Vol.2010-HPC-125 No.6
2010/6/17

マルチレール相互結合網における通信プロファイリングに基づく性能最適化

原田知徳[†1] 三木康次[†1] 三浦信一[†2]
塙敏博[†1,†2] 朴泰祐[†1,†2]

大規模 PC クラスタにおける重要な問題の一つとして，マルチコア化に伴う計算ノードの性能向上に対し，相互結合網の性能が追い付けず，性能上の乖離が進んでいることが挙げられる．マルチレールネットワークを用いたクラスタでは，基本的なバンド幅は向上するものの，実アプリケーションにおける通信パターンに応じたレール数選択等のチューニングが必要である．アプリケーションの通信特性を調べるには通信プロファイラを用いるのが一般的であるが，過剰に詳細なログ採取のためのメモリ及びファイルサイズの問題より，ソースコードを改変し特定区間の記録のみを取る工夫が必要である．これはユーザへの負担を増加し，またソースコード非公開の商用アプリケーション等には対応できない．

本研究では，MPI 関数単位での通信プロファイリングを，実行時刻ではなく実行時間を中心とした最低限の記録を保持することにより，メモリ及びファイル使用量を大幅に減らし，プロダクトランに直接適用可能とした通信プロファイラである MPIPROF を提案・実装する。MPIPROF を，マルチレールクラスタである T2K-Tsukuba に実装し，LS-DYNA 及び NICAM といった，大規模計算科学／工学並列アプリケーションに適用した．その結果，同ツールによってマルチレール化の影響を実測可能であり，プロダクトランでの使用にも問題がないことが確認された．

Performance Optimization based on MPI Profiler on Multi Rail Interconnection Network

Tomonori Harada,[†1] Koji Miki,[†1] Shin'ichi Miura,[†1]
Toshihiro Hanawa[†1] and Taisuke Boku[†1]

The performance gap between the computation performance and network performance on large scale PC clusters has been serious due to the core performance improvement and multi-core configurations. The multi-rail interconnection network to bind multiple links is a typical way to enhance the bandwidth of the network. However, it is not always effective due to the characteristics of the communication pattern on the application. For the analysis, it is practical to introduce the communication profilers, however, it may be required to modify the source code to save the memory foot-print and file size for profile recording.

In this research, we develope a new message passing profiler named MPIPROF which records the characteristics of each MPI function only recording the time consumption but the time stamp of each communication. It greatly reduces the memory foot-print and file size to record even for a large scale parallel job in product-run level. MPIPROF is implemented on multi-rail cluster T2K-Tsukuba and several computational science/engineering applications such as LS-DYNA and NICAM are applied. As a result, we could confirm the effect of multi-rail configuration as well as the availability of MPIPROF to apply for the product-run on real applications without source code modification.

1. はじめに

近年, プロセッサのマルチコア化により, プロセッサ単体での演算性能の向上が急速に進んでいる．現在の HPC(High Performance Computing) 分野で使用されている，大規模 PC クラスタにおいてもマルチコアプロセッサが搭載されたものが主流になっている．しかし，大規模 PC クラスタにおける，プロセッサ性能の著しい向上はノード当たりの高い演算性能が実現できる一方で，ノード間通信性能が並列処理時のボトルネックとなり，クラスタシステム全体での性能が最大限に発揮できない可能性がある．

プロセッサとネットワークの性能乖離によるクラスタ全体のパフォーマンス低下を防ぐためには，ノード間通信性能の最適化を行う必要がある．ノード間通信性能最適化の一手法として，プロファイラによりアプリケーションの通信性能の解析を行い，プログラムを改変する最適化手法があげられる．しかし，この手法ではアプリケーションの通信特性は実行環境により様々であり，それらを個別に調査するのはユーザにとって負担が大きなものとなる．また，商用アプリケーションなどの改変することができないプログラムに対してこの手法を適用することは困難である．

現在の大規模 PC クラスタにおけるノード間通信のバンド幅の増強の手法として，ノード間ネットワークのマルチレール化が考えられる．この手法では，ノード間を複数リンクのネットワークで接続することで高バンド幅のネットワークを可能としている．(以下，本稿

†1 筑波大学大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†2 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba

ではマルチレールネットワークの各ネットワークリンクを「レール」と呼び，それらを束ねることを「トランク」と呼ぶ.）期待通りの通信性能を出すためには, アプリケーションごとの通信特性に着目したネットワークのチューニングが必要となる．一番単純なチューニングとして，MPI 等の通信において何本のネットワークレールをトランクするかを選択するだけでも，性能に与えるインパクトが大きいと考えられる．マルチコアノードにおける CPU コアの使い方は，MPI のみのフラットなプロセス構成や，OpenMP 等のスレッド並列処理と組み合わせが考えられ，ネットワークトランキングにおける効率の低下を考慮すると，必ずしも全ての通信で全てのレールをトランクすることが最適とは限らない．他方、通信のタイミングがプロセス毎にバラつく場合は，できるだけトランクした方がよい．理想的には，大規模 PC クラスタにおけるマルチレールネットワークの最適化を，アプリケーション特性を考慮した上で，自動的に行うことが望ましい．そのためには，通信プロファイラで，アプリケーションの通信特性を解析した後，トランクすべきレール数を決定して PC クラスタを使用することが求められる．

大規模 PC クラスタで使用される代表的な通信プロファイラとして，MPE(MPI Parallel Environment)[4] が挙げられる．しかし，MPE は MPI プロセス間の並列通信状況の可視化と解析を目的としており，全ての通信履歴をログとして記録するため，大規模・長時間の並列処理において使用した場合，取得データに必要なメモリサイズが増大し，出力ファイルも膨大になる．このような問題点を改善するために，我々は大規模 PC クラスタ向けの，MPI プロファイラといて新たに MPIPROF の開発を進めている．MPIPROF では，アプリケーション実行中の通信記録におけるメモリ消費量を極力削減し，プログラムの改変をせずにプロファイルを取得し，既存の通信プロファイラでのメモリ消費量の増大やプログラムのふるまいの変化といった問題を起こすことなく，アプリケーションの通信解析を容易に行うことを目的としている．本稿では実際に HPC 分野で使用されている科学計算アプリケーションである，LS-DYNA[7]，NICAM[9] について MPIPROF を用いて通信性能を解析した実例を述べる．

2. 並列処理における通信特性解析手法

昨今の大規模 PC クラスタにおいてマルチレールネットワークの利用が増加している．筑波大学計算科学研究センター[?] で稼働中の，T2K-Tsukuba システムでは 4 本の Infiniband をトランキングして，高い通信性能を実現している.[2] しかし，マルチレールネットワークでは N 本のトランキングが N 倍の性能に結びつくとは限らず，ノード間通信性能は同時通信ストリームのレールやメッセージサイズに依存して変化する可能性がある.[3] 最適な通信性能を得るためには，アプリケーションが使用する通信特性を解析して，トランキングするレール数を決定する必要がある．

アプリケーション通信特性の解析には通信プロファイラが用いられる．MPI の既存の通信プロファイラとして MPE（MPI Parallell Environment)[4] が挙げられる．MPE は MPI の実装系として広く使われている mpich を提案・開発している Argonne National Laboratory によって作成され、MPI 通信ロギングの機能や，可視化ライブラリが含まれている．MPE でログを取得後，jumpshot[5] などの可視化ソフトを使用することにより通信状況を確認することができる．MPE の問題点として，時系列情報を個別の通信について全て記録し，かつ実行時間への副作用を最小にするために全データをメモリに貯めるため，つまり，大規模・長時間の並列処理においては取得データに必要なメモリサイズが増加し，出力ファイルも膨大になる問題がある．特に，MPI プロセス数が多く，通信回数も膨大になるような大規模実アプリケーションのプロダクトラン状況で利用すると，ノード内のメモリを使い切ってしまう場合もある．この問題を回避するために，MPE ではログを取得する範囲を指定することができるが，プログラムを改変する必要があるため，ユーザへの負担が生じる．また，元来 MPE はアプリケーションの通信特性を特定区間において詳細に解析することが目的でおり，大規模プログラムの完全な通信記録を取る事自体，想定される利用方法に反すると考えられる．しかし，商用アプリ等を個別のクラスタ環境でチューニングする場合はソースコードの改変が不可能であったり，そもそもアプリケーションユーザが余計な作業を行うことなしに，プログラムの通信特性を最低限解析したいという要求は存在し，MPE はその目的には適さない．

3. MPIPROF

商用アプリケーションや大規模並列アプリケーションにおいて，通信ログ記録区間の指定を含むソースコード改変を要求せず，ユーザの負担なしに最低限の通信記録を低コストで取得することは必要であると我々は考えている．そこで，この要求を実現するために我々はアプリケーション実行時におけるメモリ消費量を極力抑え，また通信時間の記録における時間的副作用を最小限にするよう工夫をした通信プロファイラである MPIPROF を開発した．

MPIPROF では，既存の MPI プロファイラにおける問題点であった，

- 通信プロファイリング時の全通信ロギングによるメモリの圧迫
- プロファイリングにおけるプログラム実行時間の変動

を改善する．

以下に，MPIPROF の持つ機能及び特徴について述べる．

3.1 基本設計

MPIPROF はアプリケーションで利用されている全ての MPI 関数を対象に，関数単位での通信プロファイルを取得する．また，個々の通信の発生時刻はログに記録せず，通信に要した時間だけを記録している．このため，プロファイラが消費するメモリ量は，プロファイラが記録する対象とする MPI 関数の個数にのみ比例し，実行時間に関わらずほぼ一定量となる．

3.2 通信記録の取得方法

MPIPROF では，処理時間の統計量とヒストグラムの 2 つの形式で通信記録を取得する．処理時間統計機能では，各処理の開始時刻，終了時刻を取得するが，そこから処理時間を計算し，実行時間の最大，最小，合計のみに限定して記録することによって省メモリ化を実現している．また，時間取得のためのタイマーには RDTSC (Read TimeStamp Counter) 命令を利用して実装している．RDTSC 命令はプロセッサに内蔵された起動時からの経過クロック数を保持するタイムスタンプカウンタ値を取得する命令で，この命令を利用することにより CPU クロックと同じ分解能を持つ高精度タイマを実現できる．また，RDTSC 命令を用いることによりタイムスタンプ取得の際のオーバヘッドを大幅に低減できる．MPIPROF では，実行時に与える configuration file により，ログを取得する MPI 関数を制限できるようになっている．これにより，ユーザはアプリケーション実行中に必要な情報のみを取得することができ，不要なオーバヘッドを削減するようになっている．ヒストグラムは MPI 関数の各実行時のデータサイズを個別に記録することを避けて，メッセージサイズ毎のヒストグラムとして管理することで大幅な省メモリ化を実現している．

3.3 取得可能データ

MPIPROF では，ユーザ各関数毎に取得できるデータは以下の通りである．各 MPI プロセスのそれぞれの関数毎で個別に管理している．

- 呼び出し回数
- 実行時間の最大・最小・平均値，および合計実行時間
- メッセージサイズ最大・最小・平均値，および合計メッセージサイズ
- メッセージサイズ毎のヒストグラム（2 のべき乗刻みの 1,2,4,...,4M,8M 以上の 24 階級）
- 通信相手のプロセス番号と全通信回数

また，MPI_Allgather 関数のような集合通信関数では，送出メッセージサイズと受領メッセージサイズを指定するものがあるため，そのような関数では送出量と受領量の 2 つそれぞれのメッセージサイズヒストグラムを取得する．また，MPI_Allgatherv 関数のように，各 MPI プロセス毎にメッセージサイズが異なる MPI 関数については，当該プロセスのメッセージサイズのみをヒストグラムとして取得する．

3.4 実　　装

MPI の仕様において，本システムのようなプロファイリングライブラリを構築するために MPI プロファイリング・インタフェースが用意されている．MPI プロファイリング・インタフェースでは，本来の MPI 関数名の “MPI” を “PMPI” に変えた関数がそれぞれ用意されており，本来の MPI 関数と全く同じ動作をする．そこで，通常ユーザが使う “MPI_...” という名称の関数をラッパー関数として用意し，内部で “PMPI_...” を呼び出し，前後に必要な処理を追加することによって，結果的に付加機能を付けた MPI 関数群を作成できる．本システムも PMPI 関数を用いて MPI 関数のラッパー関数を作成し，システムを実装した．図 1 は C 言語のラッパー関数 (MPI_Send) の実装例である．ラッパー関数内で PMPI 関数 (PMPI_Send) の前後に時刻取得のための関数 (mpiprof_log_in，mpiprof_log_out)，ヒストグラム取得のための関数 (HIST_LOG) を挿入することにより各種情報を取得している．実際には各時刻を記録するのではなく，先述した情報を記録するための累積情報（その関数の総実行時間，呼び出し回数，メッセージ長の総量等）のみを記録する．図 2 は Fortran 言語のラッパー関数 (MPI_Send) の実装例である．ラッパー関数内で C 言語と Fortran 言語のデータ型が異なるため，MPI ライブラリに用意されているデータ変換関数である，“MPI_..._f2c”，“MPI_..._c2f” を使用する必要がある．(図 2 では，MPI_Type_f2c，MPI_Comm_f2c がこれにあたる)

以上のように，ラッパーライブラリとして実装することによって，元の MPI ライブラリやユーザのプログラムに手を加えずにオブジェクトの再リンクのみで簡単に使用することができる．現在，38 種類の C 言語及び Fortran 言語の MPI 関数に対応している．[⋆1]

3.5 オーバヘッド

MPIPROF の時刻取得のための関数とヒストグラム取得のための関数部分にかかるログ取得のオーバヘッドの測定を行ったところ，平均で 714nsec であった．これは一般的に MPI プログラムの時間計測を行う際に使用される MPI_ Wtime を 2 回呼び出す（開始時刻, 終

⋆1 現在 MPIPROF は開発段階にあるため，全ての MPI 関数に対する実装は完了しておらず，現段階で開発対象として使用している LS-DYNA 及び NICAM で使用される全ての関数（38 個）を対象としている．原理的には，同じ手法で全関数の実装が可能である．

```
int MPI_Send(void *buf, int count,
        MPI_Datatype datatype,
        int dest, int tag,
        MPI_Comm comm) {

 HIST_LOG(HEV_SEND, &count,
                datatype, &dest);

 mpiprof_log_in(EVENT_MPI_SEND);
 int ret = PMPI_Send(buf,
                count, datatype,
                dest, tag, comm);
 mpiprof_log_out(EVENT_MPI_SEND);

 return ret;
}
```

図 1 C言語 MPI ラッパー関数例（MPI_Send）

```
void mpi_send_(void *buf, MPI_Fint *count,
                MPI_Fint *datatype,
                MPI_Fint *dest, MPI_Fint *tag,
                MPI_Fint *comm, MPI_Fint *ierr) {

 HIST_LOG(HEV_SEND, count,
                MPI_Type_f2c(*datatype), dest);

 mpiprof_log_in(EVENT_MPI_SEND);
 *ierr = PMPI_Send(MPIR_F_PTR(buf),
                *count,MPI_Type_f2c(*datatype),
                *dest,*tag, MPI_Comm_f2c(*comm));
 mpiprof_log_out(EVENT_MPI_SEND);

 return;
 }
```

図 2 Fortran 言語 MPI ラッパー関数例（MPI_Send）

了時刻）のオーバヘッドが平均で 2340nsec であることから，MPI_Wtime よりも 1 桁弱高い精度で測定ができることになる．また，実際の通信の通信時間を検証した結果,，メッセージサイズ 8byte の Pingpong 通信の通信時間で平均で 2μsec であった．（T2K-Tsukuba, MVAPICH2 における実測）この時、MPIPROF を用いると，そのオーバヘッドが通信時間の 35%ほどになってしまうため，このようなメッセージサイズが小さい通信では性能に与える影響が大きいと言える． しかし，メッセージサイズが 1KB の Pingpong 通信では通信時間が平均で 8μsec となり，MPIPROF のオーバヘッドは通信時間の 10%未満となるため，MPIPROF のオーバヘッドによる影響は少ないと考えられる．このように，通信にかかる通信時間によって，MPIPROF のオーバヘッドの影響の大きさが変わってくるため，今後，オーバヘッドの割合が高くなる通信ではログを取得しないようにする，などの改良が必要であると考えられる．現状の手法として，我々は，一度プロファイルを採取した後，総時間として影響の大きい MPI 関数について平均実行時間をチェックし，それが先述の時間計測オーバヘッドのオーダーに近ければ，その関数の実行時間については「要注意」として警告する，という手法を取っている．もしアプリケーション実行後に，結果的にそういう関数がないことがわかれば，ユーザはそのプロファイル情報を信用してよいということになる．

4. 評 価 実 験

本章では評価実験の概要，評価に用いるシステム及びアプリケーションについて述べる．

4.1 評価用システム

本研究では，T2K-Tsukuba システム[1] を使用して評価実験を行った．表 1 に T2K-Tsukuba システムの構成に示す．各ノードには 4 つの CPU コアを持つ AMD Quad-Core Opteron（2.3GHz) を搭載している．

表 1 T2K-Tsukuba のノード構成とノード当たり性能

構成	16 CPU core (4 core/socket x 4 socket)
CPU	AMD Quad-Core Opteron8356 2.3GHz （Barcelona）x 4
総メモリ容量	32GB DDR2 667MHz （ソケット当たり 8GB）
理論総メモリバンド幅	42.7GB/s
理論ピーク性能	147.2Gflops/node
ネットワークリンク	4xDDR Infiniband (Mellanox ConnectX) x 4
ピーク通信性能	8GB

T2K-Tsukuba の MPI ライブラリとして用いられている MVAPICH2[6] は，マルチレールネットワークの利用のために，マルチレールのスケジューリングポリシーを提供している．ユーザはアプリケーションの通信方式，メッセージサイズなどの通信特性に応じて適切なスケジューリングポリシーを設定することにより，通信性能の向上が期待できる．本評価では，このスケジューリングポリシーの一つである環境変数 MV2_NUM_HCAS を使用して実験を行う．環境変数 MV2_NUM_HCAS では，ノード内の全プロセスが使用する Infiniband のレール数を指定する変数である．

4.2 評価アプリケーション

本節では，評価実験に用いる科学計算アプリケーション及び評価実験の概要について述べる．

4.2.1 LS-DYNA

LS-DYNA[7] とは，Livermore Software Technology Corporation より提供される非線形動的構造解析ソフトウェアである．LS-DYNA は陽解法により構造物の大変形挙動を時刻履歴で解析するプログラムであり，衝突／衝撃解析，落下解析，塑性加工解析，貫通／亀裂／破壊解析などの現象を解析を行えることから，自動車，航空宇宙，建設・土木など様々な分野で構造解析に活用されている．計算速度の向上のために並列化されたプログラムで実装さ

れており，複雑な現象解析であっても，並列度を変更することで，計算スピードを向上させることが可能である．なお，LS-DYNA は商用アプリケーションで，そのソースの内容は公開されていない．本評価実験では，日本自動車工業会[8] の協力により，衝突現象を解析する自動車解析モデルを使用して，LS-DYNA の使用している MPI 通信関数の解析を行った．

表 2 に LS-DYNA 性能評価の実験環境を示す。LS-DYNA の評価では，MPI プロセス数 (並列度) が 64，128，272 の 3 パターンについて，マルチレール数が 1，2，4 のときの実行時間について測定した．アプリケーションモデルは構造解像度の比較的低いモデルを使用した．測定後に MPIPROF の処理時間統計量機能を使用して，使用された MPI 通信関数についての実行時間の解析も行った．

表 2 LS-DYNA 性能評価の実行環境

アプリケーション	LS-DYNA 三菱モデル
使用ノード数	4,8,17
MPI プロセス数	64,128,272[⋆1]
マルチレール数	1,2,4
MVAPICH2 オプション	MV2_NUM_HCAS=1,2,4

4.2.2 NICAM

NICAM[9] とは東京大学気候システム研究センターと地球環境フロンティア研究センターの共同研究により開発された，全球気候モデルシミュレーションプログラムである．多くの大気大循環モデルがスペクトル法により数値積分を行うのに対し，NICAM では正二十面体の格子点網による数値積分を実施する．NICAM は全球を数 km のメッシュで覆った全球雲解像大気モデルシミュレーションであり，不確定要素のある積雲対流のパラメタリゼーションを含まないのが特徴である．また，短期の予報モデルとしてだけではなく，長期の気候モデルとしても利用可能で，地球シミュレータをはじめとする大型並列計算機で動作するように設計されている．並列化の計算領域は正二十面体の一つの菱形で領域を組み，各三角形を 4 つの三角形に分割することを繰り返すことで，形成される．

表 3 に NICAM 性能評価の実験環境を示す。NICAM の評価では，MPI プロセス数 (並列度) が 32，80，160 の場合について，マルチレール数が 1，2，4 のときの実行時間について測定を行った．また，G-Level 2，R-Level 2 という，小規模モデルを使用した．測定後に MPIPROF の処理時間統計機能を使用して，使用された MPI 通信関数についての実行時間の解析も行った．

表 3 NICAM 性能評価の実行環境

アプリケーション	NICAM
使用ノード数	10
MPI プロセス数	32,80,160
マルチレール数	1,2,4
G-Level	2
R-Level	2
MVAPICH2 オプション	MV2_NUM_HCAS=1,2,4

5. 評価結果

5.1 LS-DYNA 評価結果

図 3 に LS-DYNA の実行結果を示す．縦軸が経過時間，横軸が並列度と使用したマルチレール数である．また，積み上げグラフで上段が通信時間，下段が計算時間を表している．実行時間は 1 レール指定で 272 並列の性能が大きく低下していることを除くと，マルチレール化による実行時間短縮の効果があることが分かる．2 レール指定では 128 並列と 272 並列では経過時間はほぼ変わらない．一方で 4 レー指定では 272 並列まで処理時間の短縮が見られる．次に，図 4 に各 MPI 関数の処理時間を示す．プログラム実行後に MPIPROF の処理時間統計機能を使用して，各 MPI 関数の通信所要時間の平均値を用いて，処理時間が大きい MPI 関数から順に 10 個並べた．縦軸が処理時間，横軸が MPI 関数を表す．(以降，関数名冒頭の MPI_は省略する）1 レール指定で 272 並列では，Recv，Waitany，Send が他の関数と比べて，大幅に経過時間が大きく，全通信時間の半数近くを占めている．この結果から，図 3 の実行結果で見られた，272 並列での性能悪化は，Recv，Waitany，Send が影響している可能性が高いことがわかる．

2 レール指定で 128 並列と 272 並列の各関数を比べると，272 並列での Waitany，Send が 128 並列に比べて倍以上の経過時間を使用している．この結果から，2 レール指定では 128 並列と 272 並列で並列性能が変わらなかったのは，272 並列で Waitany,Send が多くの通信時間を使用していることが原因だと考えられる．

図 3 から，Allreduce，Recv といった経過時間が大きい通信は，マルチレール化が有効に

⋆1 256 並列で一部のケースでうまく実行しなかったため（MPIPROF の影響ではない），272 プロセスで評価した．

表 4　LS-DYNA の実行時間 [sec]

MPI プロセス数	1-Rail	2-Rail	4-Rail
64	18939	17680	17592
128	15514	12300	11060
272	20221	13074	9846

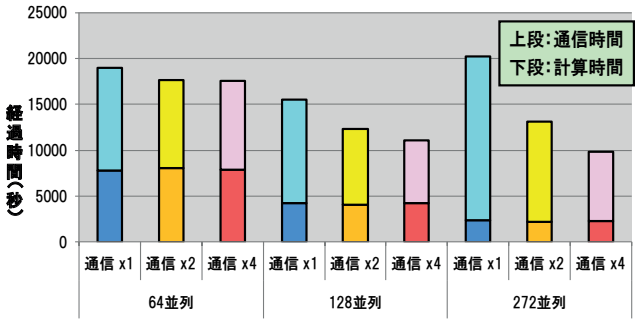

図 3　LS-DYNA：実行結果

なっていることから，4 レールを使用してプログラムを並列実行することが望ましい．

5.2　NICAM 評価結果

表 5 に NICAM の実行結果を示す．各並列度において，レール数を増やしても経過時間がほぼ変わらないことからマルチレール化の効果が出ていないことが分かる．次に図 5 に NICAM における各 MPI 関数の処理時間を示す．プログラムの実行後に，MPIPROF の処理時間統計機能を用いて，ログビューアから各 MPI 関数の処理時間の平均値を用いて，経過時間が大きい MPI 関数から順に並べた．縦軸が経過時間，横軸が MPI 関数を表す．図 5 からレール数を増やしても，処理時間の短縮になっておらず，逆に処理時間が大きくなっている MPI 関数が多いことが分かる．次に，ヒストグラム取得機能を使用して，図 5 において経過時間が大きい，集団通信である Bcast についての通信メッセージサイズを調べたところ，合計転送量 2161960[byte]、最大メッセージサイズ 648 487[byte]、最小メッセージサイズ 4[byte] であった．データサイズの分布を調べたところ，8〜15[byte] という小さいメッセージサイズが通信のうちの 90%以上 (151823/155674) を占めていることが分かった．この結果より，小さなメッセージサイズでの集団通信 (Bcast) を多く行っているため，マルチレール化によるバンド幅の増強の効果が出ていないのが原因であると考えられる．

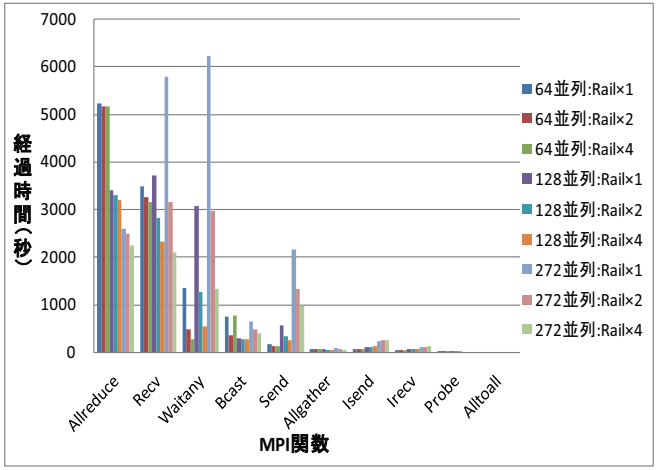

図 4　LS-DYNA：各 MPI 関数の経過時間

表 5　NICAM：実行時間 [sec]

MPI プロセス数	1-Rail	2-Rail	4-Rail
32	25.5747	25.5713	25.6604
80	9.7363	9.8884	10.0356
160	6.4391	6.5757	7.0726

6. ま と め

本研究では，大規模 PC クラスタでのマルチレールのレール数をアプリケーションの通信特性に基づいて決定するために，通信プロファイラである MPIPROF の開発を行った．MPIPROF には従来の通信プロファイラを大規模 PC クラスタ上で使用した場合の問題点であった，通信プロファイリング時の全通信ロギングによるメモリ圧迫を改善し、ユーザがプログラムの可変を行わなくても済む実装にした．そして，MPIPROF の処理時間統計機能，ヒストグラム機能を用いて，実際の HPC アプリケーションである LS-DYNA，NICAM を用いて評価実験を行い，使用された MPI 関数の解析から，トランキング時のレール数の決定の実例を紹介した．

MVAPICH2 では，プログラム実行前に環境変数 MV2_NUM_HCAS でレール数を指定

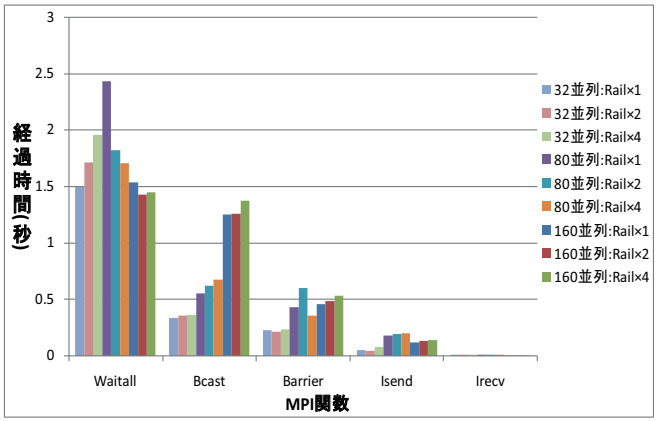

図 5 NICAM：各 MPI 関数の経過時間

する必要がある．今後は様々な HPC アプリケーションに対して，各種条件で MPIPROF でのプロファイリングをした後に，最適値を決定し，その結果に基づいて，大規模 PC クラスタにおいてマルチレールネットワークの自動最適化を行うシステムの開発を行う予定である．

謝　　辞

本研究の一部は筑波大学計算科学研究センター・学際共同利用平成 21 年度採択課題「マルチコア／マルチレーン・クラスタにおける並列処理最適化」による．本研究で対象とした LS-DYNA の解析モデル提供において自動車工業会の協力 0 頂いた．また，NICAM の解析モデルの提供において筑波大学計算科学研究センター教授・田中博博士及び同研究員・寺崎康児博士の協力を頂いた．各関係各位に深く感謝する．

参 考 文 献

1) 高橋大介, 後藤和茂, 朴泰祐, 建部修見, 佐藤三久, 三上和徳 : T2K 筑波システムにおける Linpack 性能評価, 2008-HPC-116, No. 74, pp.174 2008.
2) T2K Open Supercomputer Alliance : http://www.open-supercomputer.org/index.html
3) 松葉, 石川：コモディティネットワークによる 5GB/s 通信の可能性, 2007-HPC-109, No. 29, pp. 55-60, 2008.
4) MPI Parallel Environment(MPE) : http://www.mcs.anl.gov/research/projects/perfvis/software/MPE/index.htm
5) Jumpshot : http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm
6) MVAPICH2 : http://mvapich.cse.ohio-state.edu/overview/mvapich2/
7) Livermore Software Technology Corporation: http://www.lstc.com/
8) JAMA - 一般社団法人日本自動車工業会: http://www.jama.or.jp/
9) Nonhydrostatic icosahedral atmospheric model (NICAM) for global cloud resolving simulations: M. Satoha, T. Matsunoa, H. Tomitaa, H. Miuraa, T. Nasunoa and S. Igaa