JP0050186

JAERI-Data/Code
2000-002

異機種並列計算機間通信ライブラリ：Stampi－利用手引書　第二版

2000年2月

今村俊幸・小出　洋・武宮　博

日本原子力研究所
Japan Atomic Energy Research Institute

本レポートは、日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは、日本原子力研究所研究情報部研究情報課（〒319-1195　茨城県那珂郡東海村）あて、お申し越し下さい。なお、このほかに財団法人原子力弘済会資料センター（〒319-1195　茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布を行っております。

This report is issued irregularly.
Inquiries about availability of the reports should be addressed to Research Information Division, Department of Intellectual Resources, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 〒319-1195, Japan.

編集兼発行　日本原子力研究所

JAERI－Data/Code 2000－002

異機種並列計算機間通信ライブラリ: Stampi —利用手引書 第二版

日本原子力研究所計算科学技術推進センター
今村 俊幸・小出 洋・武宮 博

(2000年1月6日 受理)

異なる並列計算機を任意に組み合わせ計算を行うために, MPIを通信のための単一のインターフェイスとする異機種並列計算機間通信ライブラリ Stampiを開発した. StampiはMPI2のインターフェイス仕様に基づき, 異なる計算機への動的なプロセス生成とMPIのセマンティクスを保ったまま外部との通信を可能としている. Stampiの主な特徴は次のようにまとめられる. 1) 外部通信と内部通信とのプロトコル自動選択機能, 2) 通信中継モジュールによるメッセージ中継機能, 3) 動的なプロセス生成, 4) 異なる2種類の通信路 (Master/Slave, Client/Server) の動的形成方法サポート, 5) Java Appletとの通信. 従来, ベンダー提供のライブラリでは提供されていなかったこれら外部へのプロセス生成と通信の機能等を実現し, これにより異機種の並列計算機を同時に用いた計算が可能となった. 現在Stampiは計算科学技術推進センター所有の並列計算機群COMPACSの5台の並列機とグラフィックサーバ, 他に8種の並列計算機に実装 (計14機種) され, それら計算機間での任意の通信を可能としている. 本稿では, Stampiの利用方法を中心に報告している.

日本原子力研究所: 〒153–0061 東京都目黒区中目黒2-2-54

JAERI－Data/Code 2000－002

Stampi : A Message Passing Library for Distributed Parallel Computing — User's Guide, Second Edition

Toshiyuki IMAMURA, Hiroshi KOIDE and Hiroshi TAKEMIYA

Center for Promotion of Computational Science and Engineering
Japan Atomic Energy Research Institute
Nakameguro, Meguro-ku, Tokyo

(Received January 6, 2000)

A new message passing library, Stampi, has been developed to realize a computation with different kind of parallel computers arbitrarily and making MPI(Message Passing Interface) as an unique interface for communication. Stampi is based on the MPI2 specification, and it realizes dynamic process creation to different machines and communication between spawned one within the scope of MPI semantics. Main features of Stampi are summarized as follows: (i) an automatic switch function between external- and internal communications, (ii) a message routing/relaying with a routing module, (iii) a dynamic process creation, (iv) a support of two types of connection, Master/Slave and Client/Server, (v) a support of a communication with Java applets. Indeed vendors implemented MPI libraries as a closed system in one parallel machine or their systems, and did not support both functions; process creation and communication to external machines. Stampi supports both functions and enables us distributed parallel computing. Currently Stampi has been implemented on COMPACS (COMplex PArallel Computer System) introduced in CCSE, five parallel computers and one graphic workstation, moreover on eight kinds of parallel machines, totally fourteen systems. Stampi provides us MPI communication functionality on them. This report describes mainly the usage of Stampi.

Keywords : Distributed Parallel Computing, Communication Library, MPI, Dynamic Process Creation, External Communication.

目　次

Contents

1 はじめに

計算機ならびにネットワーク技術の発展に伴い, ベクトル並列計算機, スカラ並列計算機等種々のアーキテクチャの並列計算機を同時に利用し高速で大規模な科学技術計算が可能となってきている. 原研においても計算科学技術推進センターに 5 台の並列計算機, 東海, 那珂, 関西研究所にそれぞれ高性能の並列計算機を計 10 数台保有しており, それら並列機は計算物理分野を中心として所内外の研究者によって利用されている (図 1.1 参照).

従来, これら並列計算機は単体としての利用がなされてきたわけであるが, 各種並列計算機にはアーキテクチャに応じて有効かつ効率的に働く問題対象があることが過去の研究例から指摘されている (あえて文献を引用することもないであろう). また, 高速ネットワークの普及に従い, これら複数の並列計算機資源 (ここでは単に計算機だけではなく周辺の機器資源やデータベース等の情報資源も含めて) を連携させることで次に示すような利点があるともいわれている.

(1) 流体などの大規模な問題で連続アドレスへのアクセスが主要部分を占める場合にはベクトル計算機が効率的である. また, 有限要素法などに多く見られる間接参照や局所的なメモリ領域へのアクセスが多い場合にはスカラ計算機が有効である. 原研が開発中のプラズマシミュレーション [1] や翼とその回りの流体の連成計算 [2] では, プログラムの一部がベクトル計算機上で効率がよく別の部分ではスカラ計算機上で効率がよいという報告がなされている. このようなアーキテクチャに起因する性能格差のため単一の並列計算機上で実行した場合には必ず非効率な部分が生じ, 全体の実効性能を下げてしまう. この様なプログラムの各部分を効率のよい並列計算機上で計算することにより, 単一並列計算機以上の高速な計算が可能となる.

(2) 同様に複数の計算機を同時利用することで, 従来得ることができなかった大規模なメモリ空間を利用できる可能性が生じる. トカマクプラズマ計算では, 詳細な解析を行うために並列計算機単体で利用可能なメモリ容量の数倍のメモリが必要だが, 複数の大容量メモリを搭載する並列計算機を利用することで要求されるメモリ容量に近いシステムを実現することができる [3].

また, メモリ容量は左程必要としないのだが, 大量の独立したタスクを抱える問題 (例として, 分子軌道計算 Hartree-Fock 法での Fock 行列生成部分など) では, できるだけ利用可能なプロセッサエレメントを使って高速に計算したいという要求が強く, 幾つかの分散利用環境の開発が行われている [4, 5]. こういった大量資源を必要とする問題に対してこそ, 複数システムの同時利用が有効であるといえよう.

(3) 大規模な数値計算を行う現在では当たり前となったデータの可視化作業においても複数の計算機資源を利用する. データの高速な可視化が可能なグラフィックスワークステーションを用いる場合, 数値実験には高速な並列計算機サーバを使用し可視化にグラフィックスワークステーションを用いることが普通であろう. この場合, 大量 (ギガバイトに及ぶ) データをグラフィックスサーバ－並列計算機サーバ間で逐次やり取りしなくてはならない. 動画のように, フレーム単位でリアルタイムで転送しなくてはならない場合も多く存在する [6, 7](引用文献の他多数の報告がある).

これらのプログラム間で自動的にデータをやり取りすることによる, 可視化作業の即時性またはリアルタイム性向上が可能となるであろう.

(4) 原研が開発している緊急時放射線放出源推定システム [8] はモニタリングポイントでの観測データと気象庁の気象データをもとに放射線源を推定する. この問題の場合, モニタリング装置, 観測データのデータベース, 放出源推定プログラムを計算する計算サーバが地理的に分散しており, 何らかの方法でデータ, プログラムを結合させなくてはならない.

更に, JT-60 観測システムなどは, JT-60 の運転とデータベースサーバ, データ解析サーバ等全く性質の異なる装置を結合して初めて有用なデータが得られる問題である.

これら 2 例に代表されるように, 特定のサイトにのみ存在する機器を利用しなくては目的を達することができない問題に対して, システム構築やアプリケーション実行を容易に支援する枠組みが提供できることであろう.

この様に複数の計算機資源の中からハードウェア的に効率的な組合わせを選ぶ手法が議論されるなかで, その組み合わせを時々に応じて変更して一つの仮想計算機を構築する計算手法 (またはその計算機上で計算を行うこと) が提案され, その様な計算手法はメタコンピューティングと呼ばれている. メタコンピューティングは近年非常に注目されている計算パラダイムであり, 様々な機関で盛んに研究が進められている [9, 10, 11]. ここで, メタコンピューティングを行うためには, その定義から複数の計算機を連携させることが必要不可欠であることは容易に想像される. それはネットワーク上の複数の計算機相互間でデータをやり取りするための通信手段を確保しなくてはならないことを意味しており, 実用のためには, プログラム開発者が直接利用する通信インターフェイスが汎用的な通信ライブラリとして使用する全ての機種で利用可能でなくてはならないことに帰着可能である.

現在単一並列計算機上で動作する通信ライブラリは NX[12], MPL[13], p4[14], PVM[15] など多数提供されている. その中で, 事実上業界標準といえる MPI2(Message Passing Interface の第二版仕様, 以下 MPI と表記する) [16, 17] が並列計算のための最も一般的な通信ライブラリである. MPI は数多くの並列計算機上で実装されており MPI を用いて開発されたプログラムは移植性が非常に高いことが知られている. しかしながら異なる並列計算機を連携させて計算を行う場合, それら異機種並列計算機間での通信が既存 (ベンダー提供の意味で) の MPI には備わっていないことが大きな問題として浮かび上がってくる. 結果, 異機種の環境下での連携を行うために, 利用者自身がそれを支援するための通信ライブラリを新たに準備し, MPI とは別の通信手順をプログラム中に記述する必要が生じる. 既存の並列プログラムも多くが MPI で記述されており, それらプログラムを改編することなく MPI 単一の通信セマンティクスを保った形で異機種並列計算機を用いた計算が行えることが望ましいといえよう.

ベンダー提供 MPI が単一機のみの通信機能を提供する一方で, MPICH[21], MPICH-G[22], LAM[23], PACX-MPI[24] などのフリーソフトウェアで異機種上での通信機能を提供するものが存在する. しかしながら, これらは通信機能に幾つかの制限が設けられているとともに並列計算機のプライベートネットワークアドレッシングの問題を解決しておらず実用上の問題を少なからず残しているといえる. 我々が想定する異機種環境下での問題点は以下のものである.

- interoperability(複数計算機間での通信ライブラリを用いての相互の操作 / 通信可能性)
- multi-protocols (並列計算機筐体内 / 筐体外における通信の最適制御)
- dynamical process control and management of the universe
- resolving private-global IP addressing(Relay もしくは Router プロセスの対応)
- data conversion(異なるハードウェアでの浮動小数点 format, Endian の自動変換)
- security(データ盗用, 資源不正利用の防止など)
- performance-intensive networking(通信速度を意識したルーティングやバッファ制御)

そして, それらに対する MPI 実装系の対応状況を簡単な表にしてみたものが表 1.1の様になる (本論分で我々が開発したライブラリ Stampi に関しても表中に付記した). 本報告書ではこれら項目の必要性, ま

表 1.1: 各 MPI 実装における異機種分散対応の状況

	MPICH	MPICH-G	LAM	PACX	Stampi
interoperability	○	○	○	○	○
multi-protocols	△[1]	△[1]	△[1]	○	○
dynamical process control	×	×	○	×	○
management of the universe	△[2]	△[2]	○	△[2]	△[3]
Router process	×	×	×	○	○
data conversion	○	○	○	○[4]	○
special security	×	○[5]	×	×	×
performance intensive	○	○	○	○	○

[1] 同一アーキテクチャ間での内部高速通信と TCP/IP の共用が可能．もしくは共有メモリ内通信と TCP/IP の共用が可能
[2] MPI プログラム起動時にのみ総プロセス空間 (UNIVERSE) の形成が可能
[3] MPI プログラム起動中にサブプロセスの動的生成と総プロセス空間 (UNIVERSE) のマージが可能
[4] 一部データ圧縮も行う
[5] 一部 LDAP(Lightweight Directory Access Protocol) によるセキュリティ管理

た将来どの様な形になっているべきかの議論は避けるが，いずれにしてもこれらを意識した開発が現在の主流となっている．当然，並列分散計算という手法が広く認識されるに従い，これらの問題点は解決されつつあるが．実用化において解決すべき点は多く，機能的にはまだまだ不十分であるといわざるを得ない．

このような現状の計算機環境と通信ライブラリ環境を背景として，我々は MPI2 を通信のための単一ユーザインターフェイスとして異なる並列計算機を同時利用可能とする異機種並列計算機間通信ライブラリ Stampi を開発した [18, 19, 20]．Stampi は異機種利用環境下で必要とされる，異機種間での最適な通信路およびプロトコルの選択，またプロセス生成手法に関する多くの点に焦点を絞った設計および実装となっている．また，本書で報告する Stampi 第二版では FORTRAN, C で開発されたプログラムに加えて，Java[26] で記述された Java Applet コンポーネントとの間の通信を可能とする Stampi/Java 拡張を行っている．Java 言語への拡張は MPI2 仕様にも存在しない独自仕様であるが，プラットフォームを問わない Java からバックエンド上のさまざまなアプリケーションにアクセスできる拡張は画期的なアイデアといえよう．

本報告書は従来 MPI を利用し並列プログラムを作成してきた利用者を対象に，Stampi の利用法を中心に編纂したものである．本編は平成 10 年 11 月刊行の Stampi ユーザーズガイド第一版 [20] をもとにし，平成 11 年 11 月現在開発時点での最新機能を反映させた第二版である．以下に本報告書の内容を簡単に列挙する．2 章では Stampi の概要について説明する．3 章では Stampi 利用方法として，利用者セットアップならびに各種コマンド群について説明する．4 章では異機種並列計算の基本プログラミングとしてマネージャー / ワーカープログラムの例を挙げて解説する．また基本的なライブラリ関数の用例についても例示する．付録には Stampi ならびに Stampi/Java の全ライブラリインターフェイス，COMPACS での各種設定例などを付記した．

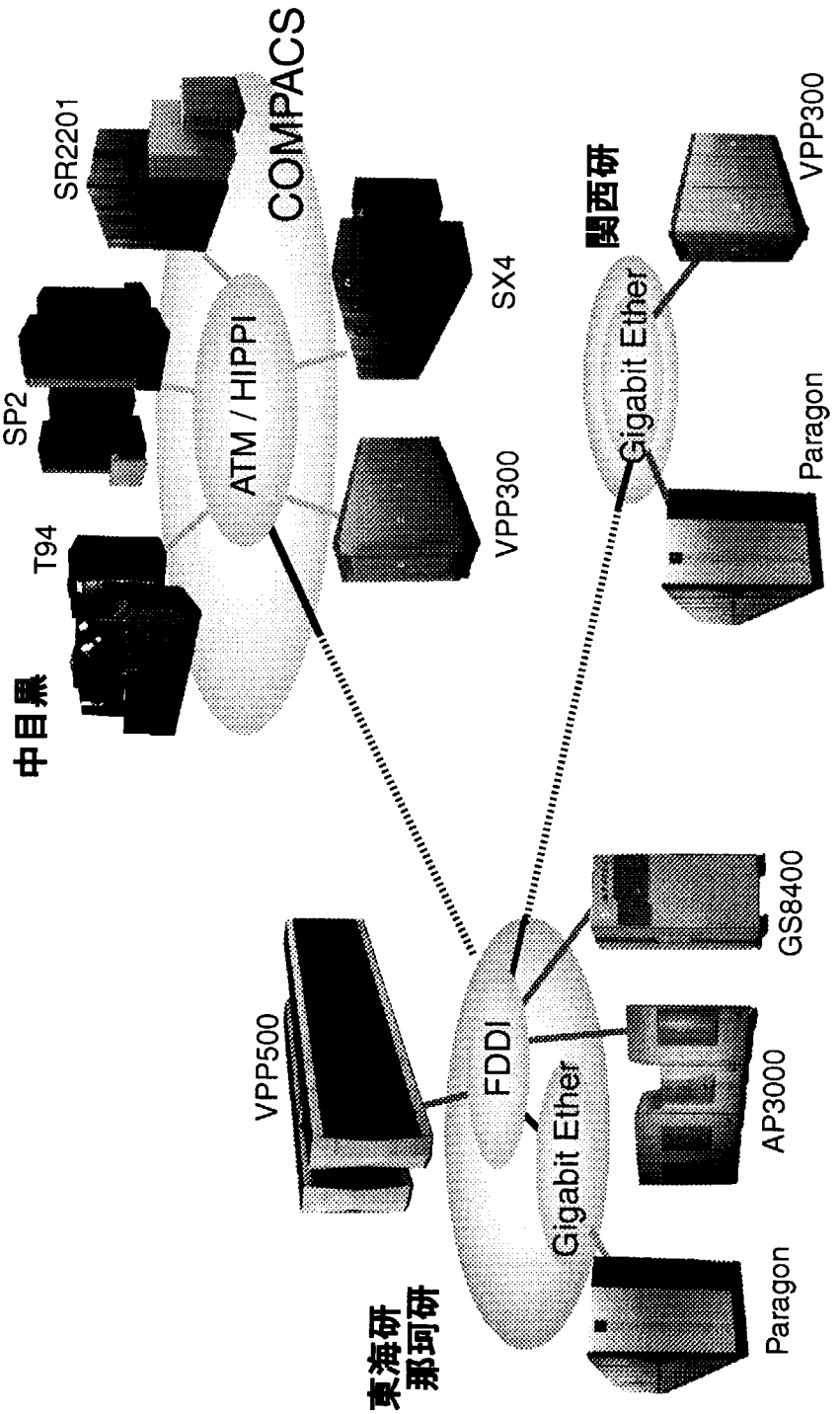


図 1.1: 日本原子力研究所 並列計算機の概要 (1999 年 12 月 現在)

2 異機種並列計算機間通信ライブラリ Stampi

2.1 Stampi システム構成

Stampi のシステム構成を, 図 2.1に示す. Stampi は, MPI のユーザインターフェイスを用いて異なる並列計算機で稼働する利用者プログラム間で通信を行う機能を提供する. 利用者は, Fortran(77/90) 言語または C 言語で記述した利用者プログラムをコンパイルし, 本システムが提供するライブラリとリンクすることで, 実行形式の利用者プログラムを作成する. 利用者プログラムは, プログラムを実行するそれぞれの計算機に用意しておく. その後, 本システムが提供する異種 MPI 起動コマンドを使用して利用者プログラムの実行を行うことにより, 複数の計算機間で MPI 通信を用いた計算が可能になる.

Stampi による単一 MPI プロセス内 (並列計算機内) の通信はベンダー提供の MPI ライブラリを利用し, 外部との通信には TCP/IP を利用する. 場合によっては機種にまたがる MPI プロセス同士の直接通信が不可能となるため, ルーターを適時置きデータの中継をする. 通信が並列計算機の内部であるか外部に対してであるかは, MPI の通信セマンティックスである (コミュニケータ, ランク) 対の指定を調べることによって決定することができる. MPI のセマンティックスでは, ある MPI プロセスが通信可能な MPI プロセスはプロセスが知るコミュニケータに属するもののみに限定され, かつコミュニケータの生成方法が集合通信と呼ばれる手法で行われるため全てのプロセスがコミュニケータに関する情報を知っている仕様になっている. Stampi もベンダー提供の MPI ライブラリが持つ情報に加えて, 外部通信に関する通信情報を管理することで 通信経路の内部 / 外部の判断機能を実現している.

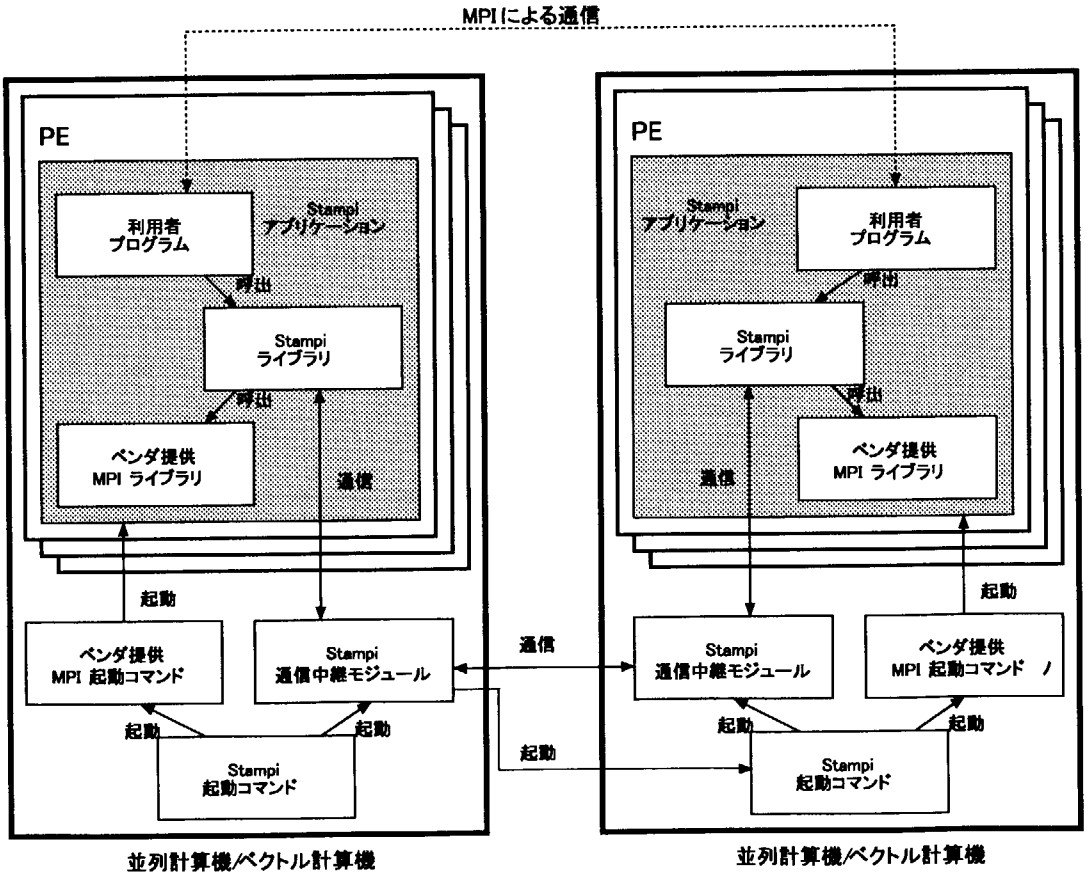


図 2.1: Stampi システムの構成

図 2.1にあるように, Stampi はシステム的には wrapper の実装になっている; つまり, 利用者プログラムは Stampi ライブラリ API(Application Programmer Interface) の皮 (wrapper) の部分を呼び出し, 先に述べたコミュニケータの管理情報をもとに内部 / 外部通信を判別する. 内部通信であればベンダ提供の MPI ライブラリを呼び出し, 外部通信であれば通信中継モジュールを中継点として外部計算機へデータを通信する. このとき, 外部プロセスとの通信には全てに共通なプロトコルである TCP/IP を用いている. 受信側では送信側と逆のプロセスを経て, 送信側から送られてきた通信データを利用者の指定に即した通信バッファに格納する.

尚, Stampi では複数の計算機で稼働する利用者プログラム間で MPI による通信路を確立する手法として, Master/Slave 型の接続 (一方の計算機の利用者プログラム:Master から, 他方の計算機の利用者プログラム:Slave を起動したのち通信路を確立する方式) と, Client/Server 型の接続 (個別の計算機上で独立して起動された利用者プログラム間で, 通信路を確立する方式) をサポートしている. これら, 通信路の確立をプログラム上で実現する方法は, 後述する 4 章で具体的にプログラミング例を示して紹介しているのでそちらを参考にされたい.

Stampi システム構成紹介の最後として, ここで通信中継モジュールに関して若干の説明を行う. 我々が Stampi のターゲットとしている機種は複数の並列計算機が集結した並列計算機クラスタである. 通常, 並列計算機個々には 1 つ以上のネットワークカードが接続されており, そのネットワークカードに設定されたネットワークアドレス (IP アドレス) に対して一般利用者がセッションを張ったり ftp 等で通信を行う. しかしながら, 並列計算機の内部にある並列計算に利用されるプロセッサ群には外部のネットワークと直結するネットワークカードを持たないことが普通である. これには, 幾つかの理由があるが第一に考えられる点として, 運用上の IP アドレスの枯渇防止が挙げられる. また他に, 内部のプロセッサが IP での通信をサポートしないなど OS 上の制限やハードウェア上の制限などが考えられる. このようなインターネットとは隔離され, 独自に運用されたネットワークアドレスで管理されたネットワークはプライベートアドレスと呼ばれ LAN 運用ではしばしば用いられる.

プライベートアドレスは運用者の責任の元自由にネットワーク空間を構築できる点が魅力的であるが, インターネットからは直接識別できないアドレスであるために, 何らかの中継プロセスを経ない限り, 任意のマシンとの通信を行うことができない. 並列計算機ではこの様な, プライベートアドレッシングが並列計算機筐体内で行われており, 異機種並列計算機間通信には何らかの中継操作が必要となる.

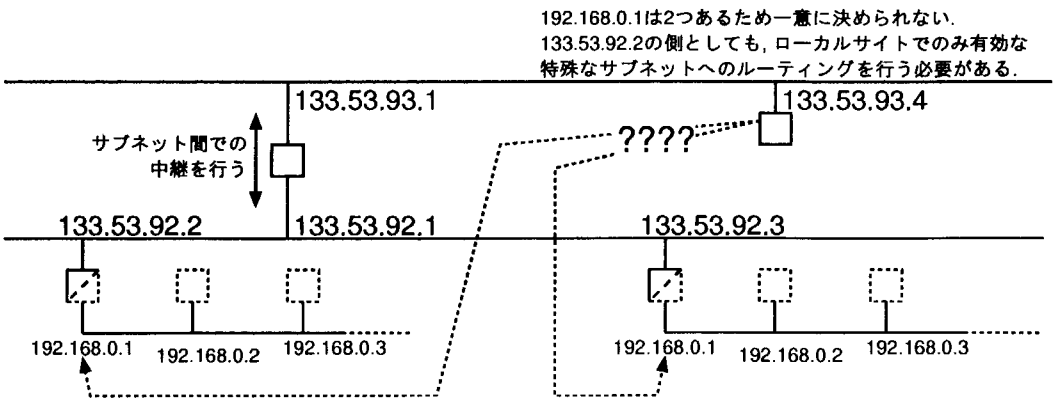


図 2.2: プライベートアドレスの例

Stampi では, このプライベートアドレス, グローバルアドレスに係わらず任意のプロセス間で最適な通信を行うために独自の通信中継モジュール (以下ルータと呼ぶ) を利用している. このルータの利用例を図 2.3に示す. この図の場合, 並列計算機 A,B 両者がプライベートアドレスであるために両サイドにルータを起動しているが, グローバルアドレスで運用される場合には直接の通信が可能であるのでグローバルアドレスで運用されている側の並列計算機にはルータは起動されない.

さらに, このルータをネットワークの状況やアプリケーションの性格に応じて複数起動し, マルチネットワーク (複数の経路を同時に使う通信) が可能となる (図 2.4).

この様に, Stampi では間接通信と直接通信の両者をサポートするためのルータをサポートし, さらにルータは通信性能を意識した通信路形成のルーティングをも可能としている. この特徴は, 序章で紹介した幾つかの MPI 実装系にはない独自の機能である.

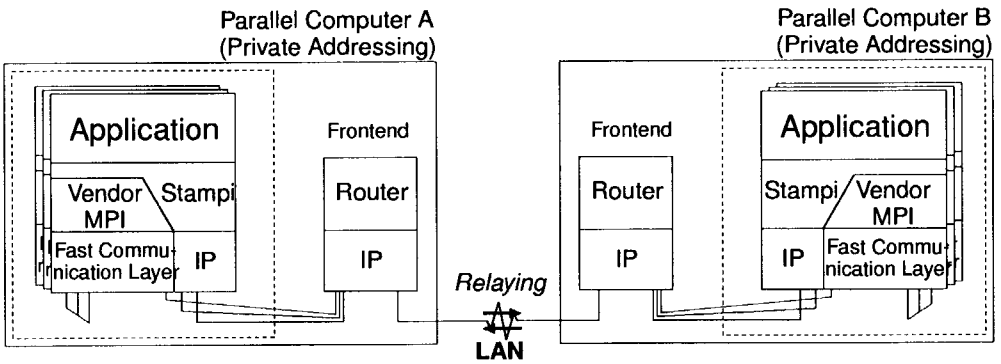


図 2.3: 通信中継モジュール (ルータ) の例

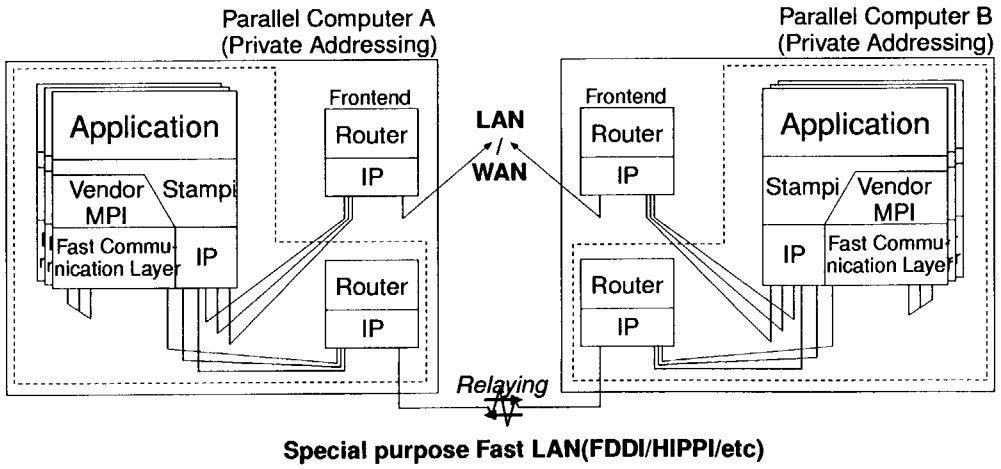


図 2.4: 複数通信中継モジュール (ルータ) の使用例

2.2 Stampi/Java システム構成

Stampi/Java は, Stampi の機能を WWW ブラウザ上で稼働する Java アプレットに拡張するためのコンポーネントである. Stampi/Java のシステム構成を図 2.5に示す. Stampi/Java では, WWW ブラウザ上で稼働する利用者作成の Java アプレット (以下, Stampi/Java アプリケーションと呼ぶ) との間での MPI 通信をサポートする.

Stampi/Java アプリケーションは、本システムが提供する Stampi/Java ライブラリを使用するように記述された利用者作成の Java プログラムであり、 WWW サーバ計算機の所定のディレクトリに格納しておくことによって、 WWW ブラウザに (Stampi/Java ライブラリと共に) ローディングされて実行することができる。

一般に Java アプレットは Java の制約により、 WWW ブラウザ計算機としか通信することができない。このため Stampi/Java では、 Stampi/Java 通信中継モジュールを WWW サーバ計算機で実行し、 WWW ブラウザで稼働する Stampi/Java アプリケーションとバックエンド計算機で稼働する Stampi アプリケーションとの通信を中継することによって、任意の並列計算機・ベクトル計算機との通信を可能にする。 Stampi/Java 通信中継モジュールは、 Stampi/Java ライブラリの初期化時にシステムが自動的に起動するため、利用者はその存在を意識する必要はない。

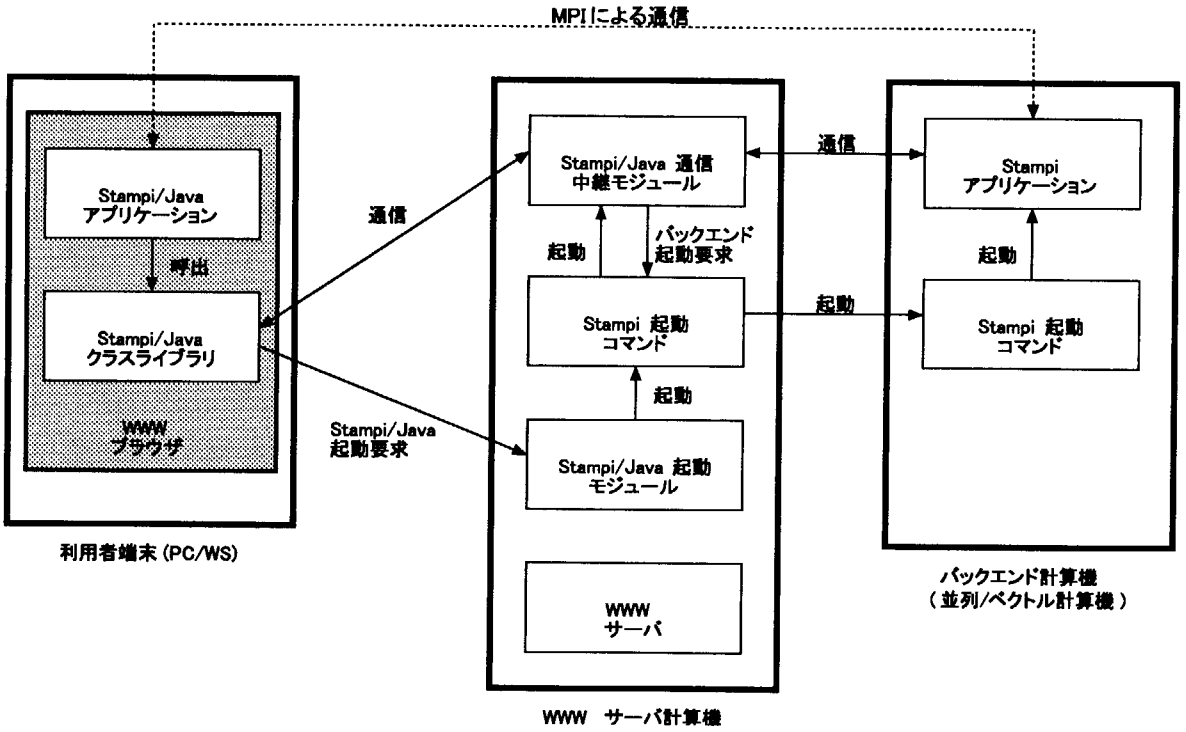


図 2.5: Stampi/Java システムの構成

2.3 Stampi が対象とする並列計算機

Stampi は原研計算科学技術推進センター (以下 CCSE) に設置された以下に示す 5 台の並列計算機とグラフィックスワークステーション (以下 COMPACS) を対象に開発されている. ただし, それ以外の高性能計算機への移植を考慮してライブラリの内部構造は計算機に依存する部分と非依存の部分を明確に切り分けた設計となっている.

実装並びに運用面から Stampi の利用可能性を言及するならば, Stampi は MPI での中核となる部分

は既存の MPI ライブラリを利用することが前提になっており，また異機種間の認証は UNIX での rsh コマンドを利用する．従って，Stampi は一般的に UNIX[1] と呼ばれる OS で運用されている並列計算機を対象として，ベンダーもしくは MPICH 等のフリーの MPI が実装され，なおかつ rsh などの遠隔利用コマンドを制限しないポリシーの元で運用されている計算機 (間) で利用可能といえる．

現在，動作確認されている機種に関して，OS, Fortran, C, MPI それぞれの組合わせを以下に示す．

(1) 日立 SR2201
- OS: HI-UX/MPP(02-03-/D)
- Fortran: 最適化 Fortran90(02-06-/C)
- C: 最適化 C(02-04-/A)
- MPI: MPI( )

(2) 富士通 VPP300
- OS: UXP/V(V10L20)
- Fortran: Fortran90/VP(V10L10)
- C: C/VP(V10L10)
- MPI: MPI(V11L10)

(3) NEC SX-4
- OS: SUPER-UX(8.1)
- Fortran: FORTRAN90/SX(8.1)
- C: C/SX(8.1)
- MPI: MPI/SX(8.1)

(4) IBM SP
- OS: AIX(4-1.4)
- Fortran: AIX XL FORTRAN(3-2)
- C: AIX C(3-1)
- MPI: AIX MPI

(5) SGI CRAY T94
- OS: UNICOS(10.0-0)
- Fortran: CF90(3.2-0)
- C: Std C(6.2-0)
- MPI: MPT(1.3-0&mpi.3.1-2)

(6) SGI Onyx2
- OS: IRIX64(6.2)
- Fortran: MPISpro Fortran90(7.2.1)
- C: MPISpro C(7.2.1)
- MPI: MPICH(1.1.2)

(7) 富士通 AP3000
- OS: Solaris2.6
- Fortran: 富士通 Fortran90(2.0.3)
- C: 富士通 C(2.0.3)
- MPI: 富士通 MPI/AP

[1]UNIX は The Open Group の登録商標である

(8) Intel Paragon
- OS: OSF/1(1.0.4 R1_4_7)
- Fortran: Paragon Fortran77(R5.0.3)
- C: Paragon C(R5.0.3)
- MPI: MPICH NX version

(9) IBM SP
- OS: AIX
- Fortran: AIX XL FORTRAN
- C: AIX C
- MPI: MPICH(1.1.2)

(10) DEC Alpha
- OS: DEC OSF/1
- Fortran: DEC OSF/1 Fortran90
- C: DEC OSF/1 C
- MPI: MPICH(1.1.2)

(11) SGI Power Challenge/Origin 2000
- OS: IRIX64(6.2)
- Fortran: Rangnorok Compiler(6.2)
- C: MIPSpro C
- MPI: SGI MPI(3.0)

(12) SUN ワークステーション
- OS: Solaris2.x
- Fortran: Sun Fortran90
- C: Sun C/gcc
- MPI: MPICH(1.1.2)

(13) HP ワークステーション
- OS: HP-UX(B.10.20)
- Fortran: HP Fortran90
- C: HP C
- MPI: MPICH(1.1.2)

(14) PC cluster
- OS: Linux/FreeBSD for Intel x86
- Fortran: f77/g77/ 市販 Fortran90
- C: gcc
- MPI: MPICH(1.1.2)

上記の, 計算機は現存する並列計算機の代表的なものを全てカバーしており殆んど全てのワークステーション, SMP, パーソナルコンピュータ (クラスタなども含む) への Stampi 移植は容易と考えられる. 未対応の 機種 /MPI に対してもプロファイリングインターフェイスと Fortran における幾つかの構造体 (`MPI_Status` や `MPI_Info` など) の仕様, 並びに NQS 等のバッチ制御方式部分のインターフェイスが明らかになっていれば, 問題となる部分はないと著者らは認識している.

3 Stampiの利用方法

異機種並列計算機間ライブラリStampiはCCSEに設置されたCOMPACS上で実装されたものであるが, 他機種への移植等も考慮して本節の利用方法については特にCOMPACSに限定しないで一般的な説明を行う. COMPACSの6台の並列計算機個々に限定した設定方法などは巻末の付録D, Eに掲載する.

3.1 Stampiを利用する前に (利用者セットアップ)

Stampiは通信ライブラリのみならず, これまで各並列計算機で異なっていたコンパイルコマンド, コンパイラのオプション指定やライブラリの指定方法, 並列プログラム実行コマンドを統一したStampi用異機種共通コマンド群を用意しており, 複数の並列計算機間で殆んど機種依存のないプログラム開発が可能となっている. これらStampi用コマンド群を利用するためには各並列計算機上で適切なpathを設定しなくてはならない. インストール時の設定によって異なるが, デフォルトでは/usr/local/stampi/bin (現在のCCSEでは/home01/g0186/j3051/stampi/bin) に実行コマンド類が格納されているので, .cshrcなどにpathを追加する設定を行うとよい[2].

- Cシェルを利用する場合 (.cshrc等に)
  set␣path=(Stampi実行ファイル群が格納されているディレクトリ␣$path)
- Bornシェルを利用する場合 (.profile等に)
  PATH=Stampi実行ファイル群が格納されているディレクトリ:$PATH;export␣PATH

/usr/local/stampiディレクトリの下には図3.1に示すディレクトリが作成されている.

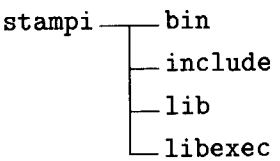


図 3.1: Stampiシステムディレクトリの構成

各ディレクトリの説明を表3.1に示す.

表 3.1: Stampiシステムディレクトリの説明

#	ディレクトリ	説明
1	stampi	Stampiシステムディレクトリ
2	stampi/bin	利用者が実行するコマンドを格納する
3	stampi/include	コンパイルに必要なインクルードファイルを格納する
4	stampi/lib	リンク時に使用するライブラリを格納する
5	stampi/libexec	システムが起動するコマンドを格納する

[2] .cshrcを変更した場合にはsource ~/.cshrc;rehashを実行するか, 再度loginし直す必要がある.

さらにStampiでは内部処理でrshコマンドを使用するため, 並列計算機間でrshが有効になるよう各自ホームディレクトリ上.rhostsの設定が不可欠である. 付録Eに CCSEでの .rhostsの設定例を掲載する. .rhostsによる設定が有効になっているかどうかは, 以下のコマンドを実行することによって確認できる (機種, OSによってはrshの代わりにremshを使用する). コマンドを実行して "Permission denied." などのエラーが発生する場合は, .rhostsを見直すこと[3].

```
% rsh␣ホスト名␣-l␣ユーザID␣ls
```

3.2 Stampiを用いたプログラム開発

3.2.1 Fortranでの開発: jmpif90/jmpif77

Stampiを用いたFortran(本書では基本的にFortran 90を想定している)プログラムには, 通常のMPIでのプログラミング同様に手続きの先頭 (IMPLICIT文, USE文を使用する場合はその直後) に以下に示すINCLUDE文を記述しなくてはならない.

```
    INCLUDE "mpif.h"
```

またコンパイル・リンクを行うために異機種共通コマンドjmpif90を用いる. ただし, Fortran90コンパイラが試用できないシステムでは, Fortran77コンパイラに対応したjmpif77を用いる. jmpif90はMPIをコンパイルするために必要だったオプションやインクルードパス, ライブラリパスの設定を全てコマンド内で吸収しているため, マシン個々で大きく異なっていたMPIに関する指定を省略できかつ同一のコマンドでMPIプログラムの開発が行えるようになった. MPI以外のコンパイルオプション指定 (最適化など) は従来通りに行うことができる. Makefileを用いる場合には, マクロの再定義 (FC=jmpif90もしくはjmpif77) を行うとよい.

```
• コンパイルの例
% jmpif90␣-c␣foo.f

• リンクの例
% jmpif90␣-o␣foo␣foo.o␣bar.o
```

jmpif77を用いる場合には, システム上のFortran77コンパイラでINCLUDE文が有効であり, かつ, 変数名の命名法の制限 (英数文字で6文字以下, アンダースコアが利用できないなど) がFortran90並みに緩和されていなくてはならない[4]. COMPACS各機種で必要なオプションを付録Dに掲載しておく.

[3] .rhostsの記述が正しいにも関わらずrshが受け付けられない場合には, まず .rhostsの実行属性が600(-rw-------)かを確認すること. それでも正しく動作しない場合には, telnetでそのホストにログインしwhoコマンドで表示されるログイン元のアドレスと .rhostsの記述が一致しているかを確認すること. しばしば経路によって使用するネットワークデバイスが違うため, ホスト名が通常のホスト名と異なって識別される場合ある.

[4] GNU g77等での動作確認は行っている.

3.2.2 C 言語での開発: jmpicc

Stampi を用いた C プログラムには, 通常の MPI でのプログラミング同様にファイルの先頭で以下のヘッダファイルを include しなくてはならない.

```
#include "mpi.h"
```

C 言語についても Fortran90 と同様に, 異機種共通コマンドjmpicc を用いてコンパイル・リンクを行う. MPI 以外のコンパイルオプション指定 (最適化など) は従来通りに行うことができる. Makefile を用いる場合には, マクロの再定義 (CC=jmpicc) を行うとよい.

- コンパイルの例

```
% jmpicc␣-c␣foo.c
```

- リンクの例

```
% jmpicc␣-o␣foo␣foo.o␣bar.o
```

尚, 現段階では C++ 言語のサポートは行っていない.

3.2.3 Java 言語 (Stampi/Java 仕様)

Stampi/Java を用いた Java アプレットを作成するには, Java プログラムの先頭部分に以下の形式でパッケージを取り込む必要がある. Stampi/Java パッケージの import 指定により, Java 言語のコンパイル時に特にオプションを指定する必要はない.

```
import sce.*;
import java.net.*;
import java.io.*;
```

Stampi/Java では, 表 3.2のクラスを提供する. クラス定義の詳細は付録 B に示す.

表 3.2: Stampi/Java 提供クラス

#	クラス	説明
1	Stampi	Stampi/Java 全体を表す
2	MPI_Comm	コミュニケータ
3	MPI_Intracomm	内部コミュニケータ
4	MPI_Intercomm	相互コミュニケータ
5	MPI_Datatype	データ型
6	MPI_Info	各種情報
7	MPI_Staus	通信情報

3.3 Stampi でのプログラム実行: jmpirun

Stampi を用いて作成した利用者プログラムの実行は, jmpirun コマンドを使用して行う. jmpirun コマンドは, /usr/local/stampi/bin ディレクトリにインストールされる. このため, コマンドを使用する際は, PATH 環境変数にこのディレクトリを加えるか, コマンドをフルパスで指定しなければならない.

jmpirun コマンドの構文は以下の通り ([] は省略可能を表す).

```
jmpirun -np ノード数 [オプション] コマンド及びパラメタ
```

ノード数は, 起動する MPI プロセスの数を表す.

オプションに指定可能な項目を表 3.3に示す.

表 3.3: jmpirun に指定可能なオプション

#	オプション	説明
1	-part *part*	SR2201 のパーティションを指定する
2	-tcpnodelay	TCP オプション TCPNODELAY を有効にする
3	-bufsize *size*	Stampi が通信に使用するバッファのサイズ (KB 単位)
4	-sockbufsize *size*	SOCKET オプション SO_SENDBUF, SO_RECVBUF に指定するサイズ (KB 単位)
5	-shortsize *size*	Stampi の内部プロトコルである short プロトコルを適用するサイズ (KB 単位)
6	-conf *file*	設定ファイル

"-part *part*" オプションは SR2201 のパーティションを表す. SR2201 以外の機種で指定した場合は無視される. SR2201 で指定を省略した場合は, DEFPART 環境変数の値が用いられる.

"-tcpnodelay", "-bufsize *size*", "-sockbufsize *size*", "-shortsize *size*" の各オプションによって, Stampi や SOCKET 通信のパラメタを変更することができる. Client/Server 型の接続を行うプログラムの実行では, "-tcpnodelay", "-bufsize *size*", "-sockbufsize *size*", "-shortsize *size*" の各オプションの指定を Client 側と Server 側とで一致させること.

"-conf *file*" オプション指定を記述した設定ファイルを指定する. 設定ファイルは, 1 行 1 オプションの形式で指定するオプションを記述する. なお, 設定ファイルには -conf オプションを指定することはできない. 設定ファイルの例を以下に示す.

```
-tcpnodelay
-sockbufsize 128
-bufsize 128
```

"-conf *file*" の指定がない場合, jmpirun は次の順番で設定ファイルを検索し, 最初に見付かったファイルを (最大でも 1 つだけ) 読み込む.

(1) カレントディレクトリの ".jmpirun" ファイル

(2) 利用者のホームディレクトリの ".jmpirun" ファイル

(3) "/usr/lcoal/stampi/libdata/jmpirun" ファイル

これら設定ファイルは, 前述の -conf オプションで指定する設定ファイルと同じ形式である. これら設定ファイルを作成することにより, jmpirun のオプションのデフォルトをシステム管理者, および, 利用者が変更することができる. オプションの優先順序は,

コマンドライン指定 → -conf オプションの設定ファイルの指定
→ 見付かった設定ファイルの指定 → システムデフォルト

となる.

コマンド及びパラメタには, 利用者プログラムと (もしあれば) パラメタを指定する.

jmpirun の使用例を以下に示す. この例では, foo コマンドをノード数 3 で起動する.

```
% jmpirun -np 3 foo
```

尚, この例は shell 環境からの起動であるが, NQS (SP2 では LoadLeveler) でバッチ実行する場合は, jmpirun コマンドをバッチキューに投入するシェルスクリプトに記述する. 特に VPP シリーズでは, shell から MPI コマンドを実行することができないため, 注意が必要である.

バッチジョブの作成方法, 実行方法については, 各ベンダの NQS マニュアル (SP2 では LoadLeveler のマニュアル) を参照のこと.

4 Stampiにおける基本的なプログラミング手法

Stampiの基本的なプログラミングは，動的プロセス生成とそれによって生成されたプロセスとの間の通信で構成される．本節ではStampiの基本的なプログラミング手法をFortranを中心に説明する．

4.1 Infoオブジェクトの取り扱い

始めに動的プロセス生成や通信路確保を行うStampi機能に対して，利用者からの詳細な情報設定をする際に利用するInfoオブジェクトの取り扱いについて解説する．Infoオブジェクトは(key,value)組の情報を格納するためのリスト構造体であり，MPI1.2仕様において採用された構造体ならびに関数仕様である．Infoオブジェクトは MPI2 の幾つかの関数で情報の指定及び引用に用いられる．

4.1.1 Infoオブジェクトの生成

- Fortran インターフェイス
  `MPI_INFO_CREATE(`*info*`,` *ierror*`)`
  `INTEGER` *info*`,` *ierror*
- C 言語インターフェイス
  `int MPI_Info_create(MPI_Info *`*info*`)`
- Stampi/Java インターフェイス
  `MPI_Info` *info* `= new MPI_Info(`*Stampi*`)`

○ 引数

出力	*info*	ハンドル	生成されたInfoオブジェクト

Infoオブジェクトはリスト構造へのハンドルであり，プログラム中に複数個定義しても構わない．

4.1.2 Infoオブジェクトへの情報の設定

- Fortran インターフェイス
  `MPI_INFO_SET(`*info*`,` *key*`,` *value*`,` *ierror*`)`
  `INTEGER` *info*`,` *ierror*
  `CHARACTER*(*)` *key*`,` *value*
- C 言語インターフェイス
  `int MPI_Info_set(MPI_Info` *info*`, char *`*key*`, char *`*value*`)`
- Stampi/Java インターフェイス
  info`.Set(string` *key*`, string` *value*`)`

○ 引数

入出力	*info*	ハンドル	Infoオブジェクト
入力	*key*	文字型	設定する情報のkey
入力	*value*	文字型	keyに対するvalue

Info オブジェクトへの情報の設定はオーバライト (上書き) であり, 同一キーの設定を複数回試みた場合には最後に設定したものが有効である.

4.1.3 Info オブジェクトの情報参照

- Fortran インターフェイス
  MPI_INFO_GET(*info*, *key*, *valuelen*, *value*, *flag*, *ierror*)
  INTEGER *info*, *valuelen*, *ierror*
  CHARACTER*(*) *key*, *value*
  LOGICAL *flag*

- C 言語インターフェイス
  int MPI_Info_get(MPI_Info *info*, char **key*, int *valuelen*, char **value*,
  int **flag*)

○ 引数

入出力	*info*	ハンドル	Info オブジェクト
入力	*key*	文字型	参照する情報の key
入力	*valuelen*	整数型	value に格納可能な文字列長
出力	*value*	文字型	key に対する value
出力	*flag*	論理型	key が定義されているかのチェック

この関数は *key* に対応する値 (*value*) を返却する関数である. もし, *key* に対応する値が Info オブジェクトに設定されていた場合, 変数 *flag* に真値 (Fortran の場合.true., C 言語の場合非 0 値) を返し *value* に対応する値を格納する. そうでなければ, *flag* に擬値 (Fortran の場合.false., C 言語の場合 0 値) を返す. このとき, 長さ *valuelen* を越える文字列部分はカットされる. C 言語の場合には, *valuelen* にヌルターミネータを含めた長さを指定しなくてはならない.

尚, 本機能に対応する Stampi/Java のクラスメソッドは提供していない.

4.1.4 Info オブジェクトの解放

- Fortran インターフェイス
  MPI_INFO_FREE(*info*, *ierror*)
  INTEGER *info*, *ierror*

- C 言語インターフェイス
  int MPI_Info_free(MPI_Info **info*)

○ 引数

入出力	*info*	ハンドル	Info オブジェクト

Stampi/Java の場合, 使われなくなったオブジェクトは JVM(Java Virtual Machine) が自動的に解放するため特に解放する必要はない. あえて挙げるならば, null を代入する操作がオブジェクト解放に相当する.

4.1.5 ファイルを介したInfoオブジェクトの設定

Infoオブジェクトの設定を，プログラム中でなくファイルを介して行うことができる．このとき，*key*に'file'を指定し，当該ファイルを値として登録することになる．この時，指定されたファイルをinfoファイルと呼び，次に示す書式でInfoオブジェクトの記述がなされる．Stampiでは，Infoファイルの書式は1行単位で

key : *value*

と決めている．なお，MPI2ではinfoファイルの書式は実装依存にしているため他のMPI2実装とは互換性がない．行頭に# が付くものはコメントと見なされる．

また，Infoオブジェクト並びにInfoファイルに指定できる*key*は任意のものが可能である．しかしながら，Infoオブジェクトを引数とするStampi関数によって有効な*key*が決まるので注意が必要である．それら有効な*key*に関しては，25,34頁に記載しているのでそれらを参考にして欲しい．

4.2 Master/Slave型動的プロセス生成と通信

動的プロセス生成とは実行中のMPIプロセスから新たに別のMPIプロセスを実行しグループ間コミュニケータを確立することをいう．このとき新たなMPIプロセスは同一並列計算機に限らず，異なる並列計算機上に生成することができる．動的プロセス生成は次に示すMPI_Comm_spawn関数を利用する．

- Fortranインターフェイス

```
MPI_COMM_SPAWN(command, argv, maxporcs, info, root, comm, intercomm,
               array_of_errcodes, ierror)
CHARACTER*(*) command, argv(*)
INTEGER maxprocs, info, root, comm, intercomm, array_of_errcodes(*), ierror
```

- C言語インターフェイス

```
int MPI_Comm_spawn(char *command, char *argv[], int maxprocs, MPI_Info info,
                   int root, MPI_Comm comm, MPI_Comm *intercomm,
                   int array_of_errcodes[]);
```

- Stampi/Javaインターフェイス

```
MPI_Intercomm.Spawn(string *command, string[] argv, int maxprocs,
                    MPI_Info info, int root)
```

○ 引数

入力	*command*	文字型	生成する外部プロセスのプログラム名
入力	*argv*	文字型配列	*command*への引数
入力	*maxprocs*	整数型	生成するプロセスの数
入力	*info*	ハンドル	(key,value)組による情報の集合によって，プロセス生成に関する詳細を指示する
入力	*root*	整数型	プロセス生成に関する前述の引数が有効なランク
入力	*comm*	ハンドル	プロセス生成を含むグループ内コミュニケータ
出力	*intercomm*	ハンドル	旧グループと生成されたグループ間のコミュニケータ
出力	*array_of_errcodes*	整数型配列	プロセス毎の戻り値

次の3つの引数項についてはMPI予約定数を指定することができる．

argv	MPI_ARGV_NULL	外部プロセスに引数なし
info	MPI_INFO_NULL	Infoオブジェクトによる詳細指定をしない
array_of_errcodes	MPI_ERRCODES_IGNORE	プロセス毎の戻り値(エラー)を返さない

MPI_Comm_spawn呼出の結果，外部プロセスの生成に成功した場合には*intercomm*に新たに生成された外部プロセスグループとのグループ間コミュニケータが格納される．外部プロセス生成の正否は，MPI_Comm_spawnの戻り値(*ierr*)と配列*array_of_errcodes*に返却される値を参照することで確認できる．MPI_Comm_spawnで返却される新しいグループ間コミュニケータ*comm*はMPIで規定されたグループ間コミュニケータとほぼ同等の機能を有する[5]．

[5]制限事項に関しては4.X章を参考にされたい．

MPI_Comm_spawn を用いた最も簡単な例として，マネージャによるワーカー制御プログラム (または Master/Slave プログラム) を示す[6].

• マネージャープログラム (Master)

```
      program manager
      include 'mpif.h'
      integer world_size, universe_size, universe_sizep, everyone
      character*100 worker_program
c
      call MPI_Init(ierr)
      call MPI_Comm_size(MPI_COMM_WORLD, world_size, ierr)
      if(world_size.ne.1) call errer("Top heavy with management")
c
      write(6,*) 'How many processes total?'
      read(5,*)   universe_size
      if(universe_size.eq.1) call error("No room to start workers")
c
      call choose_worker_program(worker_program)
      call MPI_Comm_spawn(worker_program, MPI_ARGV_NULL, universe_size-1,
     &        MPI_INFO_NULL, 0, MPI_COMM_SELF, everyone,
     &        MPI_ERRCODES_IGNORE, ierr)
c
c  並列コードをここに記述する．コミュニケータ "everyone"は生成された
c  プロセス (グループ間コミュニケータ "everyone"におけるリモートグループ内で
c  ランク 0, ... universe_size-2) 間との通信に利用される．
c
      call MPI_Finalize(ierr)
      end
```

• ワーカープログラム (Slave)

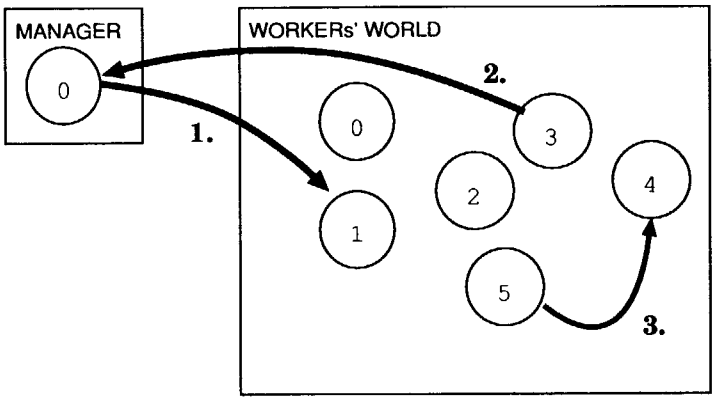
```
      program worker
      include 'mpif.h'
      integer size, parent
c
      call MPI_Init(ierr)
      call MPI_Comm_get_parent(parent, ierr)
      if(parent.eq.MPI_COMM_NULL) call error("No parent!")
      call MPI_Comm_remote_size(parent, size, ierr)
      if(size.ne.1) call error("Something's wrong with the parent")
c
c  並列コードをここに記述する．
c  マネージャーは (リモートグループの)parent のランク 0 で表される．
c  もしワーカー間で通信を行う必要があるときには，MPI_COMM_WORLD を利用する．
c
      call MPI_Finalize(ierr)
      end
```

[6]これは MPI2 仕様書 [17] の 92 頁に C 言語で書かれたソースコードをもとに主要部分のみを Fortran に翻訳したものである.

本プログラムはマネージャー, ワーカープログラムがそれぞれ別の実行オブジェクトからなる MPMD 形式であるとともに, 1 つの Master プロセスが多くの Slave プロセスをコントロールする Master/Slave 処理方式をとるものである. 単一プロセスであるマネージャー (以下 Master) が利用者からワーカー (以下 Slave) のプロセス数の入力を受け, 起動すべきプログラムを実行中に決定し (15 行目 `choose_worker_program`) 起動をかける. この例の場合, 特に詳細な指定がないので Slave は Master と同じ計算機上で起動される.

4.2.1 Master/Slave 間の一対一通信

MPI において, 通信相手の指定は '(コミュニケータ, グループ内ランク)' で行われる. 本プログラムにおいては, プログラム中のコメントにもあるように, Master 側から Slave 側への通信には, 関数 `MPI_Comm_spawn` を呼んだ際に格納されたグループ間コミュニケータ *everyone* を用いて行う. 逆に Slave 側から Master への通信を行うためには, 32 行目にある `MPI_Comm_get_parent` を呼び出し, 自分自身を spawn した Master(親) のグループとのグループ間コミュニケータ (*parent*) を取得しそれを用いて通信



- 1 の場合

```
マネージャー ::   call MPI_Send(mes,len,MPI_INTEGER,1,tag,everyone,ierr)
ワーカー [1] ::   call MPI_Recv(mes,len,MPI_INTEGER,0,tag,parent,status,
                                                                  ierr)
```

- 2 の場合

```
ワーカー [3] ::   call MPI_Send(mes,len,MPI_INTEGER,0,tag,parent,ierr)
マネージャー ::   call MPI_Recv(mes,len,MPI_INTEGER,3,tag,everyone,
                                                    status,ierr)
```

- 3 の場合

```
ワーカー [5] ::  call MPI_Send(mes,len,MPI_INTEGER,4,tag,MPI_COMM_WORLD,
                                                                  ierr)
ワーカー [4] ::  call MPI_Recv(mes,len,MPI_INTEGER,5,tag,MPI_COMM_WORLD,
                                                    status,ierr)
```

図 4.1: マネージャー / ワーカー間の一対一通信

を行わなくてはならない．Slave 同士の通信にはワーカー世界でのMPI_COMM_WORLD を用いればよい．図4.1の上部分の様な，通信を行う場合のプログラムの記述を図 4.1の下部分に示した．Master/Slave 間での一対一通信と Slave グループ内での一対一通信の記述例として参考にされたい．本例では，Master プログラムは 1 プロセスになっているが，当然，複数プロセスであっても構わない．

なお，Master 側と Slave 側のMPI_COMM_WORLD はそれぞれのグループ全体を表現するコミュニケータとして存在する．プログラムの字面上は同一のMPI_COMM_WORLD が別の実体として複数存在することに注意されたい．

4.2.2 Master/Slave とのグループ間コミュニケータの取得

- Fortran インターフェイス
  MPI_COMM_GET_PARENET(*parent*, *ierror*)
  INTEGER *parent*, *ierror*
- C 言語インターフェイス
  int MPI_Info_create(MPI_Comm **parent*)

○ 引数

出力	*parent*	ハンドル	Master グループ間のコミュニケータ

MPI_Comm_get_parent を Master プログラム内で呼び出した場合には，*parent* にはMPI_COMM_NULL が返却される．この仕様を利用すると，SPMD スタイルでの異機種の制御が可能となる．一つのプログラムに Master と Slave の両者の動作を記述しておき，MPI_Comm_get_parent の返却値をもとに動作を切り替えることができる．尚，Stampi/Java の仕様により Java アプレットは Slave プログラムになれないため，Stampi/Java には本機能に対応するメソッドは存在しない．

4.2.3 Master グループ (もしくは Slave グループ) のプロセス数取得

- Fortran インターフェイス
  MPI_COMM_REMOTE_SIZE(*parent*, *size*, *ierror*)
  INTEGER *parent*, *size*, *ierror*
- C 言語インターフェイス
  int MPI_Comm_remote_size(MPI_Comm **parent*, int **size*)
- Stampi/Java インターフェイス
  size = *comm*.Get_remote_size();

○ 引数

入力	*parent*	ハンドル	Master グループ間のコミュニケータ
出力	*size*	整数型	Master グループ間のプロセス数

Master グループのプロセス数は 34 行目にある様に MPI_Comm_remote_size を利用して得ることができる．Slave プロセス側で Master グループとのグループ間コミュニケータを引数に指定することで，Master グループのプロセス数が返却される．逆に，Master プロセス側でMPI_Comm_remote_size にグループ間コミュニケータを指定すると，Slave グループのプロセス数を得ることができる．

4.2.4 Master/Slave 間の同期と集合通信

Stampi では, MPI_Barrier 関数の呼び出しにグループ間コミュニケータを指定することによって, Master と Slave に属す全てのプロセスで同期することができる.

Master/Slave 間の同期は, Master 側のプロセスグループに属す全てのプロセスと Slave 側のプロセスグループに属す全てのプロセスがMPI_Barrier を呼び出すことによって行う. MPI_Barrier の呼び出しは, Master 側全プロセスと Slave 側全プロセスから呼び出されるまで呼び元に戻らない.

Master/Slave 間同期の記述例を以下に示す.

```
Master:   call MPI_BARRIER(everyone, ierr)
Slave:    call MPI_BARRIER(parent, ierr)
```

尚, Stampi/Java は同期をサポートしない.

また, Stampi は MPI2 で仕様が確定されたグループ間コミュニケータを用いた場合の集合通信をサポートしている. グループ間コミュニケータを用いた通信は, MPI のセマンティクス上自分から見て別のグループに属するプロセスへの通信であるため, 通常のグループ内集合通信とは実行形態が異なる. これらは, MPI-2 仕様書第 7 章にあるように大きく分けて次の 4 つに大別される (Stampi がサポートする関数名を同時に示す).

- **All-To-All** 全てのプロセスが送受信に関わり, 全プロセスが結果を保持するもの.
  MPI_Allreduce
- **All-To-One** 全てのプロセスが送信に関わり, 結果は 1 プロセスのみに保持されるもの.
  MPI_Gather, MPI_Gatherv
  MPI_Reduce
- **One-To-All** 1 プロセスのみが送信を行い, 結果は全プロセスに保持されるもの.
  MPI_Bcast
  MPI_Scatter, MPI_Scatterv
- **Other** 上記のどれにも当てはまらないもの.
  MPI_Barrier

今, グループ L,R の 2 つの間でグループ間コミュニケータが形成され, 集合通信がなされるものとして説明しよう. All-To-All タイプの集合演算は, L,R それぞれが所有するデータに対する集合演算を行った後に, 対となるグループ (L ならば R, R ならば L) に結果を送信するものである. また, All-To-One タイプの集合演算は, L が所有するデータに対する集合演算を行った後に, R の 1 つのプロセスに結果を送信する形態 (もしくは L,R 逆にしたもの) である. One-To-All タイプの集合演算は, All-To-One の送受信の立場が逆になったものといえる.

ここで, All-To-One もしくは One-To-All の送信, 受信側を指定しなくてはならないが. このタイプの集合通信には rank の指定が必要であることを利用する. 集合通信に参加する全てのプロセスは, rank に対して MPI2 で決められた適当な値を指定すれば良い. まず, One となるべきプロセスには rank にMPI_ROOT を指定し, One となるプロセスを含むグループで One とならないプロセスにはMPI_PROC_NULL を指定する. All となるべきグループでは, 送受信対象となる One の rank を指定する. この指定方法によって, All/One または送 / 受信の指定を行うことができる.

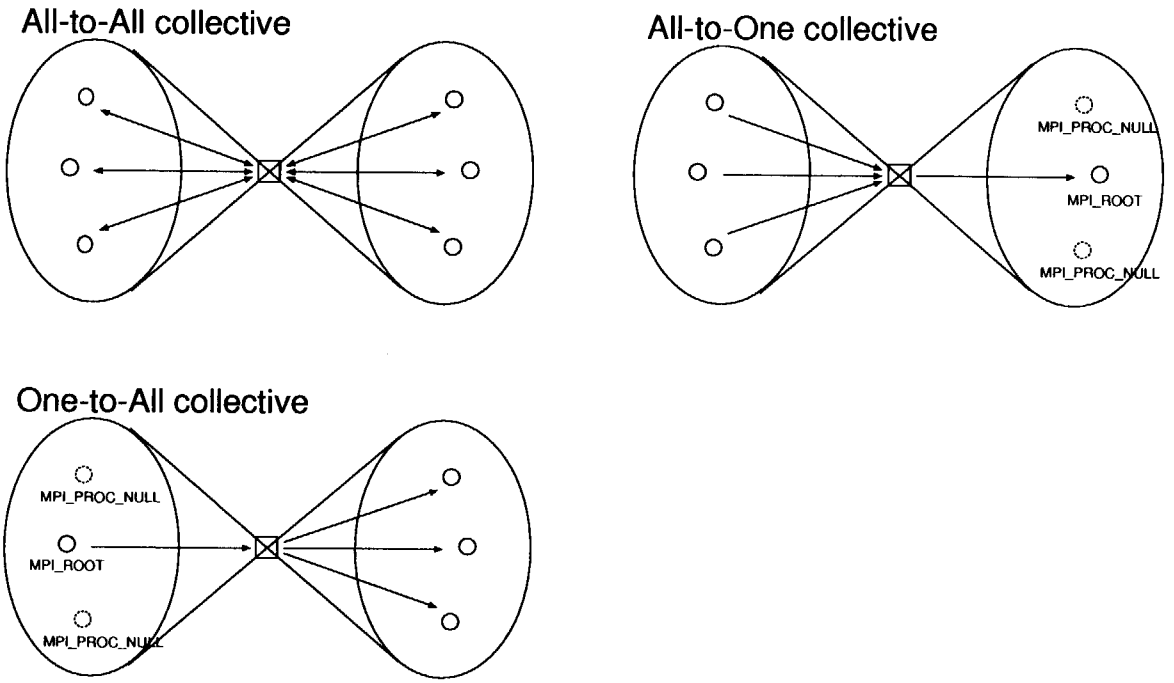


図 4.2: グループ間集合通信 (All-to-All(左上), All-to-One(右上), One-to-All(左下)) の例

次に, Master/Slave プログラムにおいて Master から Slave グループ全プロセスへのブロードキャストの例を示す.

```
Master:   call MPI_BCAST(bufer,n,MPI_INT,everyone, MPI_ROOT, ierr)
Slave:    call MPI_BCAST(bufer,n,MPI_INT,parent, 0, ierr)
```

もしここで, Master プログラムのプロセス数が 2 以上の場合は, root となるべきプロセスがMPI_ROOT の値を指定し, それ以外はMPI_PROC_NULL を指定しなくてはならない.

同様に, Master/Slave プログラムにおいて Slave グループ全プロセスのデータの総和を Master 上に格納するリダクション操作の例を示す.

```
Master:   call MPI_Reduce(bufer,n,MPI_INT,everyone, MPI_SUM, MPI_ROOT, ierr)
Slave:    call MPI_Reduce(bufer,n,MPI_INT,parent, MPI_SUM, 0, ierr)
```

もしここで, Master プログラムのプロセス数が 2 以上の場合は, root となるべきプロセスがMPI_ROOT の値を指定し, それ以外はMPI_PROC_NULL を指定しなくてはならない.

4.2.5 Info オブジェクトを使った Master/Slave 型外部プロセス生成

MPI_Comm_spawn の引数には, 外部プロセス生成のための幾つかの指定が可能であったが, Info オブジェクトを用いることで更に詳細な設定を行うことができる. Info オブジェクトを用いて次の条件でプログラムを生成する例を示す.

- hi00011 という計算機上に
- /home001/g0188/j0000/test ディレクトリ上の
- sample01 というプログラムを
- /home001/g0188/j0000/run というディレクトリにおいて
- NQS のqueX というクラス, partX というパーティションで
- プロセス数 3 個分

• Info オブジェクトを使った例

```
      call MPI_INFO_CREATE(info, ier)
      call MPI_INFO_SET(info, "host", "hi00011",ier)
      call MPI_INFO_SET(info, "path", "/home001/g0188/j0000/test", ier)
      call MPI_INFO_SET(info, "wdir", "/home001/g0188/j0000/run", ier)
      call MPI_INFO_SET(info, "nqsq", "queX", ier)
      call MPI_INFO_SET(info, "part", "partX", ier)
      call MPI_COMM_SPAWN("sample01", MPI_ARGV_NULL, 3, info,
     1         0, MPI_COMM_WORLD, icomm, MPI_ERRORCODES_IGNORE, ier)
      if(ier.ne.MPI_SUCCESS)then
            write(*,*) "Spawn failed"
            stop 1
      endif
       ......
```

この例ではプログラムに対してパラメータ群をハードコーディングしてしまうのでそれらの変更が必要になった場合には, ソースの再コンパイルをしなくてはならない. パラメータを頻繁に変更する場合には, それらを file を介してシステムに通知する仕組みが MPI2 では用意されている. Info オブジェクトのfile キーに対してファイル名を指定することでなされる.

• info ファイルを使った例

```
      call MPI_INFO_SET(info, "file", "info",ier)
      call MPI_COMM_SPAWN("", MPI_ARGV_NULL, 0, info,
     1         0, MPI_COMM_WORLD, icomm, MPI_ERRORCODES_IGNORE, ier)
```

• info ファイルの内容

```
      host: hi00011
      path: /home001/g0188/j0000/test
      -cmd: sample01
      wdir: /home001/g0188/j0000/run
      nqsq: queX
      part: partX
      -np:  3
```

また, 各情報の指定を省略した場合には, Stampi システムが規定するデフォルト時の動作を行う. 外部プロセス起動に関する同一キーへの情報指定の優先順位は次のように規定している.

> info ファイル > Info オブジェクト > MPI_Comm_spawn への引数 > デフォルト

外部プロセス起動に有効な Info オブジェクトのキー, 並びに info ファイルに指定可能な情報及ぶデフォルト動作は表 4.1に示す通りである. 第 3 カラム目に, ("F", "O") の有無に応じて, info ファイル, Info オブジェクトそれぞれに指定可能かを示している. 尚これら以外の情報が記述されている場合にはその情報は無視される. proxy キー指定の詳細は, 付録 C を参照されたい.

表 4.1: 外部プロセス起動を制御する情報

#	キー		説明	省略時の動作
1	host	FO	プロセス生成先ホスト	localhost を仮定
2	user	FO	プロセスを実行するユーザ	USER 環境変数の値を仮定
3	wdir	FO	プロセスを実行するディレクトリ	
4	path	FO	コマンドサーチパス	
5	part	FO	プロセスを実行するパーティション	DEFPART 環境変数の値を仮定
6	nqsq	FO	NQS のキュー	TSS で実行
7	node	FO	NQS で使用するノード数	引数の値を仮定
8	proxy	FO	通信制御プログラムの起動制御情報	Stampi システムが決定する
9	file	O	プロセス起動制御ファイル	
10	-cmd	F	実行するコマンド及びパラメタ	MPI_Comm_spawn の引数
11	-np	F	MPI プロセスのプロセス数	MPI_Comm_spawn の引数

第三カラムの表示 [F/O] は次の意味を持つ　F･･･info ファイルでの指定が可能
O･･･info オブジェクトでの指定が可能

4.3 Client/Server 型プログラム例

前節では，Master/Slave 型の接続を実現するプログラムの紹介を行ったが，本節では，Stampi がサポートするもう一つの接続形態 Client/Server 型接続について紹介する．

Client/Server 型接続は，個別の計算機上で既に起動されたプログラムを相互に利用するために通信路を確立し，通信を行う形態のものである．Master/Slave の時とは異なり，何らかの方法で Server,Client の両プログラムを利用者の責任のもとに起動しておかなくてはならない．

さて，前置きが長くなったが，Stampi を用いた Fortran による Client/Server 型プログラムの例を以下に示す．

• サーバプログラム

```
      program server
      include "mpif.h"
      integer client, status(MPI_STATUS_SIZE), size, again, ie, MAX_DATA
      character(MPI_MAX_PORT_NAME) port_name
      parameter(MAX_DATA = 1000)
      double precision buf(MAX_DATA)

      call MPI_INIT(ie)
      call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ie)
      if(world_size /= 1) call error("Server too big")

      call MPI_OPEN_PORT(MPI_INFO_NULL, port_name, ie)
      print *, "server available at ", port_name
      do
          call MPI_Comm_accept(port_name, MPI_INFO_NULL, 0,
     &         MPI_COMM_WORLD, client, ie)
          again = 1
          do while(again == 1)
              call MPI_RECV(buf, MAX_DATA, MPI_DOUBLE, MPI_ANY_SOURCE,
     &             MPI_ANY_TAG, client, status, ie)
              select case(status(MPI_TAG))
                case(0); call MPI_Finalize(ie)
                         stop
                case(1); call MPI_COMM_DISCONNECT(client, ie)
                         again = 0
                case default; ! 何らかの処理
              end select
          end do
      end do
      end
```

• クライアントプログラム

```
      program client
      include "mpif.h"
      integer server, done, tag, n, ie, MAX_DATA
      parameter(MAX_DATA = 1000)
      double precision buf(MAX_DATA)
      character(MPI_MAX_PORT_NAME) port_name

      call MPI_INIT(ie)
```

```
38      call getarg(2, port_name) ! コマンドライン引数にサーバポートを指定
39      call MPI_COMM_CONNECT(port_name, MPI_INFO_NULL, 0,
40     &     MPI_COMM_WORLD, server, ie)
41      done = 0
42      do while(done == 0)
43          tag = 2
44          ! 何らかの処理 (buf と n を設定)
45          call MPI_SEND(buf, n, MPI_DOUBLE, 0, tag, server, ie)
46          ! 何らかの処理 (処理終了で done = 1)
47      end do
48      call MPI_SEND(buf, 0, MPI_DOUBLE, 0, 1, server, ie)
49      call MPI_COMM_DISCONNECT(server, ie)
50      call MPI_FINALIZE(ie);
51      end
```

このプログラムでは, サーバプログラムのポート情報を標準出力に出力し, それをクライアントプログラムのコマンドライン引数に指定することによって, サーバプログラムにクライアントプログラムが接続して通信を行う.

このプログラム例をもとに, Stampi で Client/Server 型通信を行う場合のポイントを説明する.

4.3.1 接続用ポートの管理

Client/Server 型の接続は, Server 側でオープンしたポートに Client から接続することによって行われる. Server でのポートオープンは, MPI_Open_port 関数を呼び出すことによって行う.

MPI_Open_port のインタフェースを以下に示す.

- Fortran 言語インタフェース:
  MPI_OPEN_PORT(*info*, *port_name*, *ierror*)
  INTEGER *info*, *ierror*
  CHARACTER*(*) *port_name*

- C 言語インタフェース:
  MPI_Open_port(MPI_Info *info*, char **port_name*)

○ 引数

入力	*info*	ハンドル	ポートオープンに関する詳細指示
出力	*port_name*	文字型	オープンしたポートの名称

info については, MPI 予約定数である MPI_INFO_NULL を指定することができる. *info* に MPI_INFO_NULL を指定した場合, オープンするポートの詳細指定を行わないことを表す.

MPI_Open_port 関数は, オープンしたポートの名称を *port_name* に返却する. Stampi ではポートの名称を次のフォーマットで表す.

hostname:*port*

hostname は Server となる計算機のホスト名で *port* は接続を受け付ける TCP ポート番号である.

MPI_Open_port の *info* によってオープンするポートのIPアドレス (ip_address キーで指定) とTCP ポート番号 (ip_port キーで指定) を指定することができる.

TCP ポート番号を 2508 番に指定する例を以下に示す.

```
      integer info, ierr
      character(MPI_MAX_PORT_NAME) portname
      call MPI_INFO_CREATE(info, ier)
      call MPI_INFO_SET(info, "ip_port", "2508", ier)
      call MPI_OPEN_PORT(info, portname, ierr)
      if(ierr /= MPI_SUCCESS)then
         print *, "MPI_Open_port failed"
         call MPI_FINALIZE(ierr)
         stop
      endif
```

info 指定で TCP ポート番号を指定することによって, オープンされるポートの名称を予め推定することができるため, Client は予め決められた Server のポートに接続するように処理することができる. 逆に, TCP ポート番号を指定しない場合はシステムが適当な TCP ポート番号を割り当てるため, Server から何らかの手段によって Client にポートの名称を引き渡す必要がある.

一方, IP アドレスの指定は, その指定がない場合には計算機が持つ全てのネットワークインタフェースで接続を受け付けるため, 一般には指定する必要はない. 複数のネットワークインタフェースを持つ計算機で, 特定のネットワークインタフェースからの接続のみ許可する目的で, そのネットワークインタフェースに割り当てられている IP アドレスを指定する. 但し, IP アドレス指定は接続に使用するネットワークインタフェースを特定するが, 接続に用いたネットワークインタフェースを接続後の通信でも使用することを保証するものではない点に注意すること.

info で指定した TCP ポートが他のプログラムで使用中などの理由により使用不可能な場合, MPI_Open_port はエラーとなる.

MPI_Open_port でオープンしたポートは, MPI_Close_port を呼び出してクローズすることができる.

MPI_Close_port のインタフェースを以下に示す.

- Fortran 言語インタフェース:
  MPI_CLOSE_PORT(*port_name*, *ierror*)
  INTEGER *ierror*
  CHARACTER*(*) *port_name*

- C 言語インタフェース:
  MPI_Close_port(char **port_name*)

○ 引数

入力	*port_name*	文字型	クローズするポートの名称

MPI_Close_port に指定する *port_name* は, MPI_Open_port でオープンしたポートでなければならない. MPI_Open_port でオープンしたポートは, MPI_Close_port によってクローズするか, MPI_Finalize を行うまでの間有効である. Client プログラムは, Server プログラムのポートが有効な期間なら何時でも

(Server での MPI_Comm_accept 呼び出しとは無関係に) MPI_Comm_connect を呼び出して Client/Server 間の接続を要求することができる.

MPI_Open_port 及び MPI_Close_port は, 集団通信関数ではない点に注意すること. 即ち, MPI_Comm_accept の root プロセスでだけMPI_Open_port と MPI_Close_port を呼び出すようにすること.

4.3.2 Client/Server 間の接続と切断

Server 側の MPI_Open_port でオープンしたポートに対して, Client から MPI_Comm_connect を呼び出して接続を要求することができる. Server 側では, Client の MPI_Comm_connect 呼び出しによる接続要求を, MPI_Comm_accept を呼び出すことによって受け付ける.

MPI_Comm_accept のインタフェースを以下に示す.

- Fortran 言語インタフェース:
  MPI_COMM_ACCEPT(*port_name*, *info*, *root*, *comm*, *newcomm*, *ierror*)
  INTEGER *info*, *root*, *comm*, *newcomm*, *ierror*
  CHARACTER*(*) *port_name*

- C 言語インタフェース:
  MPI_Comm_accept(char **port_name*, MPI_Info *info* , int *root*, MPI_Comm *comm*, MPI_Comm **newcomm*)

○ 引数

入力	*port_name*	文字型	接続を受け付けるポートの名称
入力	*info*	ハンドル	接続受け付けに関する詳細指示
入力	*root*	整数型	ルートプロセスのランク
入力	*comm*	ハンドル	Server プロセスグループのコミュニケータ
出力	*newcomm*	ハンドル	Client との間のグループ間コミュニケータ

注) *port_name*, *info* はルートプロセスでのみ意味を持つ

info については, MPI 予約定数である MPI_INFO_NULL を指定することができる. *info* に MPI_INFO_NULL を指定した場合, 接続受け付けの詳細指定を行わないことを表す.

次に, MPI_Comm_connect のインタフェースを以下に示す.

- Fortran 言語インタフェース:
  MPI_COMM_CONNECT(*port_name*, *info*, *root*, *comm*, *newcomm*, *ierror*)
  INTEGER *info*, *root*, *comm*, *newcomm*, *ierror*
  CHARACTER*(*) *port_name*

- C 言語インタフェース:
  MPI_Comm_connect(char **port_name*, MPI_Info *info* , int *root*, MPI_Comm *comm*,
  MPI_Comm **newcomm*)

○ 引数

入力	*port_name*	文字型	接続するポートの名称
入力	*info*	ハンドル	接続に関する詳細指示
入力	*root*	整数型	ルートプロセスのランク
入力	*comm*	ハンドル	Client プロセスグループのコミュニケータ
出力	*newcomm*	ハンドル	Server との間のグループ間コミュニケータ

注) *port_name*, *info* はルートプロセスでのみ意味を持つ

info については, MPI 予約定数である MPI_INFO_NULL を指定することができる. *info* に MPI_INFO_NULL を指定した場合, 接続の詳細指定を行わないことを表す.

MPI_Comm_accept や MPI_Comm_connect を行うと, グループ間コミュニケータ *newcomm* が返却される. このグループ間コミュニケータを使用することによって, Client/Server 間の通信を行うことができる.

MPI_Comm_accept と MPI_Comm_connect の使用例を以下に示す. この例で, Server ポートは “hi000 11:2508” とする.

```
1 Server:
2       call MPI_INFO_CREATE(info, ierr)
3       call MPI_INFO_SET(info, "ip_port", "2508", ierr)
4       call MPI_OPEN_PORT(info, port_name, ierr)
5       call MPI_COMM_ACCEPT(port_name, MPI_INFO_NULL, 0, MPI_COMM_WORLD,
6      &                     client, ierr)
7 Client:
8       call MPI_COMM_CONNECT("hi00011:2508", MPI_INFO_NULL, 0, MPI_COMM_WORLD,
9      &                      server, ierr)
```

MPI_Comm_accept または MPI_Comm_connect で行った Client/Server 間の接続は, MPI_Comm_disconnect を呼び出すことによって切断することができる. Client/Server 間の接続は, MPI_Comm_accept または MPI_Comm_connect を行ってから, MPI_Comm_disconnect を呼び出すして接続を切断するか, MPI_Finalize を呼び出して MPI を終了するまで有効である.

MPI_Comm_disconnect のインタフェースを以下に示す.

- Fortran 言語インタフェース:
  MPI_Comm_disconnect(*comm*, *ierror*)
  INTEGER *comm*, *ierror*

- C 言語インタフェース:
  MPI_Close_port(MPI_Comm **comm*)

○ 引数

入出力	*comm*	ハンドル	切断の対象となるグループ間コミュニケータ

4.3.3 リモートグループのプロセス数の取得

Client プログラムから Server プログラムのプロセス数が, また, 逆に Server プログラムから Client プログラムのプロセス数がそれぞれ必要になる場合がある.

相手グループのプロセス数は, MPI_Comm_remote_size によって問い合わせることができる. MPI_Comm_remote_size のインタフェースは, 22ページ参照. Clienr/Server いずれも通信相手のプロセス数を問い合わせる場合には, MPI_Comm_accept を呼び出した際に格納されるグループ間コミュニケータを MPI_Comm_remote_size に指定する. リモートグループのプロセス数取得例を次に示す.

```
1  Server:
2        integer    client_size     ! Client プロセス数
3
4         .....
5        call MPI_COMM_REMOTE_SIZE(client, client_size, ierr)
6  Client:
7        integer   server_size     ! Server プロセス数
8
9         .....
10       call MPI_COMM_REMOTE_SIZE(server, server_size, ierr)
```

4.3.4 Client/Server 間での一対一通信

Server から Client への通信は, MPI_Comm_accept を呼び出した際に格納されるグループ間コミュニケータ (例では client) を用いて行う. 逆に Client から Server への通信は, MPI_Comm_connect を呼び出した際に格納されるグループ間コミュニケータ (例では server) を用いて行う. Server 同士, 或は, Client 同士の通信は, 各々の MPI_COMM_WORLD を用いる (Server と Client でプログラムの字面上は同一の MPI_COMM_WORLD が別々の実体として複数存在することに注意).

Client/Server 間の通信の記述例を以下に示す.

```
(1) Server → Client (ランク 1) の通信
    Server:      call MPI_SEND(msg,len,type,1,tag client,ierr)
    Client(1):   call MPI_RECV(msg,len,type,0,tag,server,status,ierr)

(2) Client (ランク 3) → Server の通信
    Client(3):   call MPI_SEND(msg,len,type,0,tag,server,ierr)
    Server:      call MPI_RECV(msg,len,type,3,tag,client,status,ierr)

(3) Client 間 (ランク 5 → ランク 4) の通信
    Client(5):   call MPI_SEND(msg,len,type,4,tag,MPI_COMM_WORLD,ierr)
    Client(4):   call MPI_RECV(msg,len,type,5,tag,MPI_COMM_WORLD,status,
                &                ierr)
```

4.3.5 Client/Server 間での同期と集合通信

Stampi では, MPI_Barrier 関数の呼び出しにグループ間コミュニケータを指定すすことによって, Server と Client に属す全てのプロセスで同期することができる.

Client/Server 間の同期は, Server 側のプロセスグループに属す全てのプロセスと, Client 側のプロセスグループに属す全てのプロセスが MPI_Barrier を呼び出すことによって行う. MPI_Barrier の呼び出しは, Server 側全プロセスと Client 側全プロセスから呼び出されるまで呼び元に戻らない.

Client/Server 間同期の記述例を以下に示す.

```
Server:   call MPI_BARRIER(client, ierr)
Client:   call MPI_BARRIER(server, ierr)
```

Master/Slave プログラムでの示したと同様に, Client/Server 間でもグループ間コミュニケータを指定することによってグループ間での集合通信が可能である. Client/Server 間での Server のランク 0 プロセスから Client グループ全プロセスへのブロードキャストの例を示す.

```
Server:   call MPI_BCAST(bufer,n,MPI_INT,everyone, MPI_ROOT, ierr)
Client:   call MPI_BCAST(bufer,n,MPI_INT,parent, 0, ierr)
```

もしここで, Master プログラムのプロセス数が 2 以上の場合は, root となるべきプロセスがMPI_ROOT の値を指定し, それ以外はMPI_PROC_NULL を指定しなくてはならない.

同様に, Client/Server プログラムにおいて Client グループ全プロセスのデータの総和を Server のランク 0 プロセス上に格納するリダクション操作の例を示す.

```
Server:   call MPI_Reduce(bufer,n,MPI_INT,everyone, MPI_SUM,
         &                                          MPI_ROOT, ierr)
Client:   call MPI_Reduce(bufer,n,MPI_INT,parent, MPI_SUM, 0, ierr)
```

もしここで, Masterプログラムのプロセス数が2以上の場合は, rootとなるべきプロセスがMPI_ROOTの値を指定し, それ以外はMPI_PROC_NULLを指定しなくてはならない.

4.3.6 Client/Server型プログラムにおけるInfoオブジェクト設定

(1) ポート生成 (MPI_Open_port) の制御

ポート生成を制御する情報として, 設定ファイルに記述できる情報を表4.2に示す.

表 4.2: ポート作成制御ファイルに記述できるデータ

#	キー		説明	省略時の動作
1	ip_port	FO	作成するポートのTCPポート番号	システムが決定
2	ip_address	FO	作成するポートのIPアドレス	任意のアドレスで接続を待ち合わせる
3	file	O	Infoファイルの指定	Infoファイルを使用しない

第三カラムの表示 [F/O] は次の意味を持つ　F･･･infoファイルでの指定が可能
O･･･infoオブジェクトでの指定が可能

(2) 接続受け付け (MPI_Comm_accept) の制御

接続受け付けを制御する情報として, 設定ファイルに記述できる情報を, 表4.3に示す.

表 4.3: 接続受け付け制御ファイルに記述できるデータ

#	キー		説明	省略時の動作
1	proxy	FO	通信制御プログラムの起動制御情報	システムが決定
2	file	O	Infoファイルの指定	Infoファイルを使用しない
3	-port	F	接続を受け付けるポートの名称	MPI_Comm_accept の引数を使用

第三カラムの表示 [F/O] は次の意味を持つ　F･･･infoファイルでの指定が可能
O･･･infoオブジェクトでの指定が可能

(3) 接続 (MPI_Comm_connect) の制御

接続を制御する情報として, 設定ファイルに記述できる情報を, 表4.4に示す.

表 4.4: 接続制御ファイルに記述できるデータ

#	キー		説明	省略時の動作
1	proxy	FO	通信制御プログラムの起動制御情報	システムが決定
2	-port	F	接続するポートの名称	MPI_Comm_connect の引数を使用

第三カラムの表示 [F/O] は次の意味を持つ　F･･･infoファイルでの指定が可能
O･･･infoオブジェクトでの指定が可能

表で示した項目以外のキーが指定されている場合, そのキーの情報は無視される (エラーにはならない). 同一キーへの情報指定を行った場合の, 優先順位は26ページに示したものと同様である. proxyキー指定の詳細は, 付録Cを参照されたい.

4.4 グループ間コミュニケータで接続されたグループのマージ

Stampi では, Master/Slave もしくは Client/Server のいずれかの方法で MPI が通信可能な世界 (以下 WORLD と呼ぶ) を広げなくてはならない. また, いずれの方法においてもグループ間コミュニケータを用いたままでの通信は対等でない複数のグループの存在を強く意識しなくてはならず不便である. MPI には, グループ間コミュニケータで接続されたグループを束ねて新しいグループを作り出す機能が備わっている. グループ間コミュニケータから新しいグループ (WORLD に対して UNIVERSE と呼ぶ) を生成するためには`MPI_Inter comm_merge` を用いる. `MPI_Intercomm_merge` のインターフェースを以下に示す.

- Fortran インターフェイス
  `MPI_INTERCOMM_MERGE(`*intercomm*`,` *high*`,` *newcomm*`,` *ierror*`)`
  `INTEGER` *intercomm*, *newcomm*, *ierror*
  `LOGICAL` *high*

- C 言語インターフェイス
  `int MPI_Intercomm_merge(MPI_Comm` *intercomm*`, int` *high*`, MPI_Comm *`*newcomm*`)`

○ 引数

入力	*intercomm*	ハンドル	グループ間コミュニケータ
入力	*high*	論理型	マージ後のグループ順序指定
出力	*newcomm*	ハンドル	マージされたグループのグループ内コミュニケータ

`MPI_Intercomm_merge` は集合通信であるため, 指定するグループに属する全てのプロセスが関数を呼び出さなくてはならない. 引数 *high* には, マージする際のグループの順序付けを指示する (`.false.` をつけたグループを`.true.` としたグループの前におくときめる. 両者が同じ値の場合にはシステムが任意に決定する). WORLD のマージ結果 (UNIVERSE) は, *newcomm* に格納されグループ内コミュニケータとして利用することができる.

4.5 制限事項

Stampi では, MPI 関数について以下の制限を設けている. Stampi 利用の際は注意すること.

(1) 異機種間通信を行う場合,

MPI_Send, MPI_Recv, MPI_Isend, MPI_Irecv,
MPI_Bcast, MPI_Gather, MPI_Gatherv, MPI_Scatter, MPI_Scatterv,
MPI_Reduce, MPI_Allreduce

の各関数は, データ型として MPI の基本データ型と大域リダクション (MPI_Reduce) で用いる組み込み型を指定できる. 異機種間通信で使用可能なデータ型を表 4.5に示す.

表 4.5: 異機種間通信で使用できるデータ型

#	言語	MPI の型	言語の型
1	C 言語	MPI_CHAR	signed char
2		MPI_SHORT	signed short
3		MPI_INT	signed int
4		MPI_LONG	signed long
5		MPI_UNSIGNED_CHAR	unsigned char
6		MPI_UNSIGNED_SHORT	unsigned short
7		MPI_UNSIGNED_INT	unsigned
8		MPI_UNSIGNED_LONG	unsigned long
9		MPI_FLOAT	float
10		MPI_DOUBLE	double
11		MPI_BYTE	任意のデータ型
12		MPI_SHORT_INT	short と int
13		MPI_2INT	int の対
14		MPI_LONG_INT	long と int
15		MPI_FLOAT_INT	float と int
16		MPI_DOUBLE_INT	double と int
17	Fortran 言語	MPI_INTEGER	INTEGER
18		MPI_REAL	REAL
19		MPI_DOUBLE_PRECISION	DOUBLE PRECISION
20		MPI_LOGICAL	LOGICAL
21		MPI_COMPLEX	COMPLEX
22		MPI_CHARACTER	CHARACTER(1)
23		MPI_BYTE	任意のデータ型
24		MPI_2INTEGER	INTEGER 型の対
25		MPI_2REAL	REAL 型の対
26		MPI_2DOUBLE_PRECISION	DOUBLE PRECISION 型の対

(2) MPI_Reduce, MPI_Allreduce に指定可能な演算は
MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD,
MPI_LAND, MPI_LOR, MPI_LXOR,
MPI_BAND, MPI_BOR, MPI_BXOR,

MPI_MAXLOC, MPI_MINLOC
の 12 種類である.

(3) MPI_Intercomm_merge でマージされたグループに対して, 更に他のグループをマージすることはできない.

(4) MPI_Intercomm_merge でマージされたグループから, MPI_Comm_spawn やMPI_Comm_connect 等を用いて外部プロセスと通信するためのグループ間コミュニケータを生成することはできない.

(5) サポートされていない MPI 関数は, 同一機種内の通信であっても使用できない.

(6) プロファイリングインタフェース (PMPI_ プレフィックスの関数) は使用できない.

(7) Stampi/Java では, Java アプレットは Master/Slave 型接続における Master としかなれない.

(8) Stampi/Java では, Java での MPI プロセスは常に 1 でなくてはならない. また, Java 内部での通信はエラーとなる.

5　おわりに

異機種並列計算機間通信ライブラリ Stampi は, 従来のベンダー提供の MPI ライブラリでは実現されていなかった異機種間の通信を MPI インターフェイスのみで可能とする世界的にも初めての MPI2 実装系である. 本報告書はユーザズガイド第二版として, 第一版に引き続き, Stampi の利用法, 異機種共通コマンドならびに関数仕様等について説明した.

すでに Stampi を使用した異機種並列プログラムの成果が報告されており, 今後更なる計算機の利用法が展開されつつあるといえよう.

現在の開発計画では, COMPACS 以外の原研所有の各並列計算機を含めて対応機種の拡張を図る予定であり, 各種ワークステーション (SMP も含む), PC クラスタ機等での利用が可能になるであろう. また, 異機種分散計算を効率的に実現するための各種新機能 (分散並列 IO など) の実現を試みていく予定である. 結果は追って報告される予定であるとともに, 将来発行される利用手引き第 3 版以降に反映されることとなる.

最後に本報告書を通じて多くの研究者に Stampi を利用していただければ幸いである.

謝辞

本報告書を取りまとめるにあたり, 貴重な機会を与えてくださいました計算科学技術推進センター長秋元正幸氏, 並列処理基本システム開発グループリーダー平山俊雄氏に深謝致します. また本ライブラリ開発にあたり, 貴重なご意見を頂きました早稲田大学理工学部 笠原 博徳教授に深謝致します.

参考文献

[1] 徳田, 小出, 今村, 松本 ほか： トカマク・プラズマにおけるハイブリッド・シミュレーション, 日本物理学会 53 回年会, 1pP5 (1998).

[2] Kimura T. and Takemiya H. : Local Area Metacomputing for Multidiciplinary Problems: A Case Study for Fluid/Structure Coupled Similation, *Proc. of Int. Conf. on Supercomputing*, pp.149–156 (1998).

[3] Imamura T. and Tokuda S. : A Hybrid Computing by Coupling Different Architectural Machines, a Case Study for Tokamak Plasma Simulation, *Proc. of IASTED Int. Conf. on Parallel and Distributed Computing and Systems*, (1999).

[4] Nieplocha J. et al. : Global Arrays: A Nonuniform Memory Access Programming Model for High Performance Computers, *The J. of Supercomputing*, No.10, pp.169–189, (1996).

[5] Nieplocha J. et al. : ChemIO: High Performance Parallel I/O for Computational Chemistry Applications, *The Intl. J. of High Performance Computing Applications*, Vol. 12, No. 3, pp.345–363 (1998).

[6] 村松 他 ： 並列計算機での流体解析のための実時間可視化システムの開発, JAERI-Data/Code 98-014 (1998)

[7] 北端 他 : 実時間可視化システムを備えた WSPEEDI 放射能放出源推定システム, 計算工学講演会論文集 Vol.4, No.1, pp.317–320 (1999).

[8] Kitabata H. and Chino M. : Development of Source Term Estimation Method during Nuclear Emergency, *Proc. of Intl. Conf. Mathematical Methods and Computations in Reactor Physics, M&C99*, (1999).

[9] Grimshaw A. et al. : Metasystems, *Comm. ACM*, No. 41, Vol. 11, pp.46–62 (1998).

[10] Foster I. and Kesselman C. : The Globus Project: A status Report, *Proc. IPPS/SPDP'98 Heterogeneous Computing Workshop*, pp.4–18 (1998).

[11] Foster I. and Kesselman C. edit : *The GRID, Blueprint for a New Computing Insrastracture* , Morgan Kaufmann Pub. Inc., (1999).

[12] Intel Corporation : *Paragon System Fortran Calls Reference Manual* (1995).

[13] 日本IBM : *Parallel Environment for AIX, MPL Programming and Subroutine Reference*, Version 2 Release 1, IBM manual GS23-3893-00 (1995).

[14] Butler R. and Lusk E. : Monitors, messages and clusters: The p4 parallel programming system. Tech. Rep. MCS-P362-0493, Argonne National Laboratory (1993).

[15] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R. and Sunderam V. : *PVM: A User's Guide and Tutorial for Networked Parallel Computing*, The MIT Press (1994). `http://www.netlib.org/pvm3/book/pvm-book.html`

[16] Message Passing Interface Forum : *MPI: Message passing interface standard* (1996). http://www.mpi-forum.org/
同日本語訳は http://www.ppc.nec.co.jp/mpi-j/mpi-report-j.ps.Z より入手可能.

[17] Message Passing Interface Forum : *MPI-2: Extensions to the message-passing interface* (1997). http://www.mpi-forum.org/

[18] 小出, 今村, 太田, 川崎 ほか : 並列分散科学技術計算環境 STA(3) — 異機種並列計算機間ライブラリの構築, 計算工学会講演論文集, Vol. 3, No. 1, pp. 81–84 (1998).

[19] 小出, 今村, 武宮, 平山 : 異機種並列計算機間の通信を支援する並列分散通信ライブラリ: Stampi, 日本流体力学会年会'98 講演論文集, pp.413–414 (1998).

[20] 今村, 小出, 武宮 : 異機種並列計算機間通信ライブラリ:Stampi－利用手引書, JAERI Data/Code 98-034 (1998).

[21] Gropp W. and Lusk E. : *User's Guide for mpich, a Portable Implementation of MPI* (1998). http://www-c.mcs.anl.gov/mpi/mpich/

[22] Foster I. et al. : Wide-Area Implementation of the Message Passing Interface, *Parallel Computing*, Vol. 24, No. 12, pp.1725–1749 (1998).

[23] Burns G. D. et al., LAM: An Open Cluster Environment for MPI, *Supercomputing Symposium '94*, 1994

[24] Gabriel E. et al. : Distributed computing in a heterogeneous computing environment, *Proc. of EuroPVMMPI'98*, (1998).

[25] IMPI Steering Committee : IMPI – Interoperable Message-Passing Interface, DRAFT, protcol version 0.0, (1999).

[26] サン・マイクロシステムズ, http://java.sun.com/

[27] Gropp W., Lusk E. and Skjellum A.: *Using MPI, Portable Parallel Programming with the Message-Passing Interface*, The MIT Press (1996).

[28] 日立製作所: *HI-UX/MPP MPI·PVM*使用の手引き, 日立ソフトウェアマニュアル 6A20-3-026-10 (1998).

[29] 富士通株式会社: *UXP/V MPI*使用手引書 *V11*用, 富士通マニュアル J2U5-0271-01 (1998).

[30] 日本IBM: *Parallel Environment for AIX: MPI Programming and Subroutine Reference*, Ver. 2, Rel. 1, IBM manual GC23-3894-00 (1995).

[31] CRAY Research Inc.: *Message Passing Toolkit: Release Overview*, CRAY manual RO-5290 1.1 (1996).

[32] CRAY Research Inc.: *Message Passing Toolkit: MPI Programmer's Manual*, CRAY manual SR-2197 1.1 (1996).

[33] 日本電機株式会社: *SUPER-UX MPI/SX*利用の手引き, 日本電気マニュアル G1AF09-1 (1996).

付録 A　Stampi ライブラリインターフェイス

Stampi がサポートする MPI 関数を表 A.1にアルファベット順に示す．一番右のカラムは外部通信への対応範囲を表している．○印が付いたものは外部通信に対応している (内部通信にも対応している)．△印は現在外部通信はサポート対象外であるが今後拡張の対象となっているものである．×印のものは内部通信のみサポートする．ここで外部通信に対応するとは，**外部に生成したグループとのグループ間コミュニケータならびにそれから派生する通信のリクエストや派生データ型，Info オブジェクトなどを通信関数の引数に指定できることをいう．**

また，本表にないものであっても MPI の関数群はグループ内通信の範囲内で利用可能のものもあるが，著者らがその使用に関して動作を保証するものではない．MPI または MPI2 の仕様ならびに使用方法については参考文献 [16, 17, 27], 各社利用マニュアル [28, 29, 30, 32, 33, 21] を参考にされたい．

表 A.1: Stampi 関数一覧

#	関数	説明	外部通信対応
1	MPI_Abort	コミュニケータ内のプロセスの実行を中断する．	○
2	MPI_Allreduce	コミュニケータ内の前プロセスで大域的なリダクション操作 (総和, 最大値, 最小値, 論理演算など) を行い，結果を全プロセスに返却する．	○
3	MPI_Barrier	コミュニケータ内の全プロセスでバリア同期を行う．	○
4	MPI_Bcast	コミュニケータ内の 1 つのプロセスから他の全プロセスに対してデータを送信 (または受信) する．	○
5	MPI_Cart_coords	コミュニケータ内のプロセスのランクを座標に変換する．	×
6	MPI_Cart_create	コミュニケータにカルテシアン (直積構造のトポロジ) 情報を付加した新しいコミュニケータを返却する．	×
7	MPI_Cart_shift	カルテシアン構造のトポロジにおいて，座標方向へのデータシフト動作を支援する．具体的には MPI_Sendrecv 関数に指定するランク (通信相手のコミュニケータ内のランク) を求める．	×
8	MPI_Close_port	MPI_Open_port で開いた Server ポートをクローズする．	○
9	MPI_Comm_accept	MPI_Open_port で開いた Server ポートに Client からの接続を受け付ける．	○
10	MPI_Comm_connect	Client/Server 型の通信で，Client から Server のポートに接続する．	○
11	MPI_Comm_disconnect	MPI_Comm_accept または MPI_Comm_connet で確立した接続を切断する．	○
12	MPI_Comm_dup	コミュニケータの複製を返却する．	×
13	MPI_Comm_free	コミュニケータを解放する．	×
14	MPI_Comm_get_parent	親プロセスのコミュニケータを返却する．親プロセスがない場合 MPI_COMM_NULL を返却する．	○
15	MPI_Comm_rank	コミュニケータ内での自プロセスのランクを返却する．	○

16	MPI_Comm_remote_size	グループ間コミュニケータでの他グループ内のプロセス数を返却する.	○
17	MPI_Comm_size	コミュニケータ内のプロセス数を返却する.	○
18	MPI_Comm_spawn	MPI プロセスを生成する.	○
19	MPI_Comm_split	コミュニケータ内のプロセスをサブグループに分割することによって, 新しいコミュニケータを定義する.	△
20	MPI_Finalize	MPI 終了関数.	○
21	MPI_Gather	コミュニケータ内の全プロセスから 1 つのプロセスにデータを収集し, ランクの順にデータを格納する.	○
22	MPI_Gatherv	コミュニケータ内の全プロセスから 1 つのプロセスに可変データを収集し, ランクの順にデータを格納する.	○
23	MPI_Get_count	MPI_Recv や MPI_Irecv で受信したデータ数を返却する.	○
24	MPI_Info_create	Info オブジェクトを生成する.	○
25	MPI_Info_delete	Info オブジェクトから情報を削除する.	○
26	MPI_Info_dup	Info オブジェクトの複製を作成する.	○
27	MPI_Info_free	Info オブジェクトを解放する.	○
28	MPI_Info_get	Info オブジェクトから key に対応する情報を返却する.	○
29	MPI_Info_get_nkeys	Info オブジェクトに登録されている情報の個数を返却する.	○
30	MPI_Info_get_nthkey	Info オブジェクトに登録された n 番目の情報を返却する.	○
31	MPI_Info_get_valuelen	Info オブジェクトに登録されてた key に対応する情報の大きさを返却する.	○
32	MPI_Info_set	Info オブジェクトに key に対応する情報を登録する.	○
33	MPI_Init	MPI 初期化関数.	○
34	MPI_Intercomm_merge	グループ間コミュニケータの両端のプロセスグループを合成する.	○
35	MPI_Iprobe	ノンブロッキングでの受信確認をする.	○
36	MPI_Irecv	ノンブロッキング通信によって, 他プロセスからデータを受信する.	○
37	MPI_Isend	ノンブロッキング通信によって, 他プロセスにデータを送信する.	○
38	MPI_Lookup_name	公開されている Server ポートを検索する.	○
39	MPI_Open_port	Client/Server 型通信の Server ポートをオープンする.	○
40	MPI_Probe	ブロッキングでの受信確認をする.	○
41	MPI_Publish_name	MPI_Open_port でオープンした Server ポートを公開する.	○
42	MPI_Recv	ブロッキング通信によって, 他プロセスからデータを受信する.	○
43	MPI_Reduce	コミュニケータ内の前プロセスで大域的なリダクション操作 (総和, 最大値, 最小値, 論理演算など) を行い, 結果を特定のプロセスに返却する.	○

44	MPI_Scatter	特定のプロセスにランク順に格納されている同一サイズのデータをコミュニケータ内の全プロセスに配送する.	○
45	MPI_Scatterv	特定のプロセスに格納されているデータをコミュニケータ内の全プロセスに配送する. データサイズと格納場所をランク毎に指定可能.	○
46	MPI_Send	ブロッキング通信によって, 他プロセスにデータを送信する.	○
47	MPI_Sendrecv	ブロッキング通信によるデータの送受信を行う.	×
48	MPI_Type_commit	派生データ型の記憶操作を行う. 通信バッファの記述 (通信バッファの内容ではない) を記憶する.	×
49	MPI_Type_extent	派生データ型の大きさ (バイト数) を求める.	×
50	MPI_Type_hvector	等間隔に並んだデータ型のコピーから, 新しい派生データ型を生成する. データの間隔はバイト数で指定する.	×
51	MPI_Type_vector	等間隔に並んだデータ型のコピーから, 新しい派生データ型を生成する. データの間隔はデータの数で指定する.	×
52	MPI_Wait	ノンブロッキング通信の終了待ちをおこなう.	○
53	MPI_Unpublish_name	MPI_Publish_name で公開した Server ポートの公開を解除する.	○
54	MPI_Wtick	MPI_Wtime の精度を返却する.	○
55	MPI_Wtime	ある時刻からの経過時間を返却する. ある時刻は利用者プログラムの開始から変更されない. 返却する時刻は MPI_Wtime を呼び出したノードにローカルである.	○

付録 B　Stampi/Java クラス定義一覧

Stampi/Java のクラス定義を以下に示す (public 部分のみ).

(1) Stampi クラス

```
public class Stampi extends Object
{
    public MPI_Comm         COMM_NULL;  // MPI_COMM_NULL
    public MPI_Intracomm    COMM_WORLD; // MPI_COMM_WORLD
    public MPI_Intracomm    COMM_SELF;  // MPI_COMM_SELF
    public MPI_Datatype     CHAR;       // MPI_CHAR
    public MPI_Datatype     SHORT;      // MPI_SHORT
    public MPI_Datatype     INT;        // MPI_INT
    public MPI_Datatype     FLOAT;      // MPI_FLOAT
    public MPI_Datatype     DOUBLE;     // MPI_DOUBLE
    public MPI_Datatype     BYTE;       // MPI_BYTE
    public MPI_Info         INFO_NULL;  // MPI_INFO_NULL
    public string[]         ARGV_NULL;  // MPI_ARGV_NULL
    public int              ANY_SOURCE; // MPI_ANY_SOURCE
    public int              ANY_TAG;    // MPI_ANY_TAG
    public Stampi(SceWindow)            // Stampi クラスコンストラクタ
    public void Init()                  // MPI_Init
            throws IOException,
                   UnknownHostException,
                   SceException,
                   OutOfBufferException
    public void Finalize()              // MPI_Finalize
            throws IOException
}
```

(2) MPI_Comm クラス

```
public class MPI_Comm
{
    public int Get_size()  // MPI_Comm_size, 返却値: size
    public int Get_rank()  // MPI_Comm_rank, 返却値: rank
}
```

(3) MPI_Intracomm クラス

```
public class MPI_Intracomm extends MPI_Comm
{
    public MPI_Intercomm Spawn(  // MPI_Comm_spawn, 返却値: MPI_Intercomm
        String   command,        //      コマンド
```

```
            String[] args,          //     コマンド引数
            int      maxprocs,      //     起動する MPI プロセス数
            MPI_Info info,          //     MPI_Info
            int      root)          //     root プロセス
               throws IOException
    }
```

(4) MPI_Intercomm クラス

```
    public class MPI_Intercomm extends MPI_Comm
    {
        public void Send(           // MPI_Send (scaler char 用)
            char buf,               //     送信バッファ
            int dest,               //     受信 rank
            int tag)                //     送信 tag
                throws IOException
        public void Send(           // MPI_Send (scaler short 用)
            short buf,              //     送信バッファ
            int dest,               //     受信 rank
            int tag)                //     送信 tag
                throws IOException
        public void Send(           // MPI_Send (scaler int 用)
            int buf,                //     送信バッファ
            int dest,               //     受信 rank
            int tag)                //     送信 tag
                throws IOException
        public void Send(           // MPI_Send (scaler float 用)
            float buf,              //     送信バッファ
            int dest,               //     受信 rank
            int tag)                //     送信 tag
                throws IOException
        public void Send(           // MPI_Send (scaler double 用)
            double buf,             //     送信バッファ
            int dest,               //     受信 rank
            int tag)                //     送信 tag
                throws IOException
        public void Send(           // MPI_Send (scaler byte 用)
            byte buf,               //     送信バッファ
            int dest,               //     受信 rank
            int tag)                //     送信 tag
                throws IOException
        public void Send(           // MPI_Send (配列 char 用)
            char[] buf,             //     送信バッファ
            int count,              //     送信要素数
            int dest,               //     受信 rank
```

```
    int tag)                    //      送信 tag
        throws IOException
public void Send(               // MPI_Send (配列 short 用)
    short[] buf,                //      送信バッファ
    int count,                  //      送信要素数
    int dest,                   //      受信 rank
    int tag)                    //      送信 tag
        throws IOException
public void Send(               // MPI_Send (配列 int 用)
    int[] buf,                  //      送信バッファ
    int count,                  //      送信要素数
    int dest,                   //      受信 rank
    int tag)                    //      送信 tag
        throws IOException
public void Send(               // MPI_Send (配列 float 用)
    float[] buf,                //      送信バッファ
    int count,                  //      送信要素数
    int dest,                   //      受信 rank
    int tag)                    //      送信 tag
        throws IOException
public void Send(               // MPI_Send (配列 double 用)
    double[] buf,               //      送信バッファ
    int count,                  //      送信要素数
    int dest,                   //      受信 rank
    int tag)                    //      送信 tag
        throws IOException
public void Send(               // MPI_Send (配列 byte 用)
    byte[] buf,                 //      送信バッファ
    int count,                  //      送信要素数
    int dest,                   //      受信 rank
    int tag)                    //      送信 tag
        throws IOException
public void Send(               // MPI_Send (string 用)
    string buf,                 //      送信バッファ
    int dest,                   //      受信 rank
    int tag)                    //      送信 tag
        throws IOException
public char Recv_char(          // MPI_recv (scaler char 用)
    int source,                 //      送信 rank
    int tag,                    //      送信 tag
    MPI_Status status)          //      受信状態
        throws IOException,
               OutOfBufferException
public short Recv_short(        // MPI_recv (scaler short 用)
```

```
    int source,              //     送信 rank
    int tag,                 //     送信 tag
    MPI_Status status)       //     受信状態
        throws IOException,
               OutOfBufferException
public int Recv_int(         // MPI_recv (scaler int 用)
    int source,              //     送信 rank
    int tag,                 //     送信 tag
    MPI_Status status)       //     受信状態
        throws IOException,
               OutOfBufferException
public float Recv_float(     // MPI_recv (scaler float 用)
    int source,              //     送信 rank
    int tag,                 //     送信 tag
    MPI_Status status)       //     受信状態
        throws IOException,
               OutOfBufferException
public double Recv_double(   // MPI_recv (scaler double 用)
    int source,              //     送信 rank
    int tag,                 //     送信 tag
    MPI_Status status)       //     受信状態
        throws IOException,
               OutOfBufferException
public byte Recv_byte(       // MPI_recv (scaler byte 用)
    int source,              //     送信 rank
    int tag,                 //     送信 tag
    MPI_Status status)       //     受信状態
        throws IOException,
               OutOfBufferException
public void Recv(            // MPI_recv (配列 char 用)
    char[] buf,              //     受信バッファ
    int count,               //     受信要素数
    int source,              //     送信 rank
    int tag,                 //     送信 tag
    MPI_Status status)       //     受信状態
        throws IOException,
               OutOfBufferException
public void Recv(            // MPI_recv (配列 short 用)
    short[] buf,             //     受信バッファ
    int count,               //     受信要素数
    int source,              //     送信 rank
    int tag,                 //     送信 tag
    MPI_Status status)       //     受信状態
        throws IOException,
```

```
                OutOfBufferException
    public void Recv(           // MPI_recv (配列 int 用)
        int[] buf,              //     受信バッファ
        int count,              //     受信要素数
        int source,             //     送信 rank
        int tag,                //     送信 tag
        MPI_Status status)      //     受信状態
            throws IOException,
                OutOfBufferException
    public void Recv(           // MPI_recv (配列 float 用)
        float[] buf,            //     受信バッファ
        int count,              //     受信要素数
        int source,             //     送信 rank
        int tag,                //     送信 tag
        MPI_Status status)      //     受信状態
            throws IOException,
                OutOfBufferException
    public void Recv(           // MPI_recv (配列 double 用)
        double[] buf,           //     受信バッファ
        int count,              //     受信要素数
        int source,             //     送信 rank
        int tag,                //     送信 tag
        MPI_Status status)      //     受信状態
            throws IOException,
                OutOfBufferException
    public void Recv(           // MPI_recv (配列 byte 用)
        byte[] buf,             //     受信バッファ
        int count,              //     受信要素数
        int source,             //     送信 rank
        int tag,                //     送信 tag
        MPI_Status status)      //     受信状態
            throws IOException,
                OutOfBufferException
    public string Recv_string(  // MPI_recv (string 用)
        int source,             //     送信 rank
        int tag,                //     送信 tag
        MPI_Status status)      //     受信状態
            throws IOException,
                OutOfBufferException
    public int Get_remote_size() // MPI_Remote_size, 返却値: remote size
}
```

(5) MPI_Datatype クラス

```
public class MPI_Datatype
```

```
{
}
```

(6) MPI_Info クラス

```
public class MPI_Info
{
    public MPI_Info(Stampi)      // MPI_Info クラスコンストラクタ
            throws IOException
    public void Set(             // MPI_Info_set
        string key,              //     key
        string val)              //     value
            throws IOException
}
```

(7) MPI_Status クラス

```
public class MPI_Status
{
    public MPI_Status()          // MPI_Status クラスコンストラクタ
    public int Get_count(        // MPI_Get_count, 返却値: 受信要素数
        MPI_Datatype)            //     受信データ型
    public int Get_source()      // MPI_SOURCE
    public int Get_tag()         // MPI_TAG
    public int Get_error()       // MPI_ERROR
}
```

付録 C　proxy キーによる通信制御プログラムの起動制御

MPI_Info または、設定ファイルに proxy キーを指定することによって、通信制御プログラムの起動を制御することができる。

STAMPI では、Master/Slave 型接続時、或は、Client/Server 型接続時、2 組の MPI プロセスグループ間の通信を中継する通信制御プログラムを使用する。通信制御プログラムの起動形態には、以下の 3 種類がある。

(1) 通信制御プログラムを使用しないでプロセスグループ間で直接通信する (通信制御プログラムは起動しない)

(2) Master/Slave (または Client/Server) のどちらか一方の計算機に通信制御プログラムを起動し、プロセスグループ間の通信を通信制御プログラム経由で行う

(3) Master/Slave (または Client/Server) の両方の計算機に通信制御プログラムを起動し、プロセスグループ間の通信を 2 つの通信制御プログラムを経由して行う

システムは、MPI プロセスを実行する計算機の性質やプロセスグループ内のプロセス数に応じて、通信制御プログラム起動方式を決定する。

proxy キーの指定によって、システムが自動的に行っている通信制御プログラム起動方式を変更することができる。proxy キーに指定できる値を表 C.1に示す。

表 C.1: proxy キーに指定できる値

#	値	説明
1	none	通信制御プログラムを起動しない
2	remote	通信制御プログラムを相手計算機側にのみ起動する
3	local	通信制御プログラムを自計算機側にのみ起動する
4	master	通信制御プログラムを MASTER 側計算機にのみ起動する
5	slave	通信制御プログラムを SLAVE 側計算機にのみ起動する
6	server	通信制御プログラムを SERVER 側計算機にのみ起動する
7	client	通信制御プログラムを CLIENT 側計算機にのみ起動する
8	both	通信制御プログラムを両方の計算機に起動する

STAMPI を実行する計算機によっては、通信制御プログラムを必ず必要とするものがある。通信制御プログラムを必ず必要とする計算機で STAMPI を利用する場合、システムは proxy キーの指定とは無関係に通信制御プログラムの起動を行い、計算機間で通信できるようにする。このように、proxy キーの指定が必ずしも有効にならない場合がある。

proxy キーに表 C.1に示した値以外の値を指定した場合、または、Master/Slave 型接続で server や client を指定した場合、Client/Server 型接続で master や slave を指定した場合には、proxy キーの指定を無視し、システムが通信制御プログラムの起動を決定する。

付録 D　COMPACS 等各機種でのコンパイルリンク・オプション

日本原子力研究所計算科学技術推進センター (日立 SR2201, 富士通 VPP300, IBM SP2 (IBM MPI 環境), CRAY T94, NEC SX4, 及び SGI Onyx), 東海研究所 (富士通 AP3000 及び 富士通 VP2400), 那珂研究所 (Intel Paragon, IBM SP2 (MPICH 環境) 及び DEC Alpha) の各マシン環境について, C 言語及び Fortran 言語でのコンパイルリンク手順を説明する.

尚, 本章では, Stampi が標準の /usr/local/stampi ディレクトリにインストールされていることを前提としている. このため, 標準以外のディレクトリにインストールされている環境では, ディレクトリ指定をインストール先に置き換えて使用すること.

D.1　日立 SR2201 でのコンパイル・リンク手順

コンパイルリンクは, C コンパイルコマンド (cc), Fortran コンパイルコマンド (f90) に以下のオプションを指定することによって行う.

- コンパイルオプション (C, Fortran 共通)

```
-I/usr/local/stampi/include
```

- C 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpi -L/usr/local/mpi/lib/hmpp2/cml -lpmpi -lmpi
```

- Fortran 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpif -ljmpi
-L/usr/local/mpi/lib/hmpp2/cml -lpmpi -lfmpi -lmpi
```

以下にコンパイルの例を示す. 以下の例では, foo.f をコンパイルし, foo.o を作成する.

```
% f90 -c -I/usr/local/stampi/include foo.f
```

次に Fortran でのリンクの例を示す. この例では, foo.o と bar.o をリンクし, foo コマンドを作成する.

```
% f90 -o foo foo.o bar.o -L/usr/local/stampi/lib -ljmpif -ljmpi
      -L/usr/local/mpi/lib/hmpp2/cml -lpmpi -lfmpi -lmpi
```

D.2 富士通 VPP300 でのコンパイル・リンク手順

コンパイルリンクは, C コンパイルコマンド (cc), Fortran コンパイルコマンド (frt) に以下のオプションを指定することによって行う.

- C 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- Fortran 言語コンパイルオプション

```
-I/usr/local/stampi/include -X9
```

- C 言語リンクオプション

```
-Wl,-P,-J,-dy -L/usr/local/stampi/lib -ljmpi -L/usr/lang/mpi/lib -lpmpi
-lmpi -lmp -lelf -lpx -lsocket
```

- Fortran 言語リンクオプション

```
-Wl,-P,-J,-dy -L/usr/local/stampi/lib -ljmpif -ljmpi -L/usr/lang/mpi/lib
-lpmpi -lfmpi -lmpi -lmp -lelf -lpx -lsocket
```

以下に Fortran でのコンパイルの例を示す. 以下の例では, foo.f をコンパイルし, foo.o を作成する.

```
% frt -c -I/usr/local/stampi/include -X9 foo.f
```

次に Fortran でのリンクの例を示す. この例では, foo.o と bar.o をリンクし, foo コマンドを作成する.

```
% frt -o foo foo.o bar.o -Wl,-P,-J,-dy -L/usr/local/stampi/lib -ljmpif
      -ljmpi -L/usr/lang/mpi/lib -lpmpi -lfmpi -lmpi -lmp -lelf -lpx -lsocket
```

D.3 IBM SP2 (IBM MPI 環境) でのコンパイル・リンク手順

コンパイルリンクは, C コンパイルコマンド (mpcc), Fortran コンパイルコマンド (mpxlf) に以下のオプションを指定することによって行う.

- C 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- Fortran 言語コンパイルオプション

```
-I/usr/local/stampi/include -qintsize=4
```

- C 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpi -L/usr/lpp/ppe.poe/lib -lmpi
```

- Fortran 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpif -ljmpi -L/usr/lpp/ppe.poe/lib
-lmpi
```

以下に Fortran でのコンパイルの例を示す. 以下の例では, foo.f をコンパイルし, foo.o を作成する.

```
% mpxlf -c -I/usr/local/stampi/include -qintsize=4 foo.f
```

次に Fortran でのリンクの例を示す. この例では, foo.o と bar.o をリンクし, foo コマンドを作成する.

```
% mpxlf -qintsize=4 -o foo foo.o bar.o -L/usr/local/stampi/lib -ljmpif
      -ljmpi -L/usr/lpp/ppe.poe/lib -lmpi
```

D.4 CRAY T94 でのコンパイル・リンク手順

CTAY T94 の場合, コンパイル前に mpt (Message Passing Toolkit) を使用する準備として以下のコマンドを実行する. 本操作は, 一回のログインについて一回だけ行えばよい. (後述の jmpicc, jmpif90 コマンドを使用する場合は不要)

```
module load mpt
```

コンパイルリンクは, C コンパイルコマンド (cc), Fortran コンパイルコマンド (f90) に以下のオプションを指定することによって行う.

- C 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- Fortran 言語コンパイルオプション

```
-I/usr/local/stampi/include -a taskcommon
```

- C 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpi -L/opt/ctl/mpt/mpt/lib -lpmpi -lmpi
```

- Fortran 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpif -ljmpi -L/opt/ctl/mpt/mpt/lib
-lpmpi -lmpi
```

以下に Fortran でのコンパイルの例を示す. 以下の例では, foo.f をコンパイルし, foo.o を作成する.

```
% f90 -c -I/usr/local/stampi/include -a taskcommon foo.f
```

次に Fortran でのリンクの例を示す. この例では, foo.o と bar.o をリンクし, foo コマンドを作成する.

```
% f90 -a taskcommon -o foo foo.o bar.o -L/usr/local/stampi/lib -ljmpif
      -ljmpi -L/opt/ctl/mpt/mpt/lib -lpmpi -lmpi
```

D.5 NEC SX4 でのコンパイル・リンク手順

コンパイルリンクは, C コンパイルコマンド (cc), Fortran コンパイルコマンド (f90) に以下のオプションを指定することによって行う.

- C 言語コンパイルオプション

```
-I/usr/local/stampi/include -h float0
```

- Fortran 言語コンパイルオプション

```
-I/usr/local/stampi/include -P multi -float0
```

- C 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpi -lmpi -lpthread
```

- Fortran 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpif -ljmpi -lmpi -lpthread
```

以下に Fortran でのコンパイルの例を示す. 以下の例では, foo.f をコンパイルし, foo.o を作成する.

```
% f90 -c -I/usr/local/stampi/include -P multi -float0  foo.f
```

次に Fortran でのリンクの例を示す. この例では, foo.o と bar.o をリンクし, foo コマンドを作成する.

```
% f90 -P multi -float0 -o foo foo.o bar.o -L/usr/local/stampi/lib -ljmpif
      -ljmpi -lmpi -lpthread
```

D.6 SGI Onyx でのコンパイル・リンク手順

コンパイルリンクは, C コンパイルコマンド (cc), Fortran コンパイルコマンド (f90) に以下のオプションを指定することによって行う.

- C 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- Fortran 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- C 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpi -L/usr/local/mpi/lib/IRIX64/ch_p4 -lpmpi -lmpi
```

- Fortran 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpif -ljmpi -L/usr/local/mpi/lib/IRIX64/ch_p4
-lpmpi -lfmpi -lmpi
```

以下に Fortran でのコンパイルの例を示す. 以下の例では, foo.f をコンパイルし, foo.o を作成する.

```
% f90 -c -I/usr/local/stampi/include foo.f
```

次に Fortran でのリンクの例を示す. この例では, foo.o と bar.o をリンクし, foo コマンドを作成する.

```
% f90 -o foo foo.o bar.o -L/usr/local/stampi/lib -ljmpif
      -ljmpi -L/usr/local/mpi/lib/IRIX64/ch_p4 -lpmpi -lfmpi -lmpi
```

D.7 富士通 AP3000 でのコンパイル・リンク手順

コンパイルリンクは, C コンパイルコマンド (fcc), Fortran コンパイルコマンド (frt) に以下のオプションを指定することによって行う.

- C 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- Fortran 言語コンパイルオプション

```
-X9 -I/usr/local/stampi/include
```

- C 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpi -L/opt/FSUNmpiap -lpmpi -lmpi
-L/opt/FSUNaprun/lib -lmpl -lemi -lthread -lm -lsocket -lnsl
```

- Fortran 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpif -ljmpi -L/opt/FSUNmpiap -lpmpi -lfmpi
-lmpi -L/opt/FSUNaprun/lib -lmpl -lemi -lthread -lm -lsocket -lnsl
```

以下に Fortran でのコンパイルの例を示す. 以下の例では, foo.f をコンパイルし, foo.o を作成する.

```
% frt -c -X9 -I/usr/local/stampi/include foo.f
```

次に Fortran でのリンクの例を示す. この例では, foo.o と bar.o をリンクし, foo コマンドを作成する.

```
% frt -o foo foo.o bar.o -L/usr/local/stampi/lib -ljmpif -ljmpi
      -L/opt/FSUNmpiap -lpmpi -lfmpi -lmpi -L/opt/FSUNaprun/lib -lmpl
      -lemi -lthread -lm -lsocket -lnsl
```

D.8 富士通 VPP2400 でのコンパイル・リンク手順

コンパイルリンクは, C コンパイルコマンド (cc), Fortran コンパイルコマンド (frt) に以下のオプションを指定することによって行う.

尚, 富士通 VPP2400 の Fortran コンパイラは Fortran77 規格対応であり, Fortran90 規格には対応していない.

- C 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- Fortran 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- C 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpi -L/usr/local/mpi/lib/UXPM/ch_p4 -lpmpi -lmpi
-lsocket -lnsl -lfj7e -lfj7ef -lfj7ev -lfj7evf
```

- Fortran 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpif -ljmpi -L/usr/local/mpi/lib/UXPM/ch_p4
-lpmpi -lfmpi -lmpi -lsocket -lnsl
```

以下に Fortran でのコンパイルの例を示す. 以下の例では, foo.f をコンパイルし, foo.o を作成する.

```
% frt -c -I/usr/local/stampi/include foo.f
```

次に Fortran でのリンクの例を示す. この例では, foo.o と bar.o をリンクし, foo コマンドを作成する.

```
% frt -o foo foo.o bar.o -L/usr/local/stampi/lib -ljmpif
      -ljmpi -L/usr/local/mpi/lib/UXPM/ch_p4 -lpmpi -lfmpi -lmpi
      -lsocket -lnsl
```

D.9 Intel Paragon でのコンパイル・リンク手順

コンパイルリンクは, C コンパイルコマンド (cc), Fortran コンパイルコマンド (f77) に以下のオプションを指定することによって行う[7].

尚, Intel Paragon の Fortran コンパイラは Fortran77 規格対応であり, Fortran90 規格には対応していない. また, Intel Paragon で Fortran を使用する場合, 通常の mpif.h をインクルードする代わりに stampif.h をインクルードしなければならない.

- C 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- Fortran 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- C 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpi -lpmpi -lmpi -nx
```

- Fortran 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpif -ljmpi -lpmpi -lfmpi -mpi -nx
```

以下に Fortran でのコンパイルの例を示す. 以下の例では, foo.f をコンパイルし, foo.o を作成する.

```
% f77 -c -I/usr/local/stampi/include foo.f
```

次に Fortran でのリンクの例を示す. この例では, foo.o と bar.o をリンクし, foo コマンドを作成する.

```
% f77 -o foo foo.o bar.o -L/usr/local/stampi/lib -ljmpif
      -ljmpi -lpmpi -lfmpi -lmpi -nx
```

[7]フロントエンドプロセッサでクロスコンパイルする場合は, icc コマンド (C クロスコンパイラ) と if77 (Fortran クロスコンパイラ) を用いる

D.10 IBM SP2 (MPICH 環境) でのコンパイル・リンク手順

コンパイルリンクは, C コンパイルコマンド (cc), Fortran コンパイルコマンド (xlf90) に以下のオプションを指定することによって行う.

- C 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- Fortran 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- C 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpi -L/usr/local/mpi/lib/rs6000/ch_p4 -lpmpi -lmpi
```

- Fortran 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpif -ljmpi -L/usr/local/mpi/lib/rs6000/ch_p4
-lpmpi -lfmpi -lmpi
```

以下に Fortran でのコンパイルの例を示す. 以下の例では, foo.f をコンパイルし, foo.o を作成する.

```
% xlf90 -c -I/usr/local/stampi/include foo.f
```

次に Fortran でのリンクの例を示す. この例では, foo.o と bar.o をリンクし, foo コマンドを作成する.

```
% xlf90 -o foo foo.o bar.o -L/usr/local/stampi/lib -ljmpif
      -ljmpi -L/usr/local/mpi/lib/rs6000/ch_p4 -lpmpi -lfmpi -lmpi
```

D.11 DEC Alpha でのコンパイル・リンク手順

コンパイルリンクは, C コンパイルコマンド (cc), Fortran コンパイルコマンド (f90) に以下のオプションを指定することによって行う.

DEC Alpha では, Fortran コンパイルコマンドとして f95 コマンド (Fortran95 規格対応) を使用してもよい.

- C 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- Fortran 言語コンパイルオプション

```
-I/usr/local/stampi/include
```

- C 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpi -L/usr/local/mpi/lib/alpha/ch_p4 -lpmpi -lmpi
```

- Fortran 言語リンクオプション

```
-L/usr/local/stampi/lib -ljmpif -ljmpi -L/usr/local/mpi/lib/alpha/ch_p4
-lpmpi -lfmpi -lmpi
```

以下に Fortran でのコンパイルの例を示す. 以下の例では, foo.f をコンパイルし, foo.o を作成する.

```
% f90 -c -I/usr/local/stampi/include foo.f
```

次に Fortran でのリンクの例を示す. この例では, foo.o と bar.o をリンクし, foo コマンドを作成する.

```
% f90 -o foo foo.o bar.o -L/usr/local/stampi/lib -ljmpif
      -ljmpi -L/usr/local/mpi/lib/alpha/ch_p4 -lpmpi -lfmpi -lmpi
```

付録 E　COMPACS 等各機種での.rhosts の設定例

本章では, 日本原子力研究所計算科学技術推進センター, 東海研究所, 那珂研究所の計算機環境 (表 E.1) を例に設定方法を説明する.

表 E.1: 日本原子力研究所計算機環境

#	ホスト名	機種名	所在
1	hi00011	日立 SR2201	計算科学技術推進センター
2	fu00011	富士通 VPP300	計算科学技術推進センター
3	ibmsp50	IBM SP2	計算科学技術推進センター
4	cr00011	CRAY T94	計算科学技術推進センター
5	ne00011	NEC SX4	計算科学技術推進センター
6	ccsemge	SGI Onyx	計算科学技術推進センター
7	desr01	デスクサイド並列計算機	計算科学技術推進センター
8	stasrv1	SUN Sparc (WWW サーバ)	計算科学技術推進センター
9	apsvr03	富士通 AP3000	東海研究所
10	vppsys01	富士通 VP2400	東海研究所
11	parags, paragn	Intel Paragon	那珂研究所
12	nanasvr	IBM SP2	那珂研究所
13	hepta, mars, saturn	DEC Alpha	那珂研究所

那珂研究所の parags と paragn はフロントエンドプロセッサ (paragf) 経由で使用する. このため, 設定作業は paragf に対して行う. また, 那珂研究所の DEC Alpha (hepta, mars, saturn) は 3 台でクラスタを構成し, htpta でジョブの起動を開始した場合にもっとも効率よく働くように設定されている. このため, 他ホストの設定は hepta に対して行うとともに, 3 台でクラスタを構成するための設定も行う.

E.1　hi00011 (日立 SR2201) の設定

各自のユーザ ID でログインし, エディタで .rhosts ファイルを編集する. 内容は, 以下の通り.

```
hi00011 ユーザ ID
fu00011 ユーザ ID
ne00011 ユーザ ID
cr00011 ユーザ ID
ibmsp50a ユーザ ID
ccsemge ユーザ ID
desr01 ユーザ ID
apsvr03.tokai.jaeri.go.jp ユーザ ID
vppsys01.tokai.jaeri.go.jp ユーザ ID
paragf.naka.jaeri.go.jp ユーザ ID
nanasvr.naka.jaeri.go.jp ユーザ ID
hepta.naka.jaeri.go.jp ユーザ ID
stasrv1.koma.jaeri.go.jp ユーザ ID
localhost ユーザ ID
```

localhost を含め, 使用しないホストのエントリは省略可能.
尚, ユーザ ID は, 各自のユーザ ID に置き換えること.

E.2 fu00011 (富士通 VPP300) の設定

各自のユーザ ID でログインし, エディタで .rhosts ファイルを編集する. VPP300 では, 各ノードプロセッサの設定も必要である. 内容は, 以下の通り.

```
fu00011 ユーザ ID
fu00112 ユーザ ID
fu00212 ユーザ ID
fu00312 ユーザ ID
fu00412 ユーザ ID
fu00512 ユーザ ID
fu00612 ユーザ ID
fu00712 ユーザ ID
fu00812 ユーザ ID
fu00912 ユーザ ID
fu01012 ユーザ ID
fu01112 ユーザ ID
fu01212 ユーザ ID
fu01312 ユーザ ID
fu01412 ユーザ ID
fu01512 ユーザ ID
hi00011 ユーザ ID
ne00011 ユーザ ID
cr00011 ユーザ ID
ibmsp50a ユーザ ID
ccsemge ユーザ ID
desr01 ユーザ ID
# apsvr03.tokai.jaeri.go.jp
133.53.164.3 ユーザ ID
# vppsys01.tokai.jaeri.go.jp
133.53.5.2 ユーザ ID
# paragf.naka.jaeri.go.jp
157.111.148.27 ユーザ ID
# nanasvr.naka.jaeri.go.jp
157.111.148.20 ユーザ ID
# hepta.naka.jaeri.go.jp
157.111.159.21 ユーザ ID
stasrv1.koma.jaeri.go.jp ユーザ ID
localhost ユーザ ID
```

VPP300 の各プロセッサの指定 (fu00011 ～ fu01512) の指定は省略不可. localhost を含め, 使用しないホストのエントリは省略可能.

尚, ユーザ ID は, 各自のユーザ ID に置き換えること.

E.3 ibmsp50 (IBM SP2) の設定

各自のユーザ ID でログインし, エディタで .rhosts ファイルを編集する. 内容は, 以下の通り. "#" で開始する行はコメントである.

```
hi00011 ユーザ ID
fu00011 ユーザ ID
ne00011 ユーザ ID
cr00011 ユーザ ID
ibmsp50a ユーザ ID
ccsemge ユーザ ID
desr01 ユーザ ID
apsvr03.tokai.jaeri.go.jp ユーザ ID
# vppsys01.tokai.jaeri.go.jp ユーザ ID
paragf.naka.jaeri.go.jp ユーザ ID
nanasvr.naka.jaeri.go.jp ユーザ ID
hepta.naka.jaeri.go.jp ユーザ ID
stasrv1.koma.jaeri.go.jp ユーザ ID
localhost ユーザ ID
```

vppsys01 から ibmsp50 に対して MPI_Comm_spawn による動的プロセス生成を行うためには, vppsys01 の設定が必要である (ibmsp50 から vppsys01 へのプロセス生成は可能). localhost を含め, 使用しないホストのエントリは省略可能.

尚, ユーザ ID は, 各自のユーザ ID に置き換えること.

E.4 cr00011 (CRAY T94) の設定

各自のユーザ ID でログインし, エディタで .rhosts ファイルを編集する. 内容は, 以下の通り. "#" で開始する行はコメントである.

```
hi00011.koma.jaeri.go.jp ユーザ ID
fu00011.koma.jaeri.go.jp ユーザ ID
ne00011.koma.jaeri.go.jp ユーザ ID
cr00011.koma.jaeri.go.jp ユーザ ID
ibmsp50a.koma.jaeri.go.jp ユーザ ID
ccsemge.koma.jaeri.go.jp ユーザ ID
desr01.koma.jaeri.go.jp ユーザ ID
apsvr03.tokai.jaeri.go.jp ユーザ ID
vppsys01.tokai.jaeri.go.jp ユーザ ID
paragf.naka.jaeri.go.jp ユーザ ID
nanasvr.naka.jaeri.go.jp ユーザ ID
hepta.naka.jaeri.go.jp ユーザ ID
stasrv1.koma.jaeri.go.jp ユーザ ID
localhost.koma.jaeri.go.jp ユーザ ID
```

localhost を含め, 使用しないホストのエントリは省略可能.

尚, ユーザ ID は, 各自のユーザ ID に置き換えること.

E.5 ne00011 (NEC SX4) の設定

各自のユーザ ID でログインし, エディタで .rhosts ファイルを編集する. 内容は, 以下の通り.

```
hi00011 ユーザ ID
fu00011 ユーザ ID
ne00011 ユーザ ID
cr00011 ユーザ ID
ibmsp50a ユーザ ID
ccsemge ユーザ ID
desr01 ユーザ ID
apsvr03.tokai.jaeri.go.jp ユーザ ID
vppsys01.tokai.jaeri.go.jp ユーザ ID
paragf.naka.jaeri.go.jp ユーザ ID
nanasvr.naka.jaeri.go.jp ユーザ ID
hepta.naka.jaeri.go.jp ユーザ ID
stasrv1.koma.jaeri.go.jp ユーザ ID
localhost ユーザ ID
```

localhost を含め, 使用しないホストのエントリは省略可能.
尚, ユーザ ID は, 各自のユーザ ID に置き換えること.

E.6 ccsemge (SGI Onyx) の設定

各自のユーザ ID でログインし, エディタで .rhosts ファイルを編集する.
内容は, 以下の通り. "#" で開始する行はコメントである.

```
hi00011 ユーザ ID
fu00013 ユーザ ID
ne00011 ユーザ ID
cr00011 ユーザ ID
ibmsp50a ユーザ ID
ccsemge ユーザ ID
desr01 ユーザ ID
apsvr03.tokai.jaeri.go.jp ユーザ ID
vppsys01.tokai.jaeri.go.jp ユーザ ID
paragf.naka.jaeri.go.jp ユーザ ID
nanasvr.naka.jaeri.go.jp ユーザ ID
hepta.naka.jaeri.go.jp ユーザ ID
stasrv1.koma.jaeri.go.jp ユーザ ID
localhost ユーザ ID
```

vppsys01 から ccsemge に対して MPI_Comm_spawn による動的プロセス生成する場合, ホスト名として ccsemga を用いる.
localhost を含め, 使用しないホストのエントリは省略可能.
尚, ユーザ ID は, 各自のユーザ ID に置き換えること.

E.7 desr01 (デスクサイド並列計算機) の設定

各自のユーザ ID でログインし, エディタで .rhosts ファイルを編集する. 内容は, 以下の通り. "#" で開始する行はコメントである.

```
hi00011 ユーザ ID
fu00013.koma.jaeri.go.jp ユーザ ID
ne00011 ユーザ ID
cr00011 ユーザ ID
ibmsp50a ユーザ ID
ccsemge ユーザ ID
desr01 ユーザ ID
apsvr03.tokai.jaeri.go.jp ユーザ ID
# vppsys01.tokai.jaeri.go.jp
paragf.naka.jaeri.go.jp ユーザ ID
nanasvr.naka.jaeri.go.jp ユーザ ID
hepta.naka.jaeri.go.jp ユーザ ID
stasrv1.koma.jaeri.go.jp ユーザ ID
localhost ユーザ ID
```

vppsys01 から desr01 に対して MPI_Comm_spawn による動的プロセス生成を行うためには, vppsys01 の設定が必要である (desr01 から vppsys01 へのプロセス生成は可能). localhost を含め, 使用しないホストのエントリは省略可能.

尚, ユーザ ID は, 各自のユーザ ID に置き換えること.

E.8 apsvr03 (富士通 AP3000) の設定

各自のユーザ ID でログインし, エディタで .rhosts ファイルを編集する. 内容は, 以下の通り. "#" で開始する行はコメントである.

```
hi00011.koma.jaeri.go.jp ユーザ ID
fu00011.koma.jaeri.go.jp ユーザ ID
ne00011.koma.jaeri.go.jp ユーザ ID
cr00011.koma.jaeri.go.jp ユーザ ID
# ibmsp50a.koma.jaeri.go.jp
133.53.188.9 ユーザ ID
# ccsemge.koma.jaeri.go.jp
133.53.184.10 ユーザ ID
desr01.koma.jaeri.go.jp ユーザ ID
apsvr03.tokai.jaeri.go.jp ユーザ ID
vppsys01.tokai.jaeri.go.jp ユーザ ID
paragf.naka.jaeri.go.jp ユーザ ID
nanasvr.naka.jaeri.go.jp ユーザ ID
hepta.naka.jaeri.go.jp ユーザ ID
stasrv1.koma.jaeri.go.jp ユーザ ID
localhost ユーザ ID
```

localhost を含め, 使用しないホストのエントリは省略可能.

尚, ユーザ ID は, 各自のユーザ ID に置き換えること.

E.9 vppsys01 (富士通 VP2400) の設定

各自のユーザ ID でログインし, エディタで .rhosts ファイルを編集する. 内容は, 以下の通り. "#" で開始する行はコメントである.

```
hi00011 ユーザ ID
fu00011 ユーザ ID
ne00011 ユーザ ID
cr00011 ユーザ ID
# ibmsp50a.koma.jaeri.go.jp
133.53.188.9 ユーザ ID
ccsemga ユーザ ID
# desr01.koma.jaeri.go.jp ユーザ ID
133.53.185.27 ユーザ ID
# apsvr03.tokai.jaeri.go.jp
cnet0011 ユーザ ID
vppsys01 ユーザ ID
paragf ユーザ ID
nanasvr ユーザ ID
# hepta.naka.jaeri.go.jp
157.111.159.21 ユーザ ID stasrv1.koma.jaeri.go.jp ユーザ ID
localhost ユーザ ID
```

localhost を含め, 使用しないホストのエントリは省略可能.
尚, ユーザ ID は, 各自のユーザ ID に置き換えること.

E.10 paragf (Intel Paragon フロントエンド) の設定

各自のユーザ ID でログインし, エディタで .rhosts ファイルを編集する. 内容は, 以下の通り. "#" で開始する行はコメントである.

```
# hi00011.koma.jaeri.go.jp
# fu00011.koma.jaeri.go.jp
# ne00011.koma.jaeri.go.jp
# cr00011.koma.jaeri.go.jp
# ibmsp50a.koma.jaeri.go.jp
# ccsemge.koma.jaeri.go.jp
# desr01.koma.jaeri.go.jp
apsvr03 ユーザ ID
vppsys01 ユーザ ID
nanasvr ユーザ ID
hepta ユーザ ID
stasrv1.koma.jaeri.go.jp ユーザ ID
localhost ユーザ ID
```

paragf は接続可能なホストを制限しており, hi00011, fu00011, ne00011, cr00011, ibmsp50, ccsemge, desr01 から MPI_Comm_spawn のプロセス生成先ホストに指定するためには paragf の設定が必要である (Paragon から他の計算機へのプロセス生成は可能). localhost を含め, 使用しないホストのエントリは省略可能.
尚, ユーザ ID は, 各自のユーザ ID に置き換えること.

E.11 nanasvr (IBM SP2) の設定

各自のユーザ ID でログインし, エディタで .rhosts ファイルを編集する. 内容は, 以下の通り. "#" で開始する行はコメントである.

```
hi00011.koma.jaeri.go.jp ユーザ ID
fu00011.koma.jaeri.go.jp ユーザ ID
ne00011.koma.jaeri.go.jp ユーザ ID
cr00011.koma.jaeri.go.jp ユーザ ID
# ibmsp50a.koma.jaeri.go.jp
133.53.188.9 ユーザ ID
# ccsemge.koma.jaeri.go.jp
133.53.184.10 ユーザ ID
desr01.koma.jaeri.go.jp ユーザ ID
apsvr03.tokai.jaeri.go.jp ユーザ ID
vppsys01.tokai.jaeri.go.jp ユーザ ID
paragf.naka.jaeri.go.jp ユーザ ID
nanasvr.naka.jaeri.go.jp ユーザ ID
hepta.naka.jaeri.go.jp ユーザ ID
stasrv1.koma.jaeri.go.jp ユーザ ID
localhost ユーザ ID
```

localhost を含め, 使用しないホストのエントリは省略可能.
尚, ユーザ ID は, 各自のユーザ ID に置き換えること.

E.12 hepta (DEC Alpha) の設定

各自のユーザ ID でログインし, エディタで .rhosts ファイルを編集する. 内容は, 以下の通り. "#" で開始する行はコメントである.

```
hi00011.koma.jaeri.go.jp ユーザ ID
fu00011.koma.jaeri.go.jp ユーザ ID
ne00011.koma.jaeri.go.jp ユーザ ID
cr00011.koma.jaeri.go.jp ユーザ ID
# ibmsp50a.koma.jaeri.go.jp
133.53.188.9 ユーザ ID
# ccsemge.koma.jaeri.go.jp
133.53.184.10 ユーザ ID
desr01.koma.jaeri.go.jp ユーザ ID
apsvr03.tokai.jaeri.go.jp ユーザ ID
# vppsys01.tokai.jaeri.go.jp ユーザ ID
paragf ユーザ ID
nanasvr ユーザ ID
hepta.naka.jaeri.go.jp ユーザ ID
mars.naka.jaeri.go.jp ユーザ ID
saturn.naka.jaeri.go.jp ユーザ ID stasrv1.koma.jaeri.go.jp ユーザ ID
localhost ユーザ ID
```

vppsys01 から hepta に対して MPI_Comm_spawn による動的プロセス生成を行うためには, vppsys01 の設定が必要である (hepta から vppsys01 へのプロセス生成は可能). クラスタを構成するため, hepta, mars, saturn のエントリは省略不可. localhost を含め, 使用しないホストのエントリは省略可能.
尚, ユーザ ID は, 各自のユーザ ID に置き換えること.

E.13 mars, saturn (DEC Alpha) の設定

各自のユーザ ID でログインし, エディタで .rhosts ファイルを編集する. 本指定は, hepta, mars, saturn の 3 台でクラスタを構成するために必要である.

```
hepta.naka.jaeri.go.jp ユーザ ID
mars.naka.jaeri.go.jp ユーザ ID
saturn.naka.jaeri.go.jp ユーザ ID
```

付録 F　C 言語, Java/Stampi でのサンプルプログラム

F.1　Master/Slave: C 言語版

• マネージャプログラム (Master)

```
\begin{verbatim}
#include <mpi.h>
main(int argc, char *argv[])
{
    int world_size, universe_size;
    MPI_Info info;
    MPI_Comm everyone;          /* intercommunicator */

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    if(world_size != 1) error("Top heavy with management");

    printf("How many processes total?");
    scanf("%d", &universe_size);
    if(universe_size == 1) error("No room to start workers");

    MPI_Info_create(&info);
    MPI_Info_set(info, "host", "hi00011");
    MPI_Comm_spawn("worker", MPI_ARGV_NULL, universe_size - 1, info,
                   MPI_COMM_WORLD, 0, &everyone, MPI_ERRCODES_IGNORE);

    /*
     * 並列コードをここに記述する。コミュニケータ "everyone" は生成された
     * プロセス（グループ間コミュニケータ "everyone" におけるリモート
     * グループ内でランク 0, ... universe_size - 2）間との通信に利用される。
     */

    MPI_Finalize();
    exit(0);
}
```

• ワーカプログラム (Slave)

```
#include <mpi.h>
main(int argc, char *argv[])
{
    int size;
    MPI_Comm parent;

```

```
39      MPI_Init(&argc, &argv);
40      MPI_Comm_get_parent(&parent);
41      if(parent == MPI_COMM_NULL) error("No parent");
42      MPI_Comm_remote_size(parent, &size);
43      if(size != 1) error("Something's wrong with the parent");
44
45      /*
46       * 並列コードをここに記述する。
47       * マネージャは（リモートグループ）parent のランク 0 で表される。
48       * もしワーカ間で通信を行う必要があるときは、MPI_COMM_WORLD を使用する。
49       */
50
51      MPI_Finalize();
52      exit(0);
53  }
```

F.2 Master/Slave: Stampi/Java 版

• マネージャプログラム (Master)

```
import sce.*;
import java.net.*;
import java.io.*;

public class test extends SceWindow
{
    Stampi          mpi;
    MPI_Info        info;
    MPI_Intercomm   everyone;
    int             universe_size;

    public void mpicomm()
    {
        mpi = new Stampi(this);
        try{
            mpi.Init();
        }
        catch(UnknownHostException e){ // WWW サーバホスト不明
        }
        catch(OutOfBufferException e){ // 通信バッファ不正
        }
        catch(SceException e){         // WWW サーバ接続失敗
        }
        catch(IOException e){          // その他通信エラー
        }

        try{
            info = new MPI_Info(mpi);
        }
        catch(IOException e){          // 通信エラー
        }

        try{
            info.Set("host", "hi00011");
        }

        catch(IOException e){          // 通信エラー
        }

        //universe_size を決定（ダイアログなどで）するコードを追加

```

```
42          try{
43              everyone = mpi.COMM_WORLD.Spawn("worker", mpi.ARGV_NULL,
44                                          universe_size - 1, info, 0);
45          }
46          catch(IOException e){           // 通信エラー
47          }
48
49          // 並列コードをここに記述する。コミュニケータ "everyone"は生成された
50          // プロセス（グループ間コミュニケータ "everyone"におけるリモート
51          // グループ内でランク 0, ... universe_size - 2) 間との通信に利用される。
52
53          try{
54              mpi.Finalize();
55          }
56          catch(IOException e){           // 通信エラー
57          }
58      }
59  }
```

F.3 Client/Server: C 言語版

• サーバプログラム

```
#include <mpi.h>
main(int argc, char *argv[])
{
    MPI_Comm client
    MPI_Status status;
    char port_name[MPI_MAX_PORT_NAME];
    double buf[MAX_DATA];
    int size, again;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if(world_size != 1) error("Server too big");

    MPI_Open_port(MPI_INFO_NULL, port_name);
    printf("server available at %s", port_name);
    while(1){
        MPI_Comm_accept(port_name, MPI_INFO_NULL, 0, MPI_COMM_WORLD,
                        &client);
        again = 1;
        while(again){
            MPI_Recv(buf, MAX_DATA, MPI_DOUBLE, MPI_ANY_SOURCE,
                     MPI_ANY_TAG, client, &status);
            switch(status.MPI_TAG){
              case 0: MPI_Finalize();
                      return 0;
              case 1: MPI_Comm_disconnect(&client);
                      again = 0;
                      break;
              default: /* 何らかの処理 */
            }
        }
    }
}
```

• クライアントプログラム

```
#include <mpi.h>
main(int argc, char *argv[])
{
    MPI_Comm server;
    double buf[MAX_DATA];
    char port_name[MPI_MAX_PORT_NAME];
```

```
42      int done, tag, n;
43
44      MPI_Init(&argc, &argv);
45      strcpy(port_name, argv[1]); /* コマンドライン引数にサーバポートを指定 */
46      MPI_Comm_connect(port_name, MPI_INFO_NULL, 0, MPI_COMM_WORLD, &server);
47      done = 0;
48      while(!done){
49          tag = 2;
50          /* 何らかの処理 (buf と n を設定) */
51          MPI_Send(buf, n, MPI_DOUBLE, 0, tag, server);
52          /* 何らかの処理 (処理終了で done = 1) */
53      }
54      MPI_Send(buf, 0, MPI_DOUBLE, 0, 1, server);
55      MPI_Comm_disconnect(&server);
56      MPI_Finalize();
57      exit(0);
58  }
```

This is a blank page.

国際単位系(SI)と換算表

表1　SI基本単位および補助単位

量	名称	記号
長さ	メートル	m
質量	キログラム	kg
時間	秒	s
電流	アンペア	A
熱力学温度	ケルビン	K
物質量	モル	mol
光度	カンデラ	cd
平面角	ラジアン	rad
立体角	ステラジアン	sr

表2　SIと併用される単位

名称	記号
分，時，日	min，h，d
度，分，秒	°，′，″
リットル	l，L
トン	t
電子ボルト	eV
原子質量単位	u

$1\ \mathrm{eV}=1.60218\times10^{-19}\ \mathrm{J}$

$1\ \mathrm{u}=1.66054\times10^{-27}\ \mathrm{kg}$

表3　固有の名称をもつSI組立単位

量	名称	記号	他のSI単位による表現
周波数	ヘルツ	Hz	s^{-1}
力	ニュートン	N	$m\cdot kg/s^2$
圧力，応力	パスカル	Pa	N/m^2
エネルギー，仕事，熱量	ジュール	J	$N\cdot m$
工率，放射束	ワット	W	J/s
電気量，電荷	クーロン	C	$A\cdot s$
電位，電圧，起電力	ボルト	V	W/A
静電容量	ファラド	F	C/V
電気抵抗	オーム	Ω	V/A
コンダクタンス	ジーメンス	S	A/V
磁束	ウェーバ	Wb	$V\cdot s$
磁束密度	テスラ	T	Wb/m^2
インダクタンス	ヘンリー	H	Wb/A
セルシウス温度	セルシウス度	℃	
光束	ルーメン	lm	$cd\cdot sr$
照度	ルクス	lx	lm/m^2
放射能	ベクレル	Bq	s^{-1}
吸収線量	グレイ	Gy	J/kg
線量当量	シーベルト	Sv	J/kg

表4　SIと共に暫定的に維持される単位

名称	記号
オングストローム	Å
バーン	b
バール	bar
ガル	Gal
キュリー	Ci
レントゲン	R
ラド	rad
レム	rem

$1\ \text{Å}=0.1\ \mathrm{nm}=10^{-10}\ \mathrm{m}$

$1\ \mathrm{b}=100\ \mathrm{fm}^2=10^{-28}\ \mathrm{m}^2$

$1\ \mathrm{bar}=0.1\ \mathrm{MPa}=10^5\ \mathrm{Pa}$

$1\ \mathrm{Gal}=1\ \mathrm{cm/s}^2=10^{-2}\ \mathrm{m/s}^2$

$1\ \mathrm{Ci}=3.7\times10^{10}\ \mathrm{Bq}$

$1\ \mathrm{R}=2.58\times10^{-4}\ \mathrm{C/kg}$

$1\ \mathrm{rad}=1\ \mathrm{cGy}=10^{-2}\ \mathrm{Gy}$

$1\ \mathrm{rem}=1\ \mathrm{cSv}=10^{-2}\ \mathrm{Sv}$

表5　SI接頭語

倍数	接頭語	記号
10^{18}	エクサ	E
10^{15}	ペタ	P
10^{12}	テラ	T
10^{9}	ギガ	G
10^{6}	メガ	M
10^{3}	キロ	k
10^{2}	ヘクト	h
10^{1}	デカ	da
10^{-1}	デシ	d
10^{-2}	センチ	c
10^{-3}	ミリ	m
10^{-6}	マイクロ	μ
10^{-9}	ナノ	n
10^{-12}	ピコ	p
10^{-15}	フェムト	f
10^{-18}	アト	a

（注）

1. 表1－5は「国際単位系」第5版，国際度量衡局 1985年刊行による。ただし，1 eVおよび1 uの値はCODATAの1986年推奨値によった。
2. 表4には海里，ノット，アール，ヘクタールも含まれているが日常の単位なのでここでは省略した。
3. barは，JISでは流体の圧力を表わす場合に限り表2のカテゴリーに分類されている。
4. EC閣僚理事会指令ではbar，barnおよび「血圧の単位」mmHgを表2のカテゴリーに入れている。

換算表

力	N(=10^5dyn)	kgf	lbf
	1	0.101972	0.224809
	9.80665	1	2.20462
	4.44822	0.453592	1

粘度　1 Pa·s(N·s/m²)=10 P(ポアズ)(g/(cm·s))

動粘度　1 m²/s=10⁴St(ストークス)(cm²/s)

圧力	MPa(=10 bar)	kgf/cm²	atm	mmHg(Torr)	lbf/in²(psi)
	1	10.1972	9.86923	7.50062×10^3	145.038
	0.0980665	1	0.967841	735.559	14.2233
	0.101325	1.03323	1	760	14.6959
	1.33322×10^{-4}	1.35951×10^{-3}	1.31579×10^{-3}	1	1.93368×10^{-2}
	6.89476×10^{-3}	7.03070×10^{-2}	6.80460×10^{-2}	51.7149	1

エネルギー・仕事・熱量	J(=10^7erg)	kgf·m	kW·h	cal(計量法)	Btu	ft·lbf	eV
	1	0.101972	2.77778×10^{-7}	0.238889	9.47813×10^{-4}	0.737562	6.24150×10^{18}
	9.80665	1	2.72407×10^{-6}	2.34270	9.29487×10^{-3}	7.23301	6.12082×10^{19}
	3.6×10^6	3.67098×10^5	1	8.59999×10^5	3412.13	2.65522×10^6	2.24694×10^{25}
	4.18605	0.426858	1.16279×10^{-6}	1	3.96759×10^{-3}	3.08747	2.61272×10^{19}
	1055.06	107.586	2.93072×10^{-4}	252.042	1	778.172	6.58515×10^{21}
	1.35582	0.138255	3.76616×10^{-7}	0.323890	1.28506×10^{-3}	1	8.46233×10^{18}
	1.60218×10^{-19}	1.63377×10^{-20}	4.45050×10^{-26}	3.82743×10^{-20}	1.51857×10^{-22}	1.18171×10^{-19}	1

1 cal = 4.18605 J(計量法)

= 4.184 J　(熱化学)

= 4.1855 J　(15 °C)

= 4.1868 J(国際蒸気表)

仕事率　1 PS(仏馬力)

= 75 kgf·m/s

= 735.499 W

放射能	Bq	Ci
	1	2.70270×10^{-11}
	3.7×10^{10}	1

吸収線量	Gy	rad
	1	100
	0.01	1

照射線量	C/kg	R
	1	3876
	2.58×10^{-4}	1

線量当量	Sv	rem
	1	100
	0.01	1

（86年12月26日現在）

異機種並列計算機間通信ライブラリ：Stampi-利用手引書　第二版