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

2006－HPC－107 (5)
2006／7／31

# MPI/GXP：広域環境用の適応的なメッセージパッシングシステム

斎藤秀雄† 田浦健次朗† 近山隆†

我々が開発している広域分散計算環境用のメッセージパッシングシステム MPI/GXP について説明する．MPI/GXP は計算環境が実行毎に変化するということを意識して，実行時に測定した遅延や通信量を基に様々な性能最適化を行う．5 クラスタ 256 プロセッサという環境では，遅延を考慮した接続確立を行うことによって，既存のグリッド用メッセージパッシングシステムのようにルータが維持できるセッション数に制限されることなく動作した．また，通信オーバヘッドを考慮した rank 割り当てを行うことによって，ランダムな rank 割り当てを行った場合と比べて NAS Parallel Benchmarks の性能が 60%から 100%向上した．

# MPI/GXP: An Adaptive Message Passing System for Wide-Area Environments

Hideo Saito,† Kenjiro Taura† and Takashi Chikayama†

We describe MPI/GXP, the wide-area message passing system that we are developing. MPI/GXP focuses on the fact that the computational environment changes from execution to execution, and uses latency and traffic measurements made at run-time to determine how to establish connections and how to map ranks to processes. In an experiment in which we used 256 processors in 5 clusters, we were able to avoid exceeding the number of sessions that a router can handle by using RTT measurements made at run-time to establish connections mainly with nearby processes instead of with all processes. In another experiment, the performance of the NAS Parallel Benchmarks increased by 60% to 100% when we replaced a random rank-process mapping with a rank-process mapping that used latency and traffic measurements made at run-time to make communication overhead small.

## 1. はじめに

近年の広域ネットワークの帯域の増加に伴い，複数のクラスタを広域ネットワークで接続して並列計算を行う機会が増加している．そのような背景を基に，広域環境用の（グリッド環境用の）メッセージパッシングシステムの研究・開発が盛んに行われている．

従来のクラスタ環境におけるメッセージパッシングでは，用いる計算環境を意識した性能最適化はメッセージパッシングシステムによって行われてきた．これによって，プログラマはメッセージパッシングシステムによって各プロセスに割り当てられた ID のみを用いて並列プログラムを記述することができた．

グリッド環境はクラスタ環境より多様であるため，環境を意識せずに並列プログラムを書くことができることの恩恵はクラスタ環境の場合より顕著である．一方，グリッド環境用のメッセージパッシングシステムは，従来のクラスタ環境用のメッセージパッシングシステムの課題に加えて，グリッド環境特有の課題も解決しなければならない．たとえば，クラスタ環境では全ノードが互いに通信できると仮定できるが，グリッド環境では一部のプロセスの間の通信がファイアウォールなどによって遮断されていることがある．また，クラスタ環境ではすべてのリンクの速度が同じと仮定できるが，グリッド環境では LAN のリンクと WAN のリンクの速度は大きく異なる．

これらの課題を解決するために様々な手法が提案されてきた．[1),5),6),12)] しかし，それらは基本的に計算環境が静的であると仮定しており，現実の動的なグリッド環境を正確に捉えていない．たとえば，直接通信できないプロセス間でもメッセージの送受信が行えるようにするために，クラスタ間のメッセージはプロキシを中継するという手法が提案されている．そのような手法では用いる計算資源は実行毎に変化しないと仮定しているため，プロセス間の接続の張り方や経路は静

† 東京大学
The University of Tokyo

的なものを用いる．しかし，実際にはグリッド環境では利用可能な計算資源は実行毎に変化するため，プロセス間の接続の張り方や経路は実行時に動的に決定する必要がある．

そこで我々は，実行毎に変化する計算環境に実行時に適応するメッセージパッシングシステム MPI/GXP を開発している．MPI/GXP は計算環境は実行時まで未知と考えて，アプリケーション起動時にプロセス間の遅延や通信量を測定することによってその実行に用いられる計算環境に応じた様々な性能最適化を行う．ノードの故障などの実行中の計算環境の変化に対応することも視野に入れているが，本稿では計算環境はアプリケーション起動時に確定するものとする．

以降，2 章で MPI/GXP の設計と実装について説明して，3 章で評価実験について説明する．4 章で関連研究について述べる．最後に，5 章でまとめと今後の課題を述べる．

## 2. MPI/GXP の設計と実装

### 2.1 MPI/GXP の概要

我々の開発している MPI/GXP は Message Passing Interface (MPI)[8] の主要な部分を実装したグリッド環境用のメッセージパッシングシステムである．

MPI/GXP は計算環境に関する様々な情報を実行時に自動的に収集するので，起動時に与える必要がある唯一の情報は各プロセスのエンドポイント（IP アドレスとポート番号）である．各プロセスはこの情報を基に他のプロセスと接続を張るが，グリッド環境では一部のプロセスの間の通信がファイアウォールなどによって遮断されていることがある．そこで，各プロセスは実行時に実際に他のプロセスと接続を試みることによってどのプロセスと接続を張れるかを発見する．また，接続を張れなかったプロセスとも通信が行えるようにメッセージのルーティングを行う．

エンドポイントの与え方には様々な方法が考えられるが，ここではグリッド用のシェルである GXP[10] を用いる場合について説明する．GXP は，ユーザが少数のノードを手動で与えると，周りのノードを NIS などを用いて見つけて，ツリー上にログインする．MPI ジョブを投入する際にはログインしたノードの部分集合を選択して MPI プロセスを立ち上げる．このとき，各プロセスは自分の IP アドレスと bind したポート番号を GXP のログインツリーに沿ってブロードキャストすることによって他のプロセスに自分のエンドポイントを伝える．

ここで，高い通信性能を出すためには，与えられたエンドポイント間で張る接続の数を調節する必要がある．全プロセスが互いに通信できるようにするためには，全プロセスを連結にするのに必要な最低限の接続（$p$ プロセスの間で $p-1$ 本）を張ってルーティングを行えば良い．しかし，これでは多くの中継が必要となるため，通信性能が低くなってしまう．特に，近いプロセス同士の通信では RTT (Round-Trip Time) が大きな影響を受けてしまう．一方，通信が可能な全プロセスの間で接続を張ると，以下のような問題が生じてしまう．

- ルータ・ファイアウォールが維持できるセッション数を超える
- TCP の送受信バッファに多くのメモリを要するふたつのプロセスの間に入るルータ・ファイアウォールの数も，必要となるバッファメモリの量も，ふたつのプロセスの間の距離が長くなるほど多くなる．

そこで，MPI/GXP はプロセス間の遅延を考慮して，近いプロセスの間では密に接続を張りつつ，遠いプロセスの間で張る接続の数は制限する．詳細については 2.2 節で説明する．

MPI/GXP が提供する他の主な機能のひとつに，通信オーバヘッドを考慮した rank 割り当てがある．MPI では，プロセス間の通信を，rank という MPI ライブラリによって実行時に各プロセスに割り当てられる ID を用いて記述する．プロセス間の通信速度が一様である場合はアプリケーションの通信オーバヘッドは rank の割り当て方によらないが，通信速度にばらつきがある場合は通信量の多い rank 対を速いリンクで繋っているノード上のプロセスに割り当てることによって通信オーバヘッドを小さくすることができる．グリッド環境では LAN 内のリンクの速度と LAN 間のリンクの速度は大きく異なるため，rank 割り当てによって通信オーバヘッドにも大きな違いが生じる．そこで，MPI/GXP はプロセス間の通信コストと rank 間の通信量を基に通信オーバヘッドを小さくするような rank 割り当てを行う．詳細については 2.3 節で説明する．

### 2.2 遅延を考慮した接続確立

MPI/GXP が近いプロセスの間では密に接続を張りつつ，遠いプロセスの間で張る接続の数を制限する方法について説明する．

各プロセスは以下のように，他のプロセスとの RTT を基に，どのプロセスと接続を張るかを決定する．ただし，$p$ はプロセス数，$k$ はパラメータとする．

- 最も近い $2k$ プロセスとは直接接続を張る
- $2^{i-1}k+1$ から $2^{i}k$ 番目に近いプロセスとはランダムに $k$ 個のプロセスと接続を張る

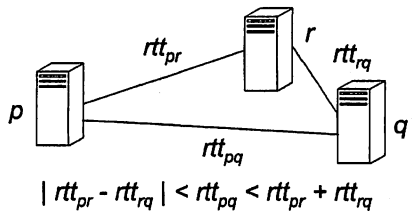



図 1 3 プロセス間の RTT と三角不等式
Fig. 1 The RTT Between 3 Processors and the Triangular Inequality

$(i = 2, 3, ..., log_2 p/k)$

- あるプロセスと互いに接続を張ってしまった場合は，片方の接続だけ残す

RTT はアプリケーション起動時に各プロセスが自律的に測定するが，全対全で測定を行うと $O(p^2)$ のメッセージ数が必要になってしまう．そこで，プロセス間で情報を共有して，実際には測定していない RTT を見積もることによって，メッセージ数を軽減する．

**図 1** のように 3 プロセスの間の RTT に三角不等式が成り立つとすると，$rtt_{pr}$ と $rtt_{rq}$ から $rtt_{pq}$ を見積もることができる．特に，$rtt_{pr}$ と $rtt_{rq}$ の比が大きい場合は，高い精度で $rtt_{pq}$ を見積もることができる．具体的には，あるパラメータ $\alpha$ に対して，以下のように $rtt)_{pq}$ を見積もる．

- $rtt_{pr} < \frac{1}{\alpha} rtt_{rq}$ の場合，$rtt_{pq} = rtt_{rq}$ とみなす
- $rtt_{pr} > \alpha rtt_{rq}$ の場合，$rtt_{pq} = rtt_{pr}$ とみなす

これによって，$O(plogp)$ のメッセージ数で全プロセス間の RTT が見積もれる．

クラスタ数を c として，プロセスはクラスタ間に均等に分散されていると仮定すると，この手法は以下のような性質を満たす．

- 各プロセスが張る接続の数は合計 $klog_2 p + k(1 + log_2 k)$ 本以下 $(O(logp))$
- 各プロセスが別のクラスタのプロセスと張る接続の数は $klog_2 c$ 以下 $(O(logc))$
- 各クラスタ間で張られる接続の数は $2kp/clog_2 c$ 本以下 (O(n/c log c))

### 2.3 通信オーバヘッドを考慮した Rank 割り当て

MPI/GXP がプロセス間の通信コストと rank 間の通信量を基に通信オーバヘッドを小さくするような rank 割り当てを行う方法について説明する．

プロセス間の通信コストは，遅延，帯域，中継オーバヘッドなど様々なものによって決まるが，ここでは簡単のために通信経路に沿って各ホップの RTT を合計したものとする．RTT は 2.2 節で説明した手法で実行時に測定・見積もりを行ったものを用いる．

表 1 実験環境
Table 1 Experimental Environment

| | | |
|---|---|---|
| Cluster A | Xeon 3.0GHz x 3 | Linux |
| Cluster B | Xeon 2.4GHz x 69<br>Xeon 2.8GHz x 59 | Linux |
| Cluster C | Xeon 2.4GHz x 109 | Linux |
| Cluster D | Athlon 2.2GHz x 1<br>Opteron 2.2GHz x 10 | Linux |
| Cluster E | Xeon 2.8GHz x 2<br>Celeron 1.3GHz x 1<br>Pentium III 1.1GHz x 1<br>PowerPC G4 1.5GHz x 1 | Linux |

Rank 間の通信量は，反復法を用いるアプリケーションなどが実行を通して似た通信を繰り返すことに注目して，「お試し実行」から見積もる．具体的には，アプリケーションを短時間実際に実行することによって，各 rank 対の間で送受信されるメッセージ数を数えて，それを通信量とする．

ここで，以下のように通信コスト行列 $C$ と通信量行列 $T$ を定義する．

- $C = \{c_{ij}\}$
  $c_{ij}$: プロセス $i$ からプロセス $j$ への通信コスト
- $T = \{t_{ij}\}$
  $t_{ij}$: お試し実行の rank 割り当てにおけるプロセス $i$ からプロセス $j$ への通信量

すると，(1) 式の値を最小化するような rank 割り当てを見つければ，アプリケーションの通信オーバヘッドを最小化したことになる．

$$\sum_{i=0}^{p-1} \vec{c_i} \cdot \vec{t_i} \qquad (1)$$

この問題は二次割り当て問題 (QAP: Quadratic Assignment Problem)[7] と同等であり NP 困難であるが，近似解を求めるために様々な手法が提案されている．MPI/GXP は Burkard らによって提案された simulated annealing に基づいた近似解法[2] と似た手法を用いる．

## 3. 評価実験

### 3.1 接続数

5 台のクラスタに分散された 256 プロセッサを用いて NAS Parallel Benchmarks (NPB3.1-MPI)[3],[11] の EP ベンチマークを実行して，張られた接続数を調査した．各クラスタの構成は**表 1** に示す．

**図 2** に結果を示す．各クラスタの隣の割合はクラスタ内で張られた接続数の割合を，クラスタを繋ぐ線の隣の割合はクラスタ間で張られた接続数の割合を示し

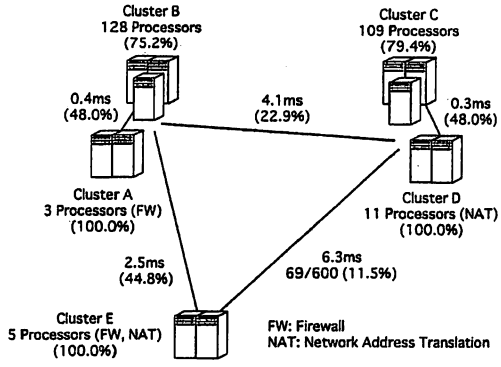



図 2 クラスタ内・クラスタ間で張られた接続数の割合
Fig. 2 The Percentage of Connections Established Within and Between Clusters

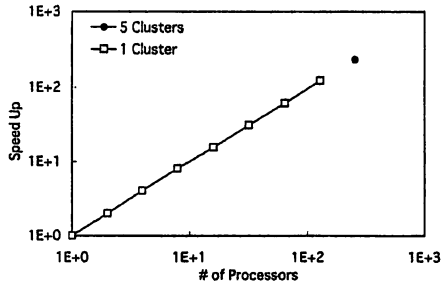



図 3 EP ベンチマーク
Fig. 3 EP Benchmark

ている．接続数の割合は以下の順に減っており，

(1) プロセッサ数の少ないクラスタ内 (A, D, E)
(2) プロセッサ数の多いクラスタ内 (B, C)
(3) 近いクラスタ間 (A-B, C-D)
(4) 遠いクラスタ間 (AB-CD, CD-E, E-AB)

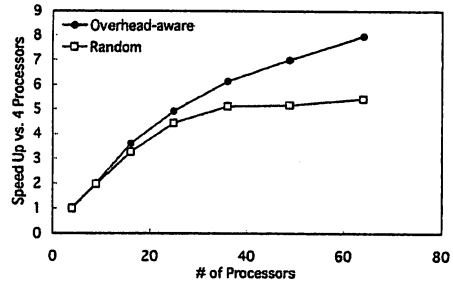
近いプロセスの間では密に接続を張られつつ，遠いプロセスの間で張る接続の数は制限されたことが分かる．

図 3 に EP ベンチマークの性能を示す．提案手法を用いて接続数を制限しなかった場合は維持できるセッション数を超えてしまったルータがあったため 5 台のクラスタを用いた実行は不可能であったが，接続数を制限することによって可能になり，単一のクラスタを用いた場合より高い性能が得られた．

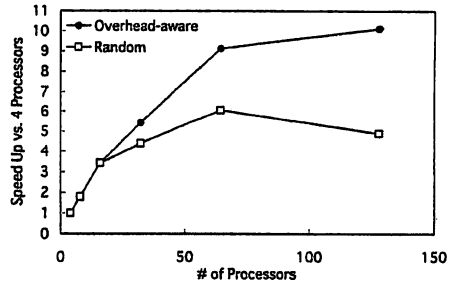
### 3.2 Rank 割り当て

NAS Parallel Benchmarks を用いて，提案した rank 割り当て手法の効果を調査した．各実行では表 1 のクラスタ B とクラスタ C のプロセッサを同数用いた．

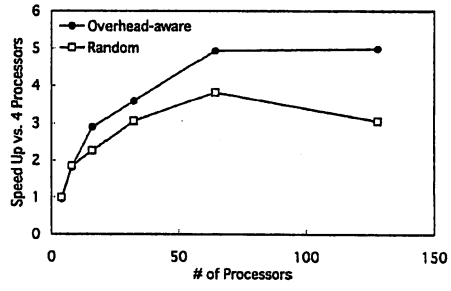
図 4 に BT, LU, MG, SP の性能を示す．Overhead-



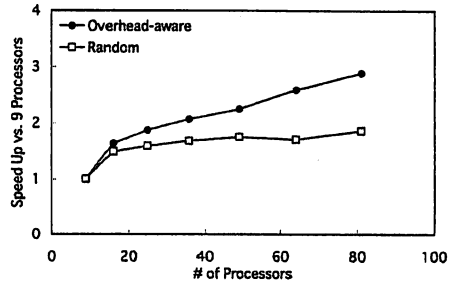

(a) BT Benchmark




(b) LU Benchmark




(c) MG Benchmark




(d) SP Benchmark

図 4 通信オーバヘッドを考慮した rank 割り当ての効果
Fig. 4 The Effect of Using a Rank-Process Mapping That Takes Communication Overhead into Account

aware と記された線は提案した rank 割り当て手法を用いた場合の結果で，Random と記された線はランダムな rank 割り当てを行った場合の結果である．Random では通信量が多い rank 対が通信コストの高いプロセス対（異るクラスタのプロセス対）に割り当てられることがよくあったのに対して，Overhead-aware ではそのようなことがなかったため，64 プロセッサ以上を用いた場合は Overhead-aware の方が Random より 60%から 100%性能が高かかった．

NPB は上記 4 つの他にもベンチマークがあるが，それらでは提案した rank 割り当て手法の効果はあまりなかった．EP はほとんど通信を行わないため，そもそも通信オーバヘッドが小さい．逆に，CG と IS は通信量が多いため，グリッド環境ではもちろんのこと，クラスタ環境でもスケールしない．

## 4. 関連研究

### 4.1 グリッド用メッセージパッシングシステム

MPICH-G2[6] と GridMPI[12] はグリッド環境用の MPI の実装であり，グリッド環境を意識した集合通信などを用いることによって高い性能を出す．しかし，ファイアウォールや NAT に対応しておらず，すべてのプロセスが互いに通信できる必要がある．

MPICH-MADIII[1] と Stampi[5] はプロキシを用いてクラスタ間のメッセージのルーティングを行うため，ファイアウォールや NAT がある環境でも動作する．しかし，プロキシを手動で設定する必要があり，MPI/GXP のように計算環境に適応しない．

### 4.2 適応的なメッセージパッシングシステム

Adaptive MPI[4] は各ノードの負荷と各プロセス対の通信量を実行時に収集して動的負荷分散を行うなどといった機能を持ったメッセージパッシングシステムである．計算環境に適応するという点では MPI/GXP に似ているが，Adaptive MPI はクラスタ環境用に設計されているため，ファイアウォールや NAT にも対応しておらず，リンクの速度の違いなども考慮していない．

## 5. おわりに

我々が開発している広域計算環境用のメッセージパッシングシステム MPI/GXP について説明した．MPI/GXP は計算環境が実行毎に変化するということを意識して，アプリケーション起動時にプロセス間の通信や通信量を測定することによってその実行に用いられる計算環境に応じた様々な性能最適化を行う．その中でも本稿では遅延を考慮した接続確立と通信オーバヘッドを考慮した rank 割り当てに注目して説明した．

実験では，5 クラスタ 256 プロセッサという環境で，提案した接続確立手法を用いることによって，全対全で接続を張ろうとするとルータが維持できるセッション数を超えてしまうという問題を回避することができた．また，別の実験では，提案した rank 割り当て手法を用いることによってランダムに rank 割り当てを行った場合と比べて NAS Parallel Benchmarks の性能が 60%から 100%向上した．

今後の課題としては以下が挙げられる．

- 本稿ではプロセス間の遅延を基に様々な最適化を行う方法を提案したが，より高い性能を出すためには帯域も考慮する必要がある．例えば，提案した接続確立手法は帯域を考慮せずに広域接続を制限するが，帯域が広い接続は制限せずに帯域が狭い接続を制限する方が良い．Rank 割り当てにおいても，通信コストに帯域を考慮する必要がある．今後は遅延と帯域の両方を同時に考慮する方法について考えていきたい．
- MPI/GXP は point-to-point 通信に関しては様々な最適化を行うが，多くのメッセージパッシングアプリケーションは集合通信を多用し，高性能な集合通信を必要とする．我々は既に広域環境用のブロードキャストとリダクションを提案しているので[9]，全対全などの他の集合通信に関する提案をしつつ，広域環境用の集合通信を MPI/GXP に組み込んでいきたい．

## 参考文献


1) Aumage, O. and Mercier, G.: MPICH/MADIII: A Cluster of Clusters Enabled MPI Implementation, *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 26–33 (2003).

2) Burkard, R. E. and Bonniger, T.: A Heuristic for Quadratic Boolean Programs with Applications to Quadratic Assignment Problems, *European Journal of Operational Research*, Vol.13 (1983).

3) der Wijngaart, R. F. V.: NAS Parallel Benchmarks Version 2.4, Nas technical report nas-02-007, NASA Ames Research Center (2002).

4) Huang, C., Zheng, G., Kumar, S. and Kale, L. V.: Performance Evaluation of Adaptive MPI, *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'06)*, pp. 12–21 (2006).

5) Imamura, T., Tsujita, Y., Koide, K. and Takemiya, H.: An Architecture of Stampi: MPI Library on a Cluster of Parallel Computers, *Proceedings of the 7th European PVM/MPI Users' Group Meeting*, pp. 200–207 (2000).
6) Karonis, N. T., Toonen, B. and Foster, I.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface, *Journal of Parallel and Distributed Computing*, Vol. 63, No. 5, pp. 551–563 (2003).
7) Koopmans, T. C. and Beckman, M. J.: Assignment Problems and the Location of Economic Activities, *Econometrica*, Vol. 25, pp. 53–76 (1957).
8) Message Passing Interface (MPI) Forum: Online at http://www.mpi-forum.org/.
9) Saito, H., Taura, K. and Chikayama, T.: Collective Operations for Wide-Area Message Passing Systems Using Adaptive Spanning Trees, *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing (Grid 2006)*, pp. 40–48 (2005).
10) Taura, K.: An Interactive Shell for the Grid Environment, *Proceedings of the International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems* (2004).
11) The NAS Parallel Benchmarks: Online at http://www.nas.gov/Software/NPB/.
12) 松田元彦, 石川裕, 鐘尾宜隆, 枝元真彦, 岡崎史裕, 鯉江英隆, 高野了成, 工藤知宏, 児玉祐悦: GridMPI Version 1.0 の概要, 並列／分散／協調処理に関するサマーワークショップ (SWoPP) , pp. 169–174 (2005).