MPI/SPにおけるクラスタ統合方式の設計

村山 和宏，落合 真一，山口 義一
三菱電機(株)

近年のハードウェア技術の急速な進歩に伴い、従来多数のDSPを組み合わせて行ってきた処理を高性能汎用プロセッサの組み合わせによる疎結合クラスタで実現するというアプローチがある。クラスタの採用によってプロセッサ間の通信に業界標準のプロトコルが使用可能となり、ソフトウェア資産継承が実現できる。本研究では、信号処理用計算機クラスタのプロセッサ間通信インタフェースにMPIを採用し、さらにクラスタの大規模化、多種通信路構成に対応するための機能拡張を実施した。本研究にて開発したMPI/SPでは、従来のメッセージパッシング機能のほか、大規模システムを構成する計算機群の分散管理、異種通信路クラスタ群の統合化を実現する。

The design of cluster-integration on MPI/SP

Kazuhiro Murayama, Shinichi Ochiai, Yoshikazu Yamaguchi
Mitsubishi Electric Corporation

Because of recent evolution of microprocessor performance, clustering of generic processors can be applicable to massive data processing. System using generic processors is more flexible than using DSPs, and it can adopt standard software and network communication. For those systems, we are developing middleware of inter-processor data exchange, which is called MPI/SP (Signal Processing). MPI/SP is based on standard MPI specification, but has new features for chaining of multiple clusters. In this paper, we describe design and specification of MPI/SP, which integrates multiple clusters into single MPI.

1. はじめに

暗号処理、音声・ビデオデータ処理、センサデータ処理は従来多数のDSP（Digital Signal Processor）を組み合わせて処理を行っていた。しかし、汎用ネットワークの高速化やプロセッサ技術の急速な進歩により、これらの演算装置を安価で高性能なクラスタで実現するというアプローチがある。クラスタでは、システム構成の柔軟性やスケーラビリティを実現することができる。

このような流れを受け、DSPで行われてきた信号処理をクラスタ上にて実現することを検討している。また、プロセッサ間の通信インタフェースにMPI（Message Passing Interface）を用いることにより、将来にわたるソフトウェア資産の継承実現を目指す。

信号処理向けの特殊なプロセッサ配置に対応するため、MPIの機能拡張を実施した。本研究で開発したMPI/SP（MPI for Signal Processing）では従来のメッセージパッシング機能や信号処理用計算機配置への対応、多種通信路の差異の隠

蔽を実現するだけでなく、システムの制御情報の交換を行い、システム全体を統合管理することが可能である。以下、MPI/SP の持つ機能であるクラスタ統合化の手法について述べる。

2.背景

2.1. ターゲットシステムの特徴

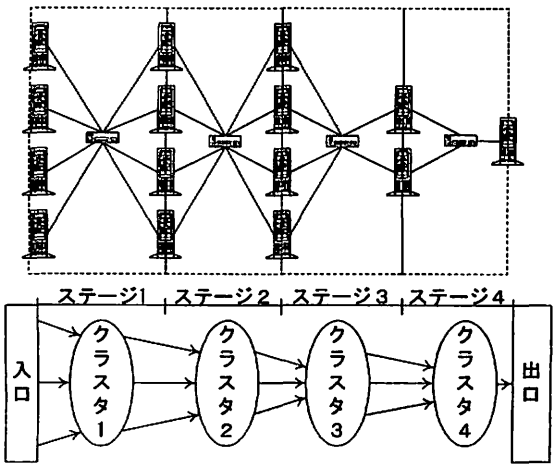

図 1：システム構成概要

図 1は、本研究においてターゲットとしたシステムの概要を示したものである。本システムは以下の特徴を持つ。

- **クラスタをチェーン状に連結した（Chain of Clusters）構成：**

  本システムでは、センサから送信される情報に対してパイプライン処理を行う。パイプラインの各ステージはクラスタ毎に独立しているため、図 1下に示すようにクラスタをチェーン状に連結した構成となっている。

- **多計算機によるクラスタ構成：**

  本システムでは数 100 台～1000 台程度の計算機を 10～20 台ごとにネットワークで接続し、クラスタ化している。

- **異種通信路を持つクラスタ：**

  本研究のパイプライン処理ではステージ毎に処理内容が異なる。そこで、高性能な処理を必要とするクラスタでは Fibre Channel、高性能処理が不要な場合には 100M イーサネットを使用するなど、要求性能によって通信路の異なるクラスタを構築している。

2.2. 本研究の課題

本研究では、ターゲットとしているシステム上でのメッセージ交換を行う手法として、MPI（Message Passing Interface）[1] を採用することにした。その理由は以下の通りである。

- 通信路の差異を隠蔽した通信が可能
- 新規ハードウェアに移行した場合のソフトウェア資産継承が可能
- 現在の計算機間メッセージ通信の主流

従来の MPI では全計算機を一括して管理するため、全計算機が同一通信路で接続されている必要があり、本研究のターゲットシステムには従来の MPI を適用することができない。ターゲットとするシステム上で MPI にてメッセージ交換を行うためには、以下の３点を実現する必要がある。

1. クラスタの境界を意識しないクラスタ間通信の実現
2. MPI による通信路ごとの分割管理の実現
3. MPI の枠組みを維持した API の拡張

2.3. 従来研究

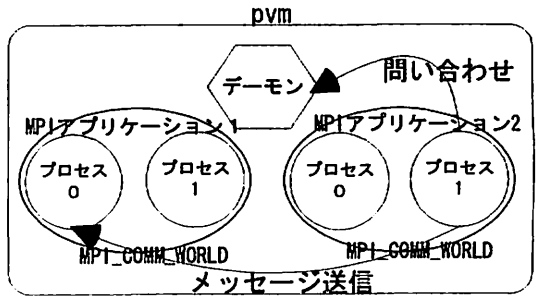

図 2：PVMPI の送信方法

本研究に関連して、例えば PVMPI[2] や MPI-GLUE[3] などでは複数のクラスタ間で連携を取り、異なる MPI アプリケーション間でメッセージ転送を実現している。PVMPI は、各 MPI アプリケーション上のプロセスを pvm デーモンに登録する。そして、通信時にはデーモンより送信先プロセスの識別子を取得し、それを使用する

ことによって異なる MPI アプリケーション間でのメッセージ交換を実現する（図 2）。

しかし、本研究で対象となるシステムでは全計算機間で互いに通信ができないため、1 つのデーモンではすべての計算機を管理することが不可能である。また、PVMPI ではデーモンに問い合わせてから通信を行うことにより、通信回数が増え、処理性能が低下する。

以上により、本研究では課題 1～3 を解決可能な MPI を新たに設計する。

3. MPI/SP 設計

3.1. MPI/SP の特徴

本研究では信号処理用計算機システム上でメッセージ通信を行うためのライブラリ MPI/SP（MPI for Signal Processing）を設計した。MPI/SP の特徴を以下に示す。

- **クラスタ統合化の実現：**
  各計算機の情報はクラスタ内で独立して持ち、クラスタ外には通知しないことにより、クラスタの独立性を保持する。その一方で、各クラスタの正常/異常に関する情報は全マスタで共有することによりクラスタを統合化する。
- **異種通信路クラスタ間通信の実現：**
  各クラスタは独立した MPI 環境を構築し、それぞれ異なるデフォルトのコミュニケータ（MPI_COMM_WORLD）を作成する。クラスタ間のメッセージ転送は、このコミュニケータを使い分けて行う。
- **API の拡張：**
  アプリケーションから MPI/SP による拡張機能を利用するため、コミュニケータを取得する関数や計算機のランクを取得する関数など、従来の API の拡張を行う。

以下の節でこの 3 点の設計内容について述べる。

3.2. システム統合化実現

3.2.1. ユーザによる初期設定

全クラスタの統合化を実現するため、各クラスタ、マスタに対して以下の設定を行う。

クラスタの設定：

MPI/SP ではそれぞれ独立したクラスタを統合管理するため、各クラスタに 0 から始まる番号（仮想クラスタ ID）を付与する。

マスタの設定：

- 各クラスタのマスタはクラスタゲートウェイ（複数のクラスタに属する計算機群）から選択する（図 3）。
- 各マスタは、自計算機が使用する通信ポートをあらかじめ決めておき、この通信ポートによってスレーブはマスタと通信を行う。

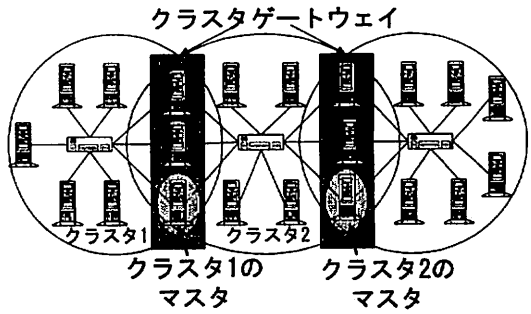

図 3：マスタ配置の例

- 各マスタはクラスタ構成情報テーブル（表 1）を持つ。クラスタ構成情報テーブルには、マスタ／スレーブの種別、計算機名、計算機が属するクラスタの仮想クラスタ ID を示す。

表 1：クラスタ構成情報テーブル

種別	計算機名	計算機が属するクラスタの仮想クラスタ ID
マスタ	pc0	1, 2
スレーブ	pc1	1
スレーブ	pc2	1
スレーブ	pc3	1

3.2.2.クラスタ統合の実現手順

3.2.1節に示したユーザからの情報をもとに、以下の 1.～5.の手順でクラスタ統合化を実現する。

1. **MPI グループの作成：**
   各クラスタのマスタはスレーブとコネクションを確立する。各スレーブは、マスタに「プロセス ID」「計算機名」「通信ポート」を示し

た表を送信することによって自プロセスが正常動作していることを通知し、マスタはスレーブの情報をもとに、全プロセスの「プロセスID」「ランク」「計算機名」「通信ポート」を示した表（プロセステーブル）を作成する（図 4）。

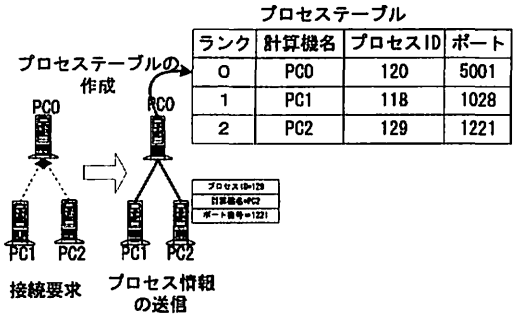

図 4：MPI グループ作成

2. 隣接クラスタの正常動作を確認：

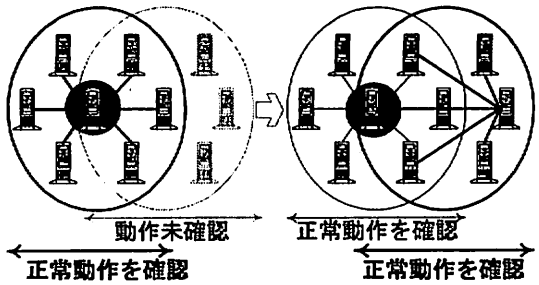

図 5：隣接クラスタの動作確認

各マスタがクラスタゲートウェイにあることを利用し、全てのマスタは、隣接クラスタが正常に動作していることを確認する（図 5）。

3. クラスタ情報のフォワード：

隣接しているクラスタが正常に動作していることを確認後、各マスタはクラスタが正常であることをパイプライン処理の「出口」に近いマスタに向けて送信する。情報を受信したマスタは、受信した情報に自クラスタが正常であるという情報を付加し、さらに「出口」に近いクラスタのマスタにフォワードする（図 6）。各マスタで3.の処理を行うことにより、パイプラインの「出口」にあるクラスタのマスタには全クラスタの情報が届く。

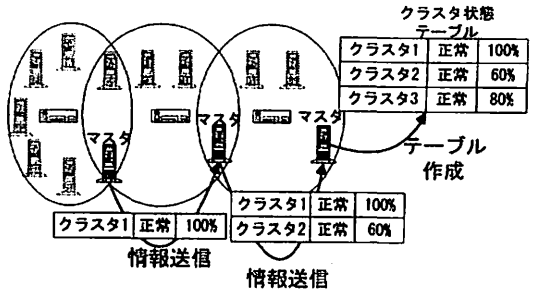

図 6：クラスタ情報のフォワード

4. 全クラスタ情報の配信：

パイプラインの「出口」にあるクラスタのマスタは、全クラスタの情報をまとめたクラスタ状態テーブルを作成し、このテーブルを「入口」に近いクラスタのマスタに通知する。テーブルを受信したマスタは、さらにパイプライン処理の「入口」に近いマスタに向けてテーブルを配信する。これを繰り返すことにより、全マスタがクラスタ状態テーブルを受信する。

5. スレーブにシステム正常動作の通知：

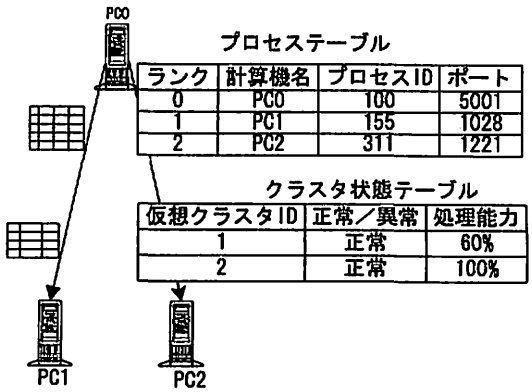

図 7：プロセステーブルの送信

4.でクラスタ状態テーブルを受信したマスタは、全クラスタが正常であることを確認後、全スレーブに1.で作成したプロセステーブルを送信することにより全クラスタが正常であることを通知する（図 7）。スレーブのプロセステーブル受信により MPI の初期化が終了し、各プロセッサ間、クラスタ間通信が可能となる。

3.2.3. 異常処理の実現

MPI/SP では、重大な異常、軽微な異常の定義、

対処方法を以下のようにする。

(1) 重大な異常

定義：クラスタの動作不可能

対処方法：クラスタ異常を全計算機に伝達してシステムを停止させる。

(2) 軽微な異常

定義：クラスタの部分異常

対処方法：クラスタの性能低下を全計算機に伝達し、処理を縮退させる。

以下、上記２つの対処方法について述べる。

(1). 重大な異常時のクラスタ異常伝達の実現：

図 8においてクラスタ３が異常である場合を例に、クラスタ異常の伝達方法を説明する。

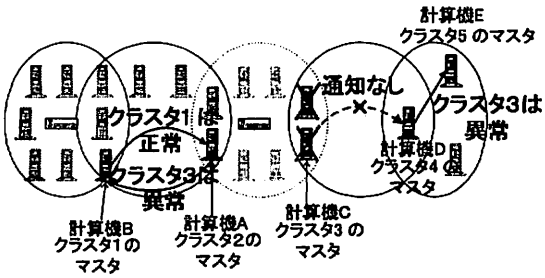

図 8：クラスタ異常情報の伝達

1. 「入口」側マスタへの異常通知：

3.2.2節の手順 2.により、マスタは隣接クラスタの異常を知ることができる。そこで、3.2.2節の手順 3.で「入口」に近いマスタからクラスタ正常通知を受け取ると、そのクラスタに向けてクラスタ３が異常であることを通知する。

2. 「出口」側マスタへの異常通知：

クラスタ４のマスタである計算機 D には、クラスタ３が正常であれば、3.2.2節の手順 3.によりクラスタ３のマスタからクラスタ正常通知が送られてくるはずである。計算機 D はこの通知を一定時間待ち、通知が来ないことにより異常を知る。計算機 D は、クラスタ５のマスタ（計算機 E）にクラスタ３の異常を通知する。

3. スレーブへの異常通知：

各マスタで 1.または 2.を行うことにより、全マスタにクラスタ３の異常を通知することができる。異常情報を受信したマスタは、同一クラスタ内の全スレーブにシステムの異常を通知し、システムを停止させる。

(2). 軽微な異常時のクラスタ情報伝達の実現：

処理縮退を実現するためには、隣接クラスタに自クラスタの処理能力を通知する必要がある。そこで3.2.2節の動作手順 3.、4.、5.を変更する。

3'. 自クラスタの処理能力値のフォワード：

各マスタは、表 1に示されたプロセス数と 3.2.2節の手順 1.で正常に接続したプロセス数から自クラスタの処理能力を知る。この値を、3.2.2節の手順 3.で自クラスタの情報の一部として加え、パイプライン処理の「出口」のクラスタに向けフォワードする（図 6）。

4'. 全クラスタの処理能力情報の配信：

パイプラインの「出口」にあるクラスタのマスタは、全クラスタの処理能力値をクラスタ状態テーブルに加える。そして、3.2.2節の手順 4 で各マスタがクラスタ状態テーブルを配信することにより、全クラスタのマスタは全クラスタの処理能力を得ることができる。

5'. 全スレーブへの処理能力情報の配信：

マスタは全クラスタに向けプロセステーブルだけでなく、クラスタ状態テーブルも送信する（図 7）。これらの処理により、各マスタは隣接クラスタの処理能力値を持つことができる。各プロセスはこの値に基づいて通信量・演算量を調整すれば、隣接クラスタの負荷を軽減させることができる。

3.3. 異種通信路間クラスタ間通信の実現

前述のように、クラスタゲートウェイの計算機にはコミュニケータ（MPI_COMM_WORLD）が複数個存在する。そこで、MPI 内部で MPI_COMM_WORLD を格納する配列を用意する。ユーザは MPI/SP の拡張 API 関数（後述）を用いてこの配列から通信を行うグループのコミュニケータを取得し、メッセージ通信を行う。

3.4. MPI API 関数の拡張

MPI/SP の拡張機能をアプリケーションが使用するため、以下の機能を実現する関数を追加した。

(1). コミュニケータの切替え機能

(2). クラスタゲートウェイ取得

1. MPISP_get_communicator
   機能：指定されたクラスタにおけるデフォルトのコミュニケータを取得する
   入力：仮想クラスタ ID
   戻り値：デフォルトのコミュニケータ
2. MPISP_get_clustergw
   機能：指定した２つのクラスタのクラスタゲートウェイを取得
   入力：クラスタ ID（２個）
   出力：クラスタゲートウェイのランク一覧
   戻り値：クラスタゲートウェイ数

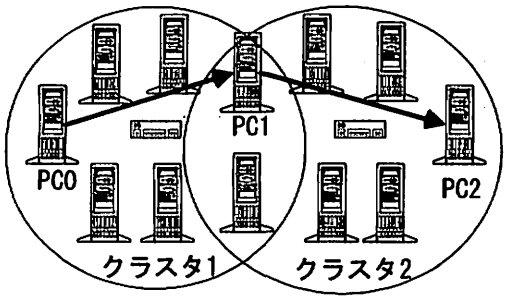

図 9：パイプライン転送例

```
int  gateways[10];
       PC0
comm=MPISP_get_communicator(1) ;
MPISP_get_clustergw(1,2,gateways) ;
MPI_Send(&a,1,.... gateways[0], comm,...);

       PC1
comm1=MPISP_get_communicator(1) ;
MPI_Recv(&a,1,....MPI_ANY_SOURCE,comm1,...);
MPI_Send(&a,1,....2,comm2,...);
comm2=MPISP_get_communicator(2) ;

       PC2
comm2=MPISP_get_communicator(2) ;
MPI_Recv(&a,1,....MPI_ANY_SOURCE,comm2...);
```

図 10：プログラム例

例えば図 9において、PC0 から PC2 へパイプライン転送を行うには図 10のようにプログラムを記述すればよい。

4.まとめ

本稿では、信号処理用大規模クラスタ上で MPI によるメッセージ交換を可能とするため、従来の MPI の拡張を行った。実現内容は以下の通り：

1. クラスタの境界を意識しないクラスタ間通信の実現…全マスタ間でクラスタに関する情報を交換することにより、隣接するクラスタ間での処理の連携を可能にした。また、複数のコミュニケータの使い分けにより、特別な操作を行わず、従来の MPI の枠組みを使用したクラスタ間通信を実現した。
2. MPI による通信路ごとの分割管理の実現…クラスタ内計算機の情報はクラスタ内で独立して持ち、他には通知しないことにより、各クラスタの独立を実現した。
3. MPI の枠組みを維持した API の拡張…API 関数の拡張を最低限（２個）にすることにより、MPI の枠組みの逸脱を最低限に留めた。

今後、この設計を実システムに適用し、評価を行う予定である。

参考文献

[1] Message Passing Inferface Forum. MPI : "A Message-Passing Interface Standard" Jun 12, 1995.

[2] Graham E. Fagg and Jack J. Dongarra. PVMPI: An integration of the PVM and MPI systems. Calculateurs Paralleles, 2, 1996.

[3] R. Rabenseifner. MPI-GLUE: Interoperable High-Performance MPI Combining Different Vendor's MPI Worlds, Proceedings of the Euro-Par'98, 4th International Euro-Par Conference, D.Pritchard, J. Reeve (ed.), Southampton, UK, Sept. 1998, LNCS 1470.

[4] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek and Vaidy Sunderam. PVM : Parallel Virtual Machine. MIT Press, 1994.