用いると必然的にデータのコピーが必要となる。

このデータコピーのオーバヘッドを PACS-CS 上での主要アプリケーションが使うと想定されているプロセッサーのキャッシュが有効に使えない環境下における、3 次元隣接双方向通信プログラムで測定した。その結果、PM レベルの通信性能では 3 次元隣接双方向通信で、1403 MB/s の通信性能であったが、MPI を利用した場合には、630 MB/s 付近まで性能劣化した。この性能劣化の原因を調べたところコピー処理によるものであることがわかった。

また、MPI の collective 通信の実装は、例えば、MPICH のように MPI の point-to-point 通信をベースに実装されていることが多い。このため、MPI の collective 通信実行時には、MPI 内部の通信に閉じていても、プロセッサによるコピーが新たに生じる上、新たに MPI 内部でデータ領域の割り当てが発生する場合もあり、全体のメモリ利用量に影響を与える場合がある。

4.3 実現する通信機構の目標

第 4.1, 4.2 節での議論より、、プロセッサによるコピーオーバヘッドを最小限にすること、ならびに、複数回にわたり実行する処理を一括処理することにより、通信手続きのオーバヘッドの削減を実現すべき通信機構の目標として設定する。

5. 通信ライブラリインターフェイスの設計

本章では、第 4 で述べた目標を実現するための通信ライブラリインターフェイスの設計について述べる。ここでは、2 つの目標を実現するための実現手法とインターフェイスの設計を論ずる。

5.1 コピーオーバヘッドの削減

PMv2 のインターフェイスに対してコピーオーバヘッドを削減するためには、現状の API では許していないバッファの再利用を可能にする必要がある。このため、第 4.1 節のメッセージ通信のところで述べたバッファのアドレスをユーザが固定的に扱える必要がある他、データの転送が完了し再利用が可能かどうかを確認する手段が必要になる。

データの再利用が可能な利用法としては、次の 2 つの利用法が想定できる。

- 送信バッファ上に主要なデータを配置し修正しながら送信して再利用する：
  ある程度送信バッファ自体をデータ領域として利用して、非同期 I/O のように送信の完了を確認した後に送信バッファを更新し、再度送信する。
- 受信データのバッファをそのまま送信データとして送信する：
  例えば、MPI の broadcast などの collective 通信のように、同じデータをバイナリツリー形式などで転送する場合、あるいは受信データにそのまま何らかの処理を施して送信する場合に有効である。

以上の 2 つの利用法に対して適用可能なユーザインターフェイスを実現する。

5.2 通信手続きのオーバヘッドの削減

大規模なデータを送受信する場合に、MTU サイズ毎のメッセージをメッセージ数分送受信処理するのはオーバヘッドが大きい。しかし 市販の Gigabit Eth-

ernet では、MTU 単位でしかメッセージ通信ができない。かつ、前節のコピーオーバヘッドが必要ない手法で実現する必要がある。

これを実現するために、もっとも簡単な実現手法は、バッファ自体を複数持ち、これを一つのバッファとして扱うことである。

5.3 pmBufferArray 構造体の導入とユーザインターフェイス

第 5.1、5.2 節で議論した機能を効果的に実現するために、pmBufferArray 構造体を導入し、pmBufferArray 構造体単位でメッセージの送受信が可能なインターフェイスを実現する。

pmBufferArray 構造体の持つデータは、基本的に通信用バッファ配列へのポインタと制御を行なうための情報である。また、pmBufferArray 構造体の送信時には、送信用の ID を付与することができ、受信側では、送信時に付与された ID を元にメッセージの選択受信が可能となっている。

表 2 pmBufferArray 利用のための主要 API の概要

関数名	関数の説明
pmGetSendBufferArray	送信 pmBufferArray を一つ割り当てる
pmSendBufferArray	送信 pmBufferArray を一つ送信する
pmSendDoneBufferArray	指定した pmBufferArray の送信完了確認
pmReleaseSendBufferArray	指定した送信 pmBufferArray の解放
pmReceiveBufferArray	pmBufferArray の受信確認
pmReleaseReceiveBufferArray	受信 pmBufferArray の解放

表 2 に pmBufferArray 利用のための主要 API の概要について示す。表 2 の他にいくつかの補助関数が存在する。

6. 既存 PMv2 と pmBufferArray を用いた通信処理比較

本章では、簡単なプログラムを用いて、既存 PMv2 と pmBufferArray を用いた通信処理の比較を行なう。

6.1 既存 PMv2 を利用した pingpong の実現例

本節では既存 PMv2 の API を利用した pingpong の実現例の処理手順を示す。ここで取り扱う pingpong 処理は受信したデータをそのまま相手に返すプログラムである。

ping 処理の手続き

(1) pmGetSendBuffer により送信バッファを一つ割り当てる
(2) 送信バッファにデータをコピーする
(3) 以下の処理を繰り返す。
  (a) pmSend により送信先にデータを送信する
  (b) pmReceive によりデータが来るまで待つ
  (c) pmGetSendBuffer により送信バッファを一つ割り当てる
  (d) 受信データを送信バッファにデータをコピーする
  (e) pmReleaseReceiveBuffer により受信バッファを解放する

pong 処理の手続き

(1) 以下の処理を繰り返す。
  (a) pmReceive によりデータが来るまで待つ
  (b) pmGetSendBuffer により送信バッファを一つ割り当てる
  (c) 受信データを送信バッファにデータをコピーする
  (d) pmReleaseReceiveBuffer により受信バッファを解放する
  (e) pmSend により送信先にデータを送信する

以上の手順は、メッセージのサイズが MTU 以内の場合はそのまま使えるが、メッセージのサイズが MTU 以上の場合は、メッセージの送信と受信のそれぞれについてメッセージの数だけ処理を行なう必要がある。

また、受信バッファデータを送信バッファにコピーする処理が必ず入る。

6.2 pmBufferArray を利用した pingpong の実現例

本節では pmBufferArray を利用した pingpong の実現例の処理手順を示す。

ping 処理の手続き

(1) pmGetSendBufferArray により pmBufferArray を一つ割り当てる
(2) 送信バッファにデータをコピーする
(3) pmSendBufferArray により送信先にデータを送信する
(4) 以下の処理を繰り返す。
  (a) pmReceiveBufferArray によりデータが来るまで待つ
  (b) 受信した pmBufferArray を pmSendBufferArray により送信先にデータを送信する
  (c) pmReleaseReceiveBufferArray により一つ前に受信した pmBufferArray を解

放する

(5) pmReleaseSendBufferArrayにより送信用pmBufferArrayを解放する

pong処理の手続き

(1) 以下の処理を繰り返す。
  (a) pmReceiveBufferArrayによりデータが来るまで待つ
  (b) 受信したpmBufferArrayをpmSendBufferArrayにより送信先にデータを送信する
  (c) pmReleaseReceiveBufferArrayにより一つ前に受信したpmBufferArrayを解放する

ping pong処理の場合の特徴は受信したpmBufferArrayをそのまま送信用のpmBufferArrayとして使える点である。

以上の手順は、メッセージのサイズがMTU以上の場合についても同様に同じある。

6.3 pingpongの実現例の比較

第6.1節と第6.2節の例で比較する。

通信処理関数の呼出回数に関しては、PMv2での実装例では必ず送信バッファの割り当て関数を呼ぶ必要があるが、pmBufferArrayでの実装例では、ping側の初期化時に1回呼ぶだけで良い。また、PMv2による実装例では、メッセージサイズがMTU以内であれば、単純な実装で済むが、MTU以上の場合はメッセージの数の回数分だけ送受信処理を実行する必要があるが、pmBufferArrayでの実装例では、同じ実装で、つまり1回だけの実行で済むことになる。

また、ホストプロセッサのコピー処理の必要性については、PMv2の処理では必ず必要であるが、pmBufferArrayでは最初の1回のコピーのみ必要でそれ以降ではそのままコピーなしで実行が可能になっている。

以上のように、pmBufferArrayの導入により、大規模データのpingpong処理においては、関数の呼出回数を減らすばかりでなく、ホストプロセッサのコピー処理自体も最小限に抑えられていることがわかる。

7. おわりに

本稿では、PACS-CS上での主要アプリケーションで通信性能を最大限に引き出すための通信ライブラリについて、その課題と課題を解決するためにpmBufferArrayを導入した設計について述べた。

簡単なpingpongプログラムの例についてインターフェイスレベルで実装比較をしたところ、大規模データ時の通信処理関数の呼出回数を削減できる他、プロセッサによるデータコピーも削減できることを示した。

現在、pmBufferArrayの導入をPMv2の拡張インターフェイスとしてAPIを追加することにより実装を進めており、実装完了後、通信性能評価を行なう予定である。

参考文献

1) 朴泰祐, 佐藤三久, 宇川彰. 計算科学のための超並列クラスタPACS-CSの概要. 情報処理学会研究報告 05-HPC-103 (SWoPP'2005). 情報処理学会, August 2005.

2) 住元真司, 大江和一, 久門耕一, 朴泰祐, 佐藤三久, 宇川彰. 複数Gigabit Ethernetを用いたPACS-CSのための高性能通信機構の設計と評価. SACSIS 2006 - 先進的計算基盤システムシンポジウム. 情報処理学会, May 2006.

3) SCore Cluster System Software: http://www.pccluster.org/.

4) 住元真司, 堀敦史, 手塚宏史, 原田浩, 高橋俊行, 石川裕. 高速通信機構PM2の設計と評価. 情報処理学会論文誌, Vol.41 No. SIG 5 (HPS-1), pp. 80–90, August 2000.

5) Toshiyuki Takahashi, Shinji Sumimoto, Atsushi Hori, Hiroshi Harada, and Yutaka Ishikawa. PM2: A High Performance Communication Middleware for Heterogeneous Network Environments. In *Supercomputing 2000, IEEE and ACM SIGARCH,. November, 2000, (Published by CD-ROM).*, November 2000.

6) PM 2.1 API :http://www.pccluster.org/score/dist/score/html/en/man/man3/PM.html.

7) 住元真司, 堀敦史, 手塚宏史, 原田浩, 高橋俊行, 石川裕. 既存ＯＳの枠組を用いたクラスタシステム向け高速通信機構の提案. 情報処理学会論文誌, 第41巻 第6号, pp. 1688–1696, June 2000.

8) Shinji Sumimoto and Kouichi Kumon. PM/Ehernet-kRMA: A High Performance Remote Memory Access Facility Using Multiple Gigabit Ethernet Cards. In *3rd International Symposium on Cluster Computing and the Grid*, pp. 326–334. IEEE, May 2003.