FUJITSU

3.3. MPI

3.3.1. MPIの基本的通信と入出力

富士通株式会社　　志田 直之

Contents

FUJITSU

- MPI概要
  - MPIの歴史
  - MPIライブラリの系譜
- 1対1通信
  - ブロッキング通信とノンブロッキング通信
  - デッドロック
  - 4つの通信モード
  - 1対1通信プロトコル
- 集団通信関数
  - 集団通信とは
  - 集団通信の機能
  - 実装アルゴリズムの例
    - Allgather
    - Alltoall
- MPI-IO
  - HPCにおけるIOアーキテクチャ
  - アプリの状況
  - Data Model (HDF5, NetCDF)
  - MPI-IO規格
  - ROMIO
  - ストライド転送

FUJITSU

MPI概要

- MPIの歴史
- MPIライブラリの系譜

MPIの歴史

FUJITSU

1992/04 MPI Forum設立

1994/10 MPI forumよりMPI-1.0がリリースされる。
MPIライブラリの基本的な128関数をサポート。

1995/07 MPI forumよりMPI-1.1がリリースされる。

1995/07 MPI forumよりMPI-2.0(MPI-1.2を含む)がリリースされる。
MPI-IO・動的プロセス生成・片側通信・ Fortran/C++対応などが追加。

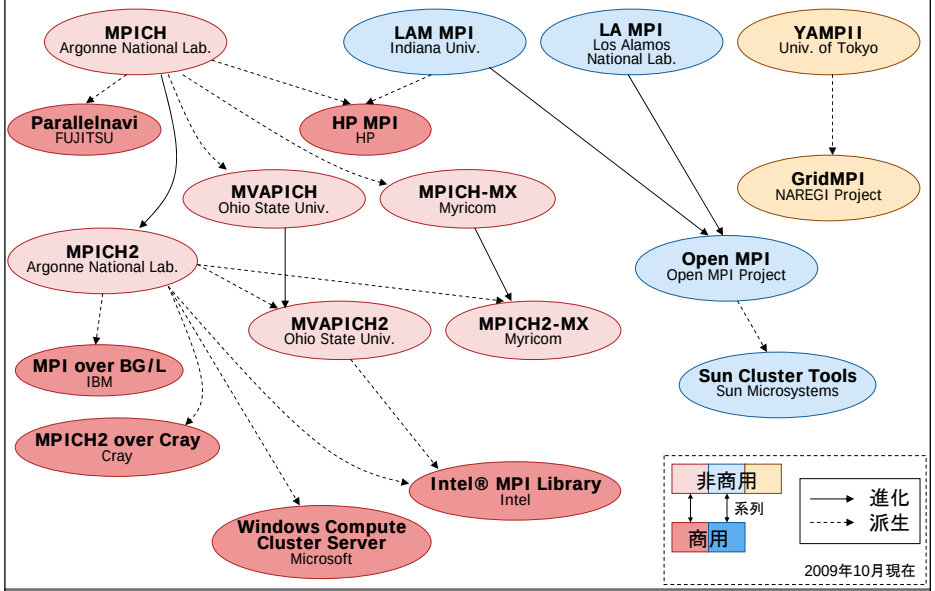
MPI-1.2の分類

1対1通信	39
集団通信	16
派生データ型	13
コミュニケータ	30
プロセス・トポロジー	16
環境管理	14

2008/09 MPI forumよりMPI-2.1がリリースされる。
MPI-2.0の曖昧だった規格を修正する

現在、MPI-2.2, MPI-3.0が検討されている。
MPI-2.2: MPI-2.0に若干の仕様追加を行う。
MPI-3.0: 将来のMPIに向けて、大きな仕様変更を予定。

MPIライブラリの系譜



1対1通信

- ブロッキング通信とノンブロッキング通信
- デッドロック
- 4つの通信モード
- 1対1通信プロトコル

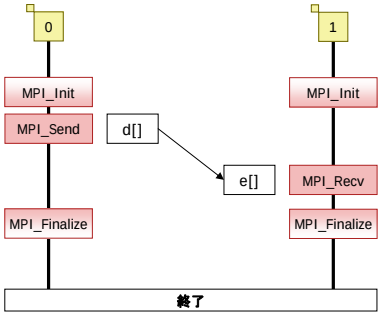
1対1通信とは

■ 1対1通信とは、送信と受信が一対となり、通信を行います。

test.c

```
#include "mpi.h"
main(int argc, char *argv[])
{
    int size, rank;
    double d[10], e[10];
    MPI_Status status;

    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    switch(rank){
    case(0):
        MPI_Send(d, 10, MPI_DOUBLE, 1,
1000, MPI_COMM_WORLD); //to 1
        break;
    case(1):
        MPI_Recv(e, 10, MPI_DOUBLE, 0,
1000, MPI_COMM_WORLD, &status); // from 0
        break;

    }

    MPI_Finalize();
}
```

```
$ mpicc test.c
$ mpiexec -n 2 -hostfile mpd.hosts ./a.out
```



1対1通信の種類

■ 1対1通信のAPI

	ブロッキング		ノンブロッキング	
	送信	受信	送信	受信
標準モード	MPI_Send	MPI_Recv	MPI_Isend	MPI_Irecv
バッファモード	MPI_Bsend		MPI_Ibsend	
同期モード	MPI_Ssend		MPI_Issend	
レディモード	MPI_Rsend		MPI_Irsend	
送受信	MPI_Sendrecv			
	MPI_Sendrecv_replace			

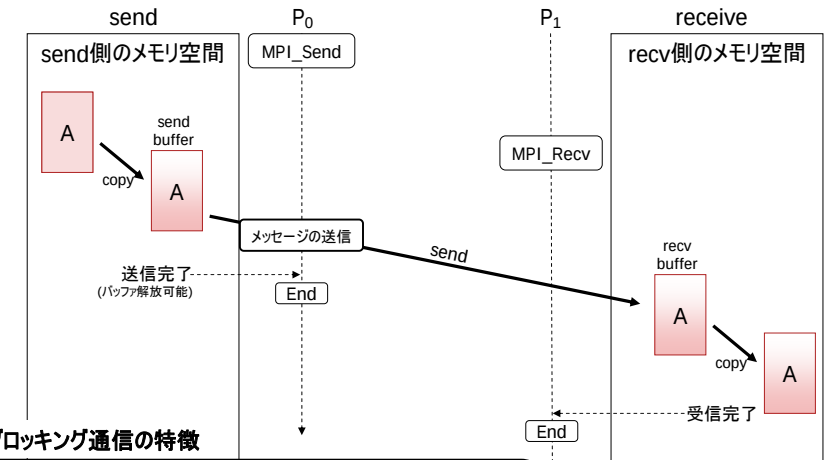
■ ブロッキング
- 送信側=送信バッファを解放しても良いタイミング
- 受信側=受信バッファを解放しても良いタイミング

  になるまで、送信関数・受信関数から復帰しません。すなわち、送信の場合はインタコネクト上にデータがすべて乗った状態、受信の場合はインタコネクトからデータをすべて受け取ったデータが実メモリに書き込まれたことが保証されます。

■ ノンブロッキング
- 送信処理・受信処理を開始する宣言のみで、送信関数・受信関数から復帰します。実際のデータの同期は、MPI_Test関数やMPI_Wait関数などを利用して、ユーザー自身が保証する必要があります。

ブロッキング通信

FUJITSU

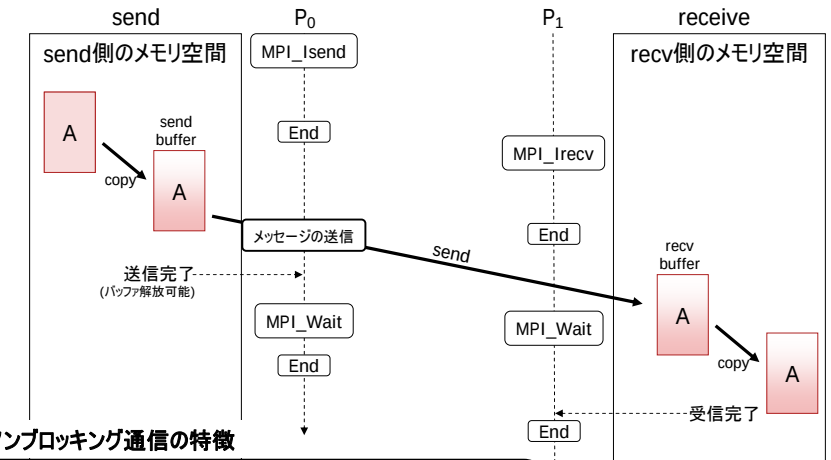


ブロッキング通信の特徴

- 通信処理が終了するまで送受信関数から復帰しない。
- 送受信の対応が簡単になるので、プログラムがわかりやすくなる。
- プログラムの設計次第で、デッドロックの危険性が常にある。

ノンブロッキング通信

FUJITSU



ノンブロッキング通信の特徴

- 通信処理が完了する前に、送受信関数から復帰できるため、通信と計算処理をオーバーラップできる可能性がある。
- ユーザー責任で、ノンブロッキング通信の完了を確認する必要がある。

デッドロック

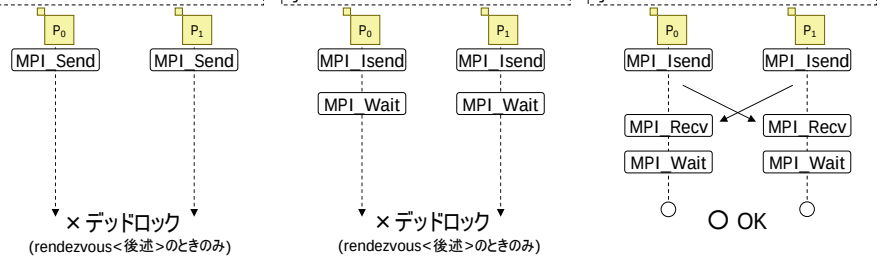
FUJITSU

- 送受信処理が待ち状態のため、完了できない状態
  - デッドロックするプログラムは?

```
switch(RANK){
case 0:
  MPI_Send(To=1,TAG=100);
  MPI_Recv(To=1,TAG=101);
  break;
case 1:
  MPI_Send(To=0,TAG=101);
  MPI_Recv(To=0,TAG=100);
  break;
}
```

```
switch(RANK){
case 0:
  MPI_Isend(To=1,TAG=100);
  MPI_Wait();
  MPI_Recv(To=1,TAG=101);
  break;
case 1:
  MPI_Isend(To=0,TAG=101);
  MPI_Wait();
  MPI_Recv(To=0,TAG=100);
  break;
}
```

```
switch(RANK){
case 0:
  MPI_Isend(To=1,TAG=100);
  MPI_Recv(To=1,TAG=101);
  MPI_Wait();
  break;
case 1:
  MPI_Isend(To=0,TAG=101);
  MPI_Recv(To=0,TAG=100);
  MPI_Wait();
  break;
}
```

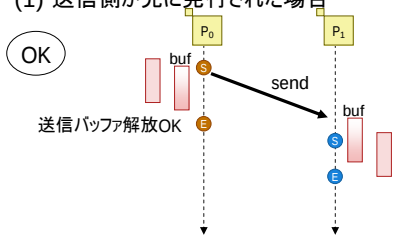


4つの送信モード(1/4)

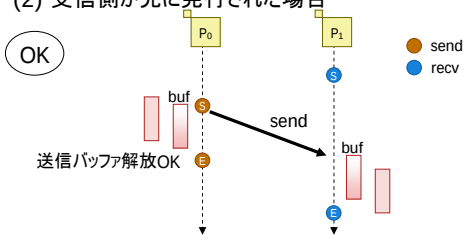
FUJITSU

- 標準モード
  - 対応する受信の順番に関わらず、送信処理を開始することができる。
  - 送信するメッセージがバッファリングされるかどうかは、処理系依存となる。
  - 送信バッファが再利用可能になった場合に、送信処理を完了することができる。
    - メッセージが受信側でバッファリングされる場合
      対応する受信関数が発行される前に、送信関数を復帰することができる。
    - メッセージが受信側でバッファリングされない場合
      対応する受信関数が発行されると、送信関数を復帰することができる。

(1) 送信側が先に発行された場合



(2) 受信側が先に発行された場合

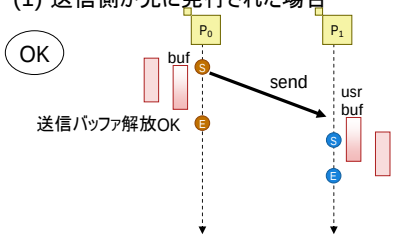


4つの送信モード(2/4)

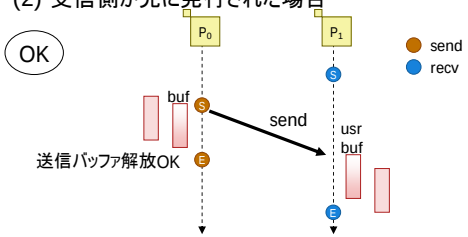
FUJITSU

- バッファモード
  - 対応する受信の順番に関わらず、送信処理を開始することができる。
  - 送信するメッセージは、受信側にバッファリングされる。受信側のバッファアロケーションは、ユーザーが行う必要がある。(MPI_Buffer_attach)
  - 受信バッファに十分な大きさがない場合、送信関数はエラーとなる。
  - 送信バッファが再利用可能になった場合に、送信処理を完了することができる。
  - 送信処理は、対応する受信が発行されなくても、送信処理を完了することができる。

(1) 送信側が先に発行された場合



(2) 受信側が先に発行された場合

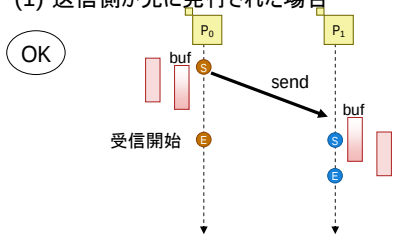


4つの送信モード(3/4)

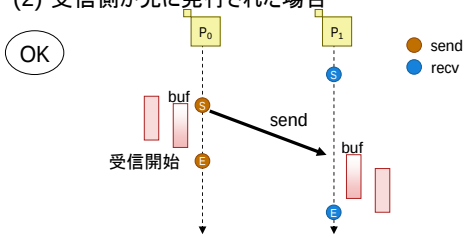
FUJITSU

- 同期モード
  - 対応する受信の順番に関わらず、送信処理を開始することができる。
  - 送信処理は、対応する受信が発行され、さらに、受信操作が開始された場合に、完了することができる。

(1) 送信側が先に発行された場合



(2) 受信側が先に発行された場合

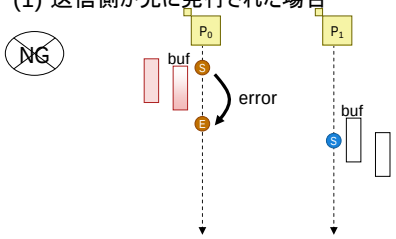


4つの送信モード(4/4)

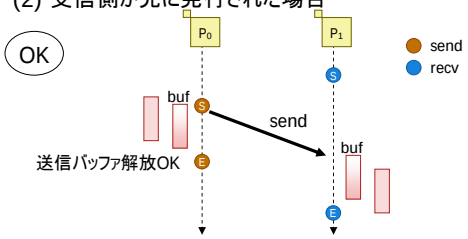
FUJITSU

- レディ通信モード
  - 対応する受信が先に発行されている場合のみ、送信処理を開始することができる。それ以外の場合は、エラーとなる。

(1) 送信側が先に発行された場合

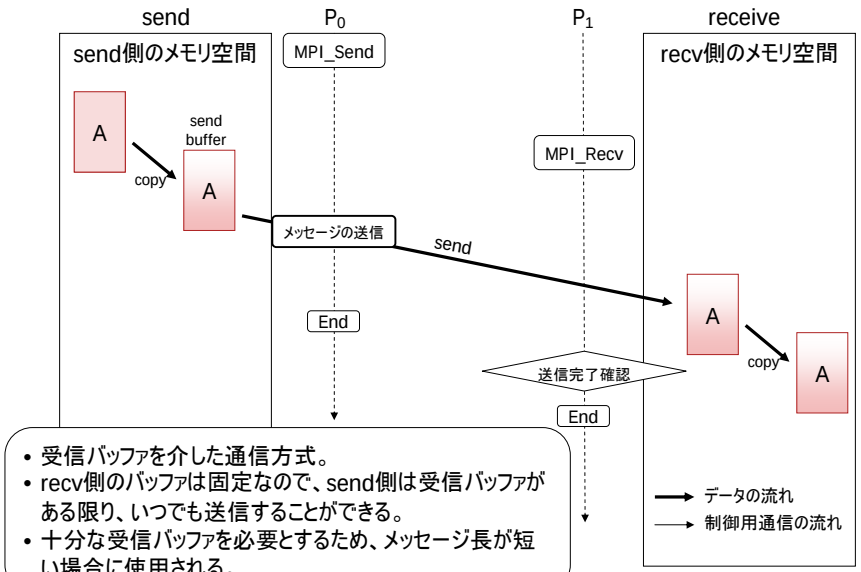


(2) 受信側が先に発行された場合



1対1通信プロトコル(eager send/recv)

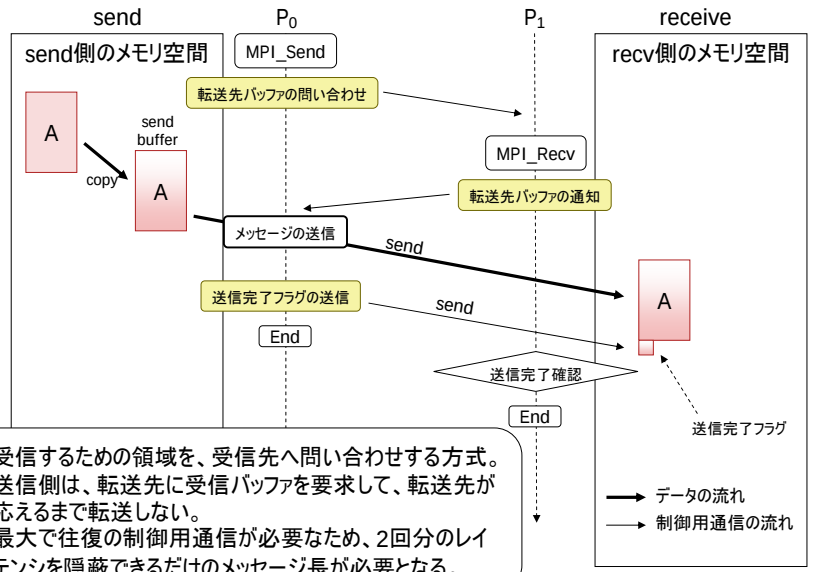
FUJITSU



- 受信バッファを介した通信方式。
- recv側のバッファは固定なので、send側は受信バッファがある限り、いつでも送信することができる。
- 十分な受信バッファを必要とするため、メッセージ長が短い場合に使用される。

1対1通信プロトコル(rendezvous send/recv)

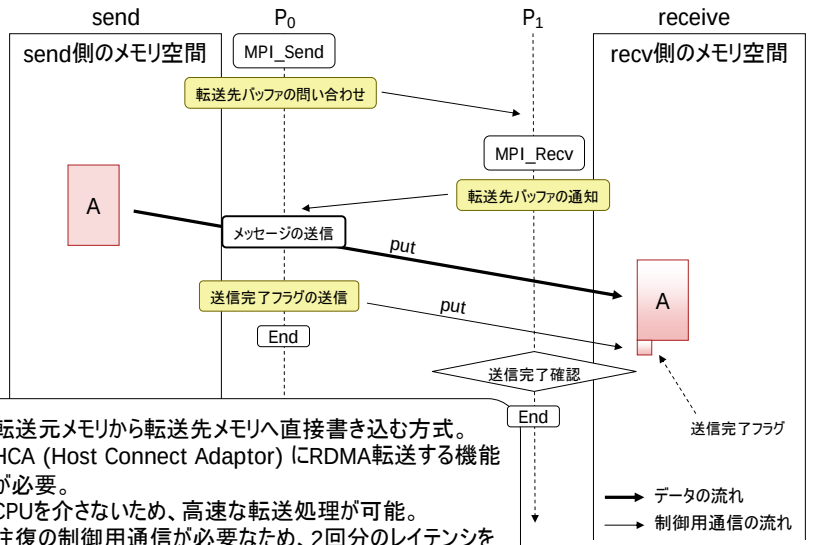
FUJITSU



- 受信するための領域を、受信先へ問い合わせする方式。
- 送信側は、転送先に受信バッファを要求して、転送先が応えるまで転送しない。
- 最大で往復の制御用通信が必要なため、2回分のレイテンシを隠蔽できるだけのメッセージ長が必要となる。

1対1通信プロトコル(RDMA*1 put)

FUJITSU

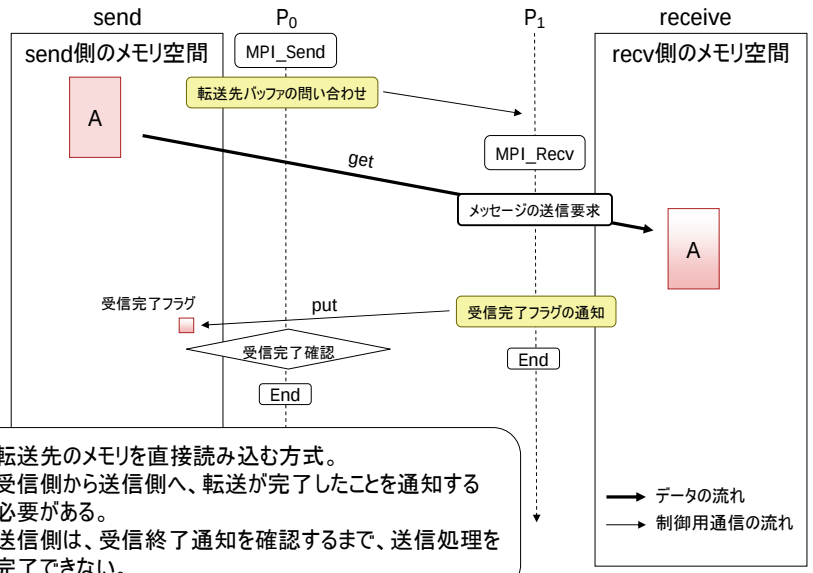


- 転送元メモリから転送先メモリへ直接書き込む方式。
- HCA (Host Connect Adaptor) にRDMA転送する機能が必要。
- CPUを介さないため、高速な転送処理が可能。
- 往復の制御用通信が必要なため、2回分のレイテンシを隠蔽できるだけのメッセージ長が必要となる。

*1RDMA (Remote Direct Memory Access)
ホストCPUの割り込みをほとんど発生させることなく、独立したシステムのメインメモリ間のデータ転送を行う技術。

1対1通信プロトコル(RDMA get)

FUJITSU



- 転送先のメモリを直接読み込む方式。
- 受信側から送信側へ、転送が完了したことを通知する必要がある。
- 送信側は、受信終了通知を確認するまで、送信処理を完了できない。

FUJITSU

集団通信関数

- 集団通信とは
- 集団通信の機能
- 実装アルゴリズムの例
- Allgather
- Alltoall

集団通信とは

FUJITSU

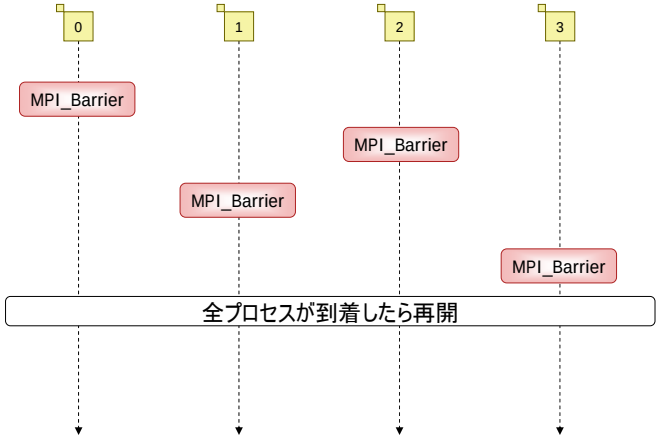
- あるプロセスグループ内でグループ内に関連した通信を定義しています。
- 全てのプロセスグループが、同じ引数を持って呼び出すことが前提となります。
- ブロッキング処理です。

MPIがサポートする集団通信関数

放送	MPI_Bcast
収集	MPI_Gather
	MPI_Gatherv
拡散	MPI_Scatter
	MPI_Scatterv
収集データを全メンバで共有	MPI_Allgather
	MPI_Allgatherv
完全交換	MPI_Alltoall
	MPI_Alltoallv
	MPI_Alltoallw
リダクション	MPI_Reduce
リダクション結果を全メンバで共有	MPI_Allreduce
リダクション結果を全メンバに拡散	MPI_Reduce_scatter
スキャン	MPI_Scan
バリア同期	MPI_Barrier

Barrier (バリア同期)

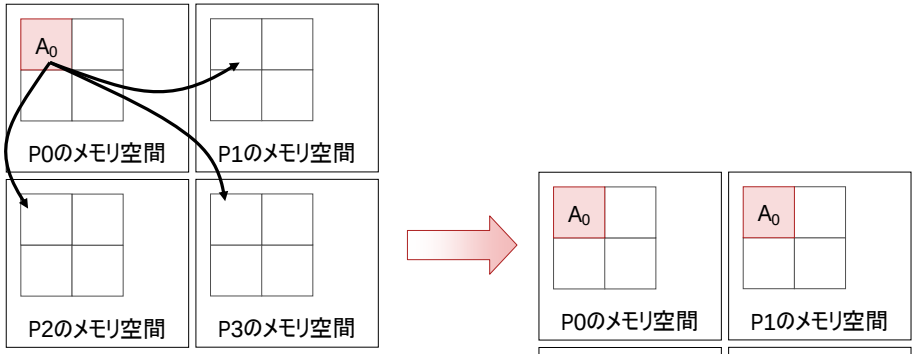
FUJITSU




```
MPI_Barrier(comm)

comm        コミュニケータ
```


Bcast (放送)

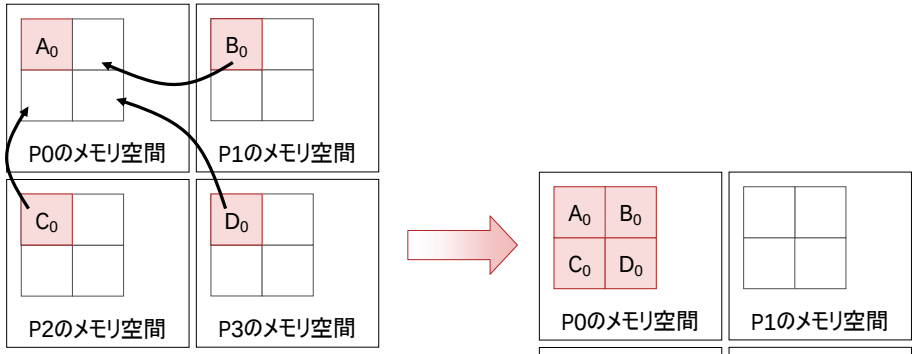
FUJITSU




```
MPI_Bcast(buffer, count, datatype, root, comm)

buffer      バッファの先頭アドレス
count       バッファ内の要素数
datatype    バッファのデータタイプ
root        ルートとなるランク
comm        コミュニケータ
```


Gather (収集)

FUJITSU

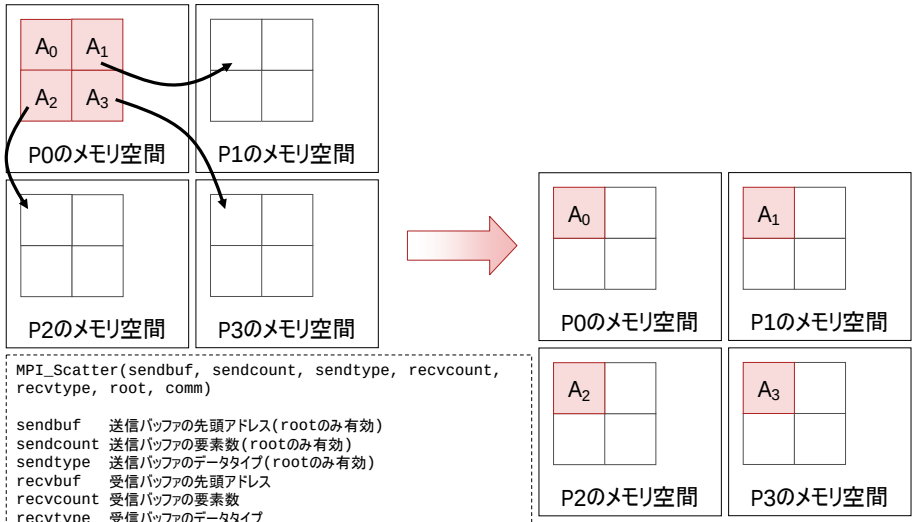



```
MPI_Gather(sendbuf, sendcount, sendtype, recvcount,
recvtype, root, comm)

sendbuf     送信バッファの先頭アドレス
sendcount   送信バッファの要素数
sendtype    送信バッファのデータタイプ
recvbuf     受信バッファの先頭アドレス(rootのみ有効)
recvcount   受信バッファの要素数(rootのみ有効)
recvtype    受信バッファのデータタイプ(rootのみ有効)
root        ルートとなるランク
comm        コミュニケータ
```

可変長のMPI_Gathervもある

Scatter (拡散=Gatherと反対の処理)

FUJITSU

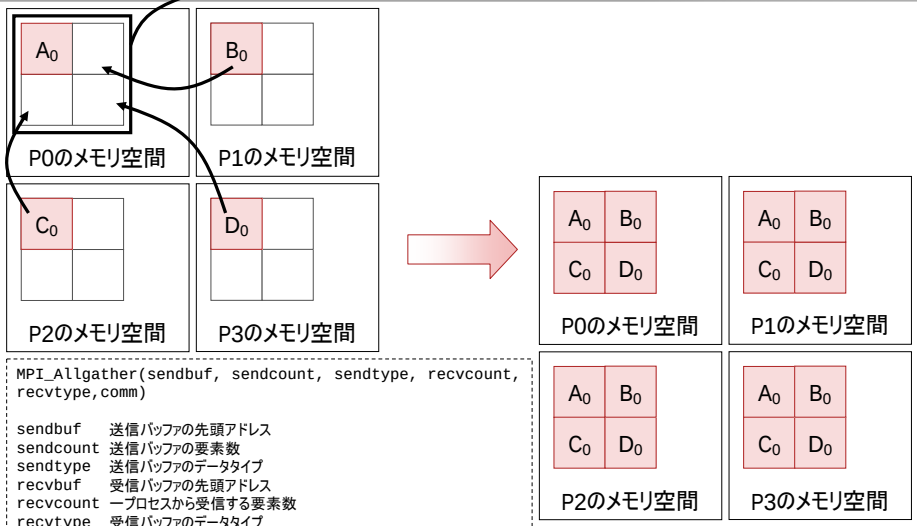



```
MPI_Scatter(sendbuf, sendcount, sendtype, recvcount,
recvtype, root, comm)

sendbuf    送信バッファの先頭アドレス(rootのみ有効)
sendcount  送信バッファの要素数(rootのみ有効)
sendtype   送信バッファのデータタイプ(rootのみ有効)
recvbuf    受信バッファの先頭アドレス
recvcount  受信バッファの要素数
recvtype   受信バッファのデータタイプ
root       ルートとなるランク
comm       コミュニケータ
```

可変長のMPI_Scattervもある

Allgather

FUJITSU

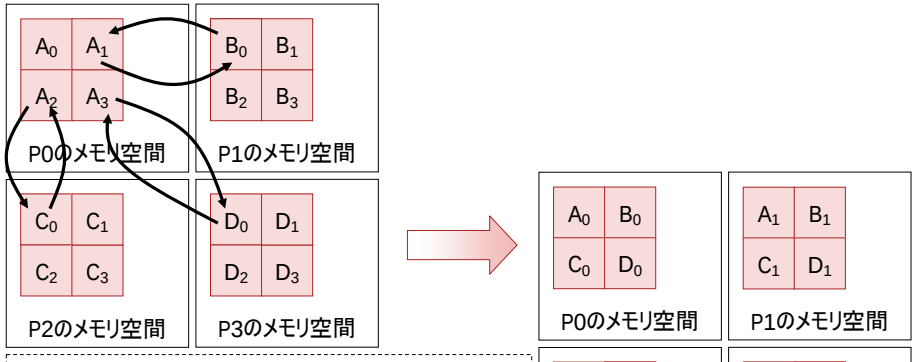



```
MPI_Allgather(sendbuf, sendcount, sendtype, recvcount,
recvtype,comm)

sendbuf    送信バッファの先頭アドレス
sendcount  送信バッファの要素数
sendtype   送信バッファのデータタイプ
recvbuf    受信バッファの先頭アドレス
recvcount  一プロセスから受信する要素数
recvtype   受信バッファのデータタイプ
comm       コミュニケータ
```

(参考) Gather + Bcastと同等の処理

Alltoall

FUJITSU

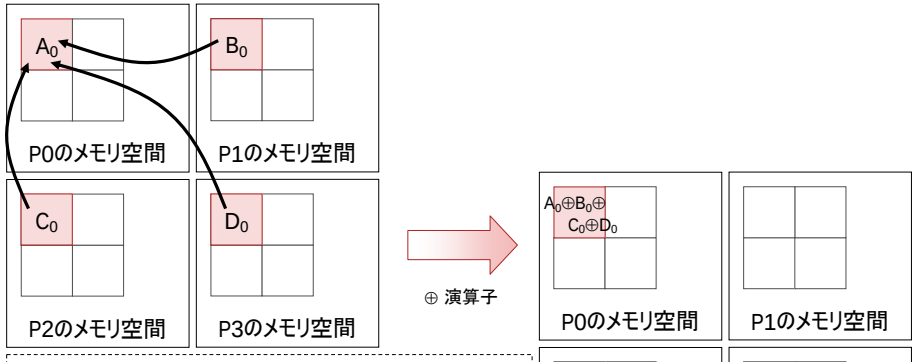



```
MPI_Alltoall(sendbuf, sendcount, sendtype, recvcount,
recvtype, comm)

sendbuf    送信バッファの先頭アドレス
sendcount  各プロセスへ送信する要素数
sendtype   送信バッファのデータタイプ
recvbuf    受信バッファの先頭アドレス
recvcount  一プロセスから受信する要素数
recvtype   受信バッファのデータタイプ
comm       コミュニケータ
```

可変長のMPI_AlltoallvやMPI_Alltoallwもある

Reduce

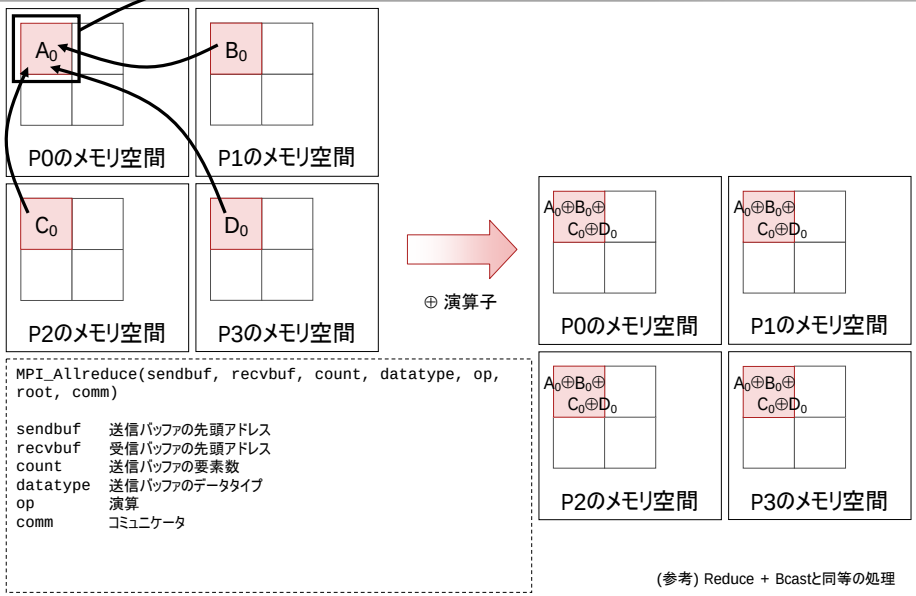
FUJITSU




```
MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root,
comm)

sendbuf    送信バッファの先頭アドレス
recvbuf    受信バッファの先頭アドレス
count      送信バッファの要素数
datatype   送信バッファのデータタイプ
op         演算
root       ルートとなるランク
comm       コミュニケータ
```

Allreduce

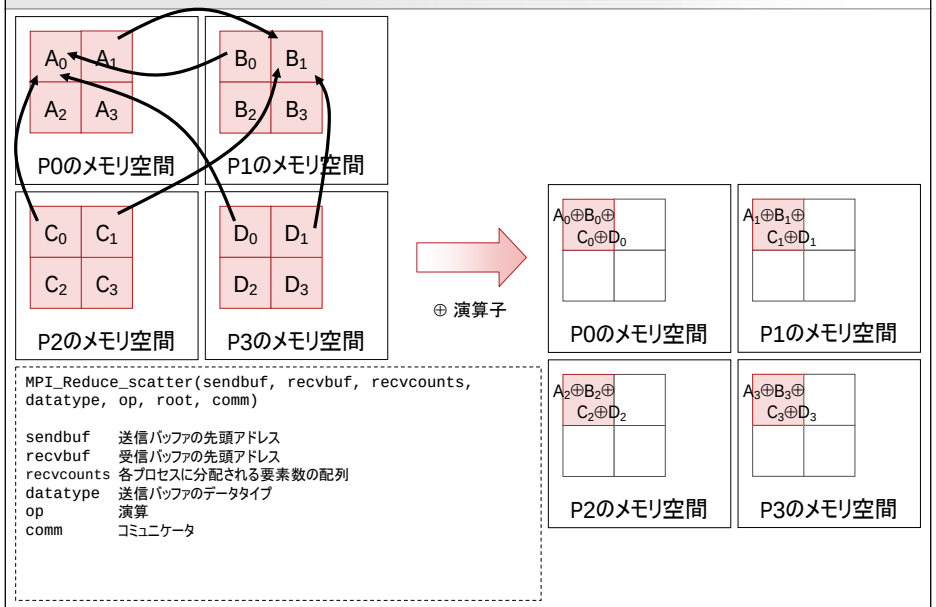
演算結果を各プロセスにコピー




```
MPI_Allreduce(sendbuf, recvbuf, count, datatype, op,
root, comm)

sendbuf    送信バッファの先頭アドレス
recvbuf    受信バッファの先頭アドレス
count      送信バッファの要素数
datatype   送信バッファのデータタイプ
op         演算
comm       コミュニケータ
```

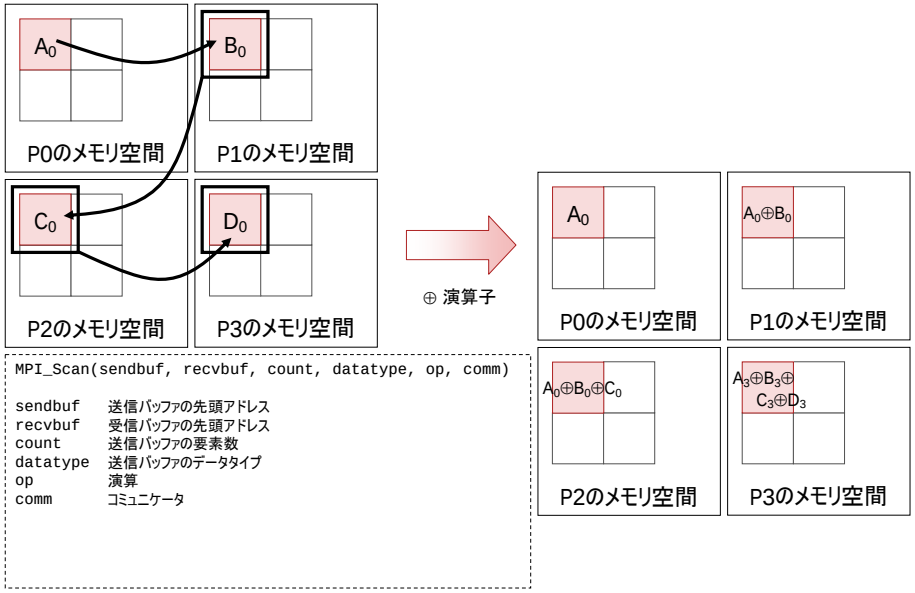
(参考) Reduce + Bcastと同等の処理

Reduce_scatter




```
MPI_Reduce_scatter(sendbuf, recvbuf, recvcounts,
datatype, op, root, comm)

sendbuf    送信バッファの先頭アドレス
recvbuf    受信バッファの先頭アドレス
recvcounts 各プロセスに分配される要素数の配列
datatype   送信バッファのデータタイプ
op         演算
comm       コミュニケータ
```


Scan




```
MPI_Scan(sendbuf, recvbuf, count, datatype, op, comm)

sendbuf    送信バッファの先頭アドレス
recvbuf    受信バッファの先頭アドレス
count      送信バッファの要素数
datatype   送信バッファのデータタイプ
op         演算
comm       コミュニケータ
```


MPIが対応するリダクション演算

Op	Function	対応する型
MPI_MAX	最大値	C整数型・Fortran整数型,実数型,複素数型
MPI_MIN	最小値	C整数型・Fortran整数型,実数型,複素数型
MPI_SUM	和	C整数型・Fortran整数型,実数型,複素数型
MPI_PROD	積	C整数型・Fortran整数型,実数型,複素数型
MPI_LAND	論理積	C整数型,論理型
MPI_BAND	ビット演算の積	C整数型・Fortran整数型,バイト型
MPI_LOR	論理和	C整数型,論理型
MPI_BOR	ビット演算の和	C整数型・Fortran整数型,バイト型
MPI_LXOR	排他的論理和	C整数型,論理型
MPI_BXOR	ビット演算の排他的論理和	C整数型・Fortran整数型,バイト型
MPI_MAXLOC	最大値と位置	
MPI_MINLOC	最小値と位置	

集団通信のアルゴリズム例

FUJITSU

- Allgatherのアルゴリズムの例
  - Ring Algorithm
  - Recursive Doubling Algorithm
  - Bruck Algorithm
  - Neighbor Exchange Algorithm
- Alltoallのアルゴリズムの例
  - Simple Spread Algorithm
  - Pair-wise Exchange Algorithm

Algorithm <Allgather> (1/4)

FUJITSU

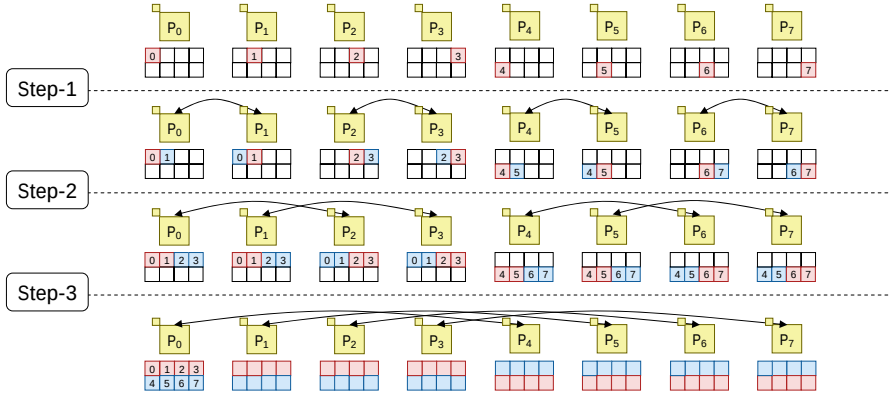
Ring Algorithm

一プロセス分のデータを次々と隣に送る。ステップ数=(P-1)

Algorithm <Allgather> (2/4)

FUJITSU

Recursive Doubling Algorithm

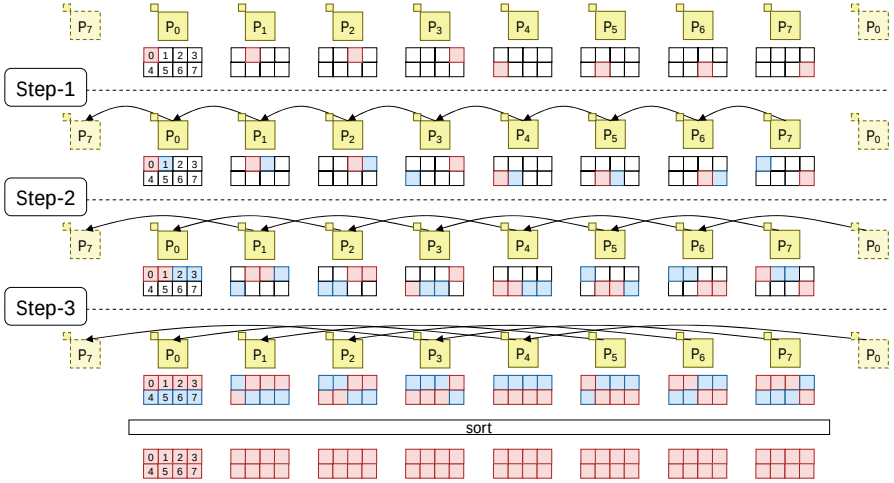


ステップ数(=log2P)が抑えられるが、2べきのプロセス数でないと適用できない。

Algorithm <Allgather> (3/4)

FUJITSU

Bruck Algorithm

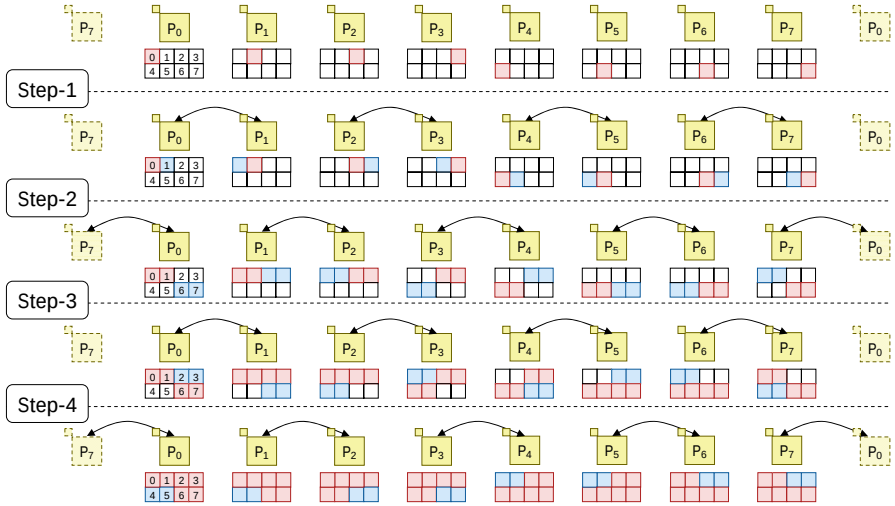


非連続データを扱うため、最後でソートが必要になる。ステップ数(=log2P)。

Algorithm <Allgather> (4/4)

FUJITSU

Neighbor Exchange Algorithm

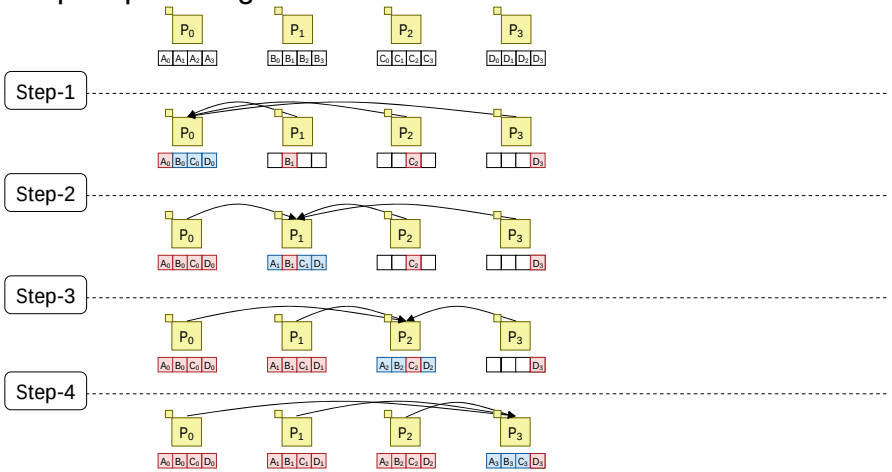


ステップ数(=P/2)はRecursive Doublingより多いが、通信は隣接通信のみ。

Algorithm <Alltoall> (1/2)

FUJITSU

Simple Spread Algorithm



一ステップで、一つのプロセスにすべてのデータを集める。ステップ数=P。

Algorithm <Alltoall> (2/2)

FUJITSU

Pair-wise Exchange Algorithm

一要素ずつ近くから遠くのランクへ転送する。ステップ数=P-1。

FUJITSU

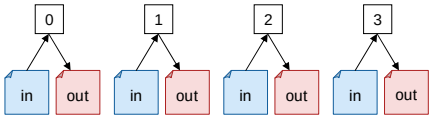
MPI-IO

- HPCにおけるIOアーキテクチャ
- アプリの状況
- Data Model (HDF5, NetCDF)
- MPI-IO規格
- ROMIO

HPCにおけるIO

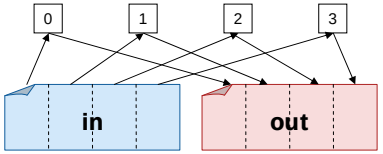
FUJITSU

- 現在のアプリケーションの傾向
  - 1プロセスごとに入力ファイル・出力ファイルが存在



特徴
- プロセス数が少ないうちは、高速にファイルへアクセスできるが、プロセス数が増える(作成するファイル数が増える)に連れて、ファイルシステムへの負荷が増大する。

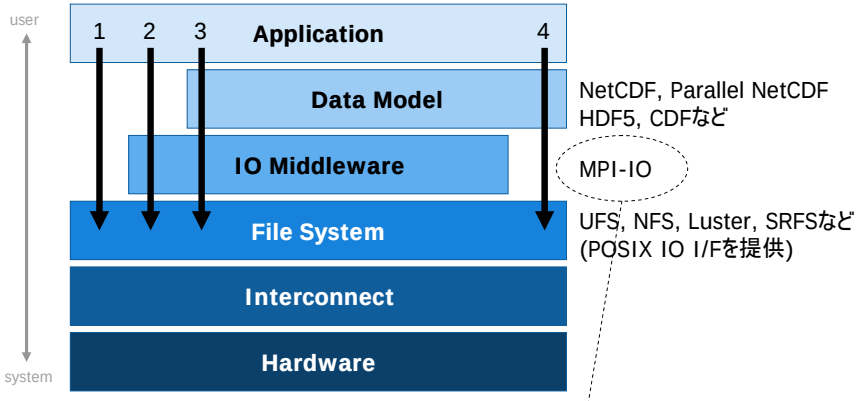
- 将来予測されるアプリケーションの傾向
  - 1アプリケーション全体で入力ファイルと出力ファイルを管理



特徴
- 分散並列上のアプリケーション全体から、一つのファイルへの入出力が可能なため、ファイルシステムへの負荷をかけずに入出力ができる。

HPCにおけるIOアーキテクチャ

FUJITSU



1. POSIX IOでの入出力
2. MPI-IOでの入出力
3. Data ModelがMPI-IOを利用して入出力
4. Data ModelがPOSIX IOを利用して入出力

MPI-IOとは、POSIXインターフェースでは実現できない、
- 複数プロセスが共有ファイルを効率的にアクセスするためのインターフェースを提供
- プロセス間IOのポータビリティ性を提供

している、MPI-2規格の一部です。

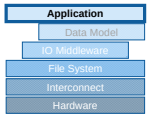
アプリの状況

FUJITSU

Science Domain	Code	Programming Language	Programming Model	I/O Libraries	Math Libraries
Accelerator Design	T3P	C/C++	MPI	NetCDF	MUMPS, ParMETIS, Zoltan
Astrophysics	CHIMERA	F90	MPI	HDF5, pNetCDF	LAPACK
Biology	LAMMPS	C/C++	MPI		FFTW
Chemistry	MADNESS	F90	MPI		BLAS
Chemistry	NWChem	F77, C/C++	MPI, Global Arrays, ARMCI		BLAS, ScaLAPACK, FFTPACK
Chemistry	ORetran	F95	MPI		LAPACK
Climate	CAM	F90, C, CAF	MPI, OpenMP	NetCDF	SciLib
Climate	POP/CICE	F90, CAF	MPI, OpenMP	NetCDF	
Climate	MITgcm	F90, C	MPI, OpenMP	NetCDF	
Combustion	S3D	F90	MPI		
Fusion	AORSA	F77, F90		NetCDF	ScaLAPACK, FFTPACK
Fusion	GTC	F90, C/C++	MPI, OpenMP	MPI-IO, HDF5, NetCDF, XML	PetSC
Fusion	GYRO	F90, Python	MPI	MPI-IO, NetCDF	BLAS, LAPACK, UMFPACK, MUMPS, FFTW, SciLib, ESSL
Geophysics	PFLOTRAN	F90	MPI		BLAS, PetSC
Materials Science	LSMS	F77, F90, C/C++	MPI2	HDF5, XML	BLAS, LAPACK
Materials Science	QBOX	C/C++	MPI	XML	LAPACK, ScaLAPACK, FFTW
Materials Science	QMC	F90	MPI		BLAS, LAPACK, SPRNG
Nanoscience	CASINO	F90	MPI		BLAS
Nanoscience	VASP	F90	MPI		BLAS, ScaLAPACK
Nuclear Energy	NEWTRNX	F90, C/C++, Python		HDF5	LAPACK, PARPACK
Nuclear Physics	CCSD	F90	MPI	MPI-IO	BLAS
QCD	MILC, Chroma	C/C++	MPI		

NCCS High Performance Computing Allocations for FY 2006
Dr. Bronson Messer (Oak Ridge National Laboratory)

- アメリカでの主要アプリの状況は?
  - まだまだ、Fortranが主力
  - 高レベルIOは、HDF5とNetCDFが主流
  - MPI-IOで記述されたアプリが3本存在
  - Parallel NetCDFは間接的にMPI-IOを使用しています。
  - HDF5は間接的にMPI-IOを使用する可能性があります。
  - POSIX IOのジョブもまだまだ多い。

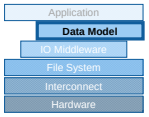


データモデルの役割

FUJITSU

- 多様なAPIの提供
  - C/C++/Fortranに加えて、最近のトレンドはJavaをサポートしています
- ランダムアクセス
  - ランダムアクセスの提供により、目的のデータを高速に取り出すことができます。
- 既存データの読み出し
  - データモデルがサポートする処理系ならば、ファイルを読み書きすることができます。
  - ファイルの転送先やリモートアクセスでも、ファイルを読み書きすることができます。

  ⇒ データファイルの一般提供が可能
- ファイルの圧縮
  - zlibによるデータ圧縮をサポートしています。
- 並列IOによるアクセス
  - より高速化するために、並列IOのサポートが開始されています。(Parallel NetCDFとHDF5)



HDF5の特徴

FUJITSU

- 特徴
  - NCSA(アメリカ国立スーパーコンピュータ応用研究所)で開発された、階層データフォーマット
  - 複雑なデータオブジェクトや大量のメタデータを表現できるデータモデル
  - データオブジェクトの数や大きさに制限のない可搬性の高いファイルフォーマット
  - C, C++, Fortran 90, JavaにAPIを提供
  - Virtual File Layerにより、用途に応じたファイルを出力

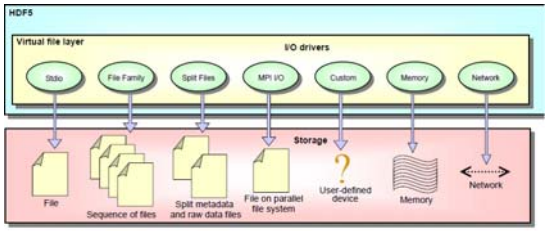
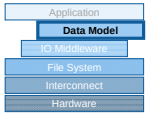


Figure 10.3: The HDF5 virtual file layer (VFL). Through the use of the VFL, an HDF5 file can be stored as a conventional UNIX file or as multiple files (to overcome a 32-bit system limit on file size); it can be stored as two or more files containing separated meta data and raw data; it can be stored as multiple files on a parallel file system; it can be saved in memory or sent over the network; or it can be handled by another non-standard, user-supplied driver.

HDF5 Wins 2002 R&D 100 Award
http://hdf.ncsa.uiuc.edu/HDF5/RD100-2002/All_About_HDF5.pdf

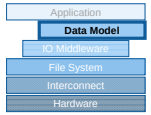


NetCDF・Parallel NetCDFの特徴

FUJITSU

- 特徴
  - UCAR (University Corporation for Atmospheric Research)の、Unidata Centerが開発したデータモデル
  - Network-transparent / Self-Describing
    - NetCDFファイルは、どの環境(たとえばエンディアンが違っても)であっても、ユーザーは変換することなく、データを取り出すことができます。
    - NetCDFファイルをネットワーク転送したり、リモートアクセスすることも可能です。
  - C, C++, FOTRAN77, Fortran90, JavaにAPIを提供
  - 第三者によって、MATLAB, Objective-C, Perl, Python, R, Ruby, Tcl/Tkで利用できるAPIを提供
  - Parallel NetCDFは、NetCDFを拡張して、MPI-IOを使用して並列化

    ⇒内部的に並列IOを使用するため、ユーザーが意識することなく、並列ファイルシステムを有効に活用できます。
  - 主に気象系データの解析に使用



MPI-IOの概要

FUJITSU

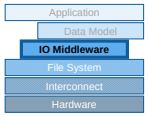
- 集団的操作によるファイルのオープンとクローズ

```
MPI_FILE_OPEN(comm, filename, amode, info, fh)
IN      comm        コミュニケータ
IN      filename    ファイル名
IN      amode       アクセスモード
IN      info        infoオブジェクト
OUT     fh          ファイルハンドル
```

効率的にアクセスするための情報を、ヒントとして処理系に渡すことができます

```
MPI_FILE_CLOSE(fh)
INOUT   fh          ファイルハンドル
```

MPI-IOは、プロセス間で共有する一つのファイルにアクセスすることができます。(プロセスごとに別のファイルを作成することも可能です)



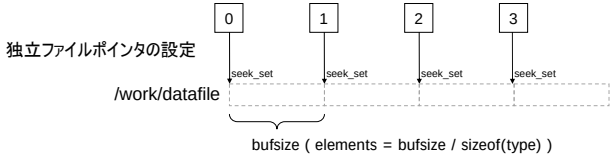
MPI-IOの動作イメージ

FUJITSU

一つのファイルに複数のプロセスから書き込みを行うことができます

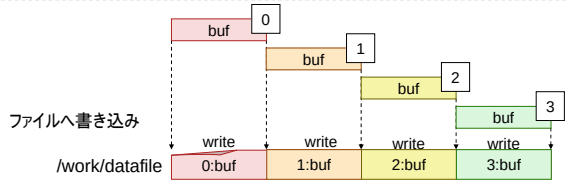
```
MPI_File_open(MPI_COMM_WORLD, "/work/datafile",
    MPI_MODE_CREATE | MPI_MODE_WRONLY,
    MPI_INFO_NULL, &fh);
MPI_File_seek(fh, rank*bufsize, MPI_SEEK_SET);
```

ファイルをオープンして、オフセットを設定する

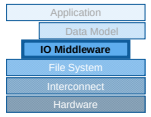



```
MPI_File_write(fh, buf, elements, MPI_INT, &s);
```

MPI_INT型をelements個書き込む




```
MPI_File_close(&fh);
```



ファイルシーク(明示的オフセット)

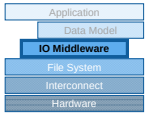
FUJITSU

- 3種類のファイルシークを提供 (1/3)
  - ファイルの先頭を0としたときのオフセット値を指定して、ファイルへアクセスします。ファイルポインタは更新されません。ファイルのオープン時に、MPI_MODE_SEQUENTIALモードを指定した場合は、利用できません。

```
MPI_FILE_WRITE_AT(fh, offset, buf, count,
datatype, status)
INOUT     fh                ファイルハンドル
IN        offset            オフセット
IN        buf               バッファの先頭アドレス
IN        count             バッファの要素数
IN        datatype          バッファのデータタイプ
OUT       status            ステータスオブジェクト
```

```
MPI_FILE_READ_AT(fh, offset, buf, count, datatype,
status)
IN        fh                ファイルハンドル
IN        offset            オフセット
OUT       buf               バッファの先頭アドレス
IN        count             バッファの要素数
IN        datatype          バッファのデータタイプ
OUT       status            ステータスオブジェクト
```

```
MPI_FILE_READ_AT          関連API
MPI_FILE_WRITE_AT
MPI_FILE_IREAD_AT
MPI_FILE_IWRITE_AT
MPI_FILE_READ_AT_ALL
MPI_FILE_WRITE_AT_ALL
MPI_FILE_READ_AT_ALL_BEGIN
MPI_FILE_READ_AT_ALL_END
MPI_FILE_WRITE_AT_ALL_BEGIN
MPI_FILE_WRITE_AT_ALL_END
```



ファイルシーク(独立ファイルポインタ)

FUJITSU

- 3種類のファイルシークを提供 (2/3)
  - 各プロセスが独立したファイルポインタを持って、ファイルへアクセスします。共有ファイルポインタは更新されません。ファイルのオープン時に、MPI_MODE_SEQUENTIALモードを指定した場合は、利用できません。

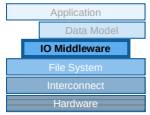
```
MPI_FILE_WRITE(fh, buf, count, datatype, status)
INOUT     fh                ファイルハンドル
IN        buf               バッファの先頭アドレス
IN        count             バッファの要素数
IN        datatype          バッファのデータタイプ
OUT       status            ステータスオブジェクト
```

```
MPI_FILE_READ(fh, buf, count, datatype, status)
INOUT     fh                ファイルハンドル
OUT       buf               バッファの先頭アドレス
IN        count             バッファの要素数
IN        datatype          バッファのデータタイプ
OUT       status            ステータスオブジェクト
```

```
MPI_FILE_SEEK(fh, offset, whence)
INOUT     fh                ファイルハンドル
IN        offset            オフセット
IN        whence            更新モード
```

```
MPI_FILE_GET_POSITION(fh, offset)
IN        fh                ファイルハンドル
OUT       offset            オフセット
```

```
MPI_FILE_READ             関連API
MPI_FILE_WRITE
MPI_FILE_IREAD
MPI_FILE_IWRITE
MPI_FILE_READ_ALL
MPI_FILE_WRITE_ALL
MPI_FILE_READ_ALL_BEGIN
MPI_FILE_READ_ALL_END
MPI_FILE_WRITE_ALL_BEGIN
MPI_FILE_WRITE_ALL_END
```



ファイルシーク(共有ファイルポインタ)

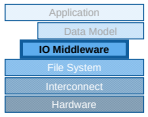
FUJITSU

- 3種類のファイルシークを提供 (3/3)
  - 各プロセス間で共通のファイルポインタを持って、ファイルへアクセスします。同時に共有ファイルポインタを使用したデータアクセスをする場合、シリアライズされて実行されますが、その順序は処理系依存です。順序保証するには、ユーザー自身で同期処理を行う必要があります。

```
MPI_FILE_WRITE_SHARED(fh, buf, count, datatype,
status)
INOUT     fh                ファイルハンドル
IN        offset            オフセット
IN        count             バッファの要素数
IN        datatype          バッファのデータタイプ
OUT       status            ステータスオブジェクト
```

```
MPI_FILE_READ_SHARED(fh, buf, count, datatype,
status)
IN        fh                ファイルハンドル
OUT       buf               バッファの先頭アドレス
IN        count             バッファの要素数
IN        datatype          バッファのデータタイプ
OUT       status            ステータスオブジェクト
```

```
MPI_FILE_READ_SHARED      関連API
MPI_FILE_WRITE_SHARED
MPI_FILE_IREAD_SHARED
MPI_FILE_IWRITE_SHARED
MPI_FILE_READ_ORDERED
MPI_FILE_WRITE_ORDERED
MPI_FILE_READ_ORDERED_BEGIN
MPI_FILE_READ_ORDERED_END
MPI_FILE_WRITE_ORDERED_BEGIN
MPI_FILE_WRITE_ORDERED_END
```



ブロッキングとノンブロッキング

FUJITSU

- ブロッキングIOとノンブロッキングIO

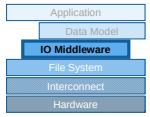
  1対1通信と同様に、ブロッキングIOとノンブロッキングIOが提供されています。
  - ブロッキングIO

    IO要求が完了するまで戻りません。ただし、書き込みバッファを利用するシステムでは、記憶デバイスへの書き込みを保証するものではありません。記憶デバイスへの転送を保証するためには、MPI_FILE_SYNCを使用します。

```
MPI_FILE_SYNC(fh)
INOUT     fh                ファイルハンドル
```

  - ノンブロッキングIO

    IO操作の開始ルーチンを呼び出しますが、完了するまで待ちません。実際のIO処理を行っている間に、他の演算処理をオーバーラップすることができます。ユーザーバッファが再利用可能かどうかは、ノンブロッキング通信と同様に、MPI_WaitやMPI_Testなどを使用して確認します。



集団的データアクセス

FUJITSU

- 非集団的データアクセスと集団的データアクセス

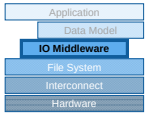
  1対1通信と同様に、非集団的IOと集団的IOが提供されています。

  - 非集団的データアクセス

    処理の完了は、呼び出したプロセスのみに依存します。

  - 集団的データアクセス

    処理の完了は、集団的呼び出しに参加する全てのプロセスに依存します。しかし、大域的データアクセスの場合、MPIライブラリ内に入出力の最適化の余地が十分にある場合は、性能面で有利になる可能性があります。



ファイル情報

FUJITSU

- ファイル情報

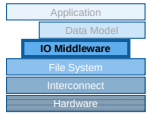
  ユーザーは、最適化の指示として、infoを通じてシステム側にヒントを渡すことができます。

```
MPI_FILE_SET_INFO(fh, info)
INOUT     fh                ファイルハンドル
IN        info              infoオブジェクト
```

```
MPI_FILE_GET_INFO(fh, info_used)
INOUT     fh                ファイルハンドル
OUT       info_used         infoオブジェクト
```

(参考) Fujitsu MPI(V3系)がサポートするinfoオブジェクトのkey

key	value(省略値)	意味
cb_buffer_size	4194304	集団的アクセスに使用する一時バッファの大きさ
cb_nodes	コミュニケータの大きさ	集団的アクセスで実際に入出力を行うプロセス数
ind_rd_buffer_size	4194304	プロセス個別読み込み時のバッファ域の大きさ
ind_wr_buffer_size	524288	プロセス個別書き込み時のバッファ域の大きさ



MPI-IOにおけるデータ表現(1)

FUJITSU

- 3種類のデータ表現
  - MPI-IOは、データファイルのポータビリティのために、3つのデータ表現を用意しています。
  - “native”

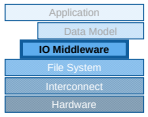
    データ表現は、メモリ中のバイナリイメージと同じです。同一機種内でデータファイルを使用する場合は、精度が保証され、変換コストが省略できる利点があります。
  - “internal”

    ファイル形式は、同一システム内(異機種間であっても)で保証されます。データ表現は、MPIの実装によって決められますが、”external32”として置き換えられる場合があります。
  - “external32”

    データ表現は、他システムのMPIでも保証されます。データ変換が必要な処理系では、精度が失われたり、実行性能の低下を招きます。”external32”は、MPI-2規格で厳密に定義されていますが、一般的なIEEE754+ビッグエンディアン形式(SPARCのデータフォーマット)がイメージされています。

    - 全ての浮動小数点型はIEEE型のビッグエンディアンで表現される。4倍精度は、16バイト(指数部15ビット、仮数部112ビット)とする。
    - 全ての整数型は、2の補数のビッグエンディアンで表現される。
    - FortranのLOGICAL型とC++のbool型は、偽を0、真を非零とする。
    - FortranのCOMPLEX型とDOUBLE COMPLEX型は、二つの浮動小数点フォーマットの組み合わせとする。
    - 文字型はISO 8859-1とし、MPI_WCHARはUnicodeフォーマットとする。
    - NaNは伝搬するものとする。

(注意) Fujitsu MPI(V3系)は、”internal”と”external32”は、”native”で解釈されます。



MPI-IOにおけるデータ表現(2)

FUJITSU

- 3種類のデータ表現の設定と参照

```
MPI_FILE_SET_VIEW(fh, disp, etype, filetype,
datarep, info)
INOUT     fh                ファイルハンドル
IN        disp              オフセット
IN        etype             基本データ型
IN        filetype          ファイル型
IN        datarep           データ表現
IN        info              infoオブジェクト
```

独立ファイルポインタ・共有ファイルポインタは更新される

“native”, “internal”, “external32”のいずれかを指定する

```
MPI_FILE_GET_VIEW(fh, disp, etype, filetype,
datarep)
IN        fh                ファイルハンドル
OUT       disp              オフセット
OUT       etype             基本データ型
OUT       filetype          ファイル型
OUT       datarep           データ表現
```

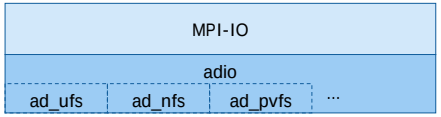
MPIの実装(ROMIO)

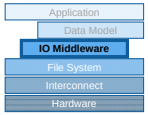
FUJITSU

- ROMIO
  - Argonne National Laboratoryによって実装されたMPI-IO
  - MPI-IO実装のde facto standard
    - MPICH
    - LAM-MPI
    - HP MPI
    - SGI MPI
    - NEC MPI
    - 富士通MPI(V3系から)

    } などに採用済み
  - adioレイヤー構成により、ファイルシステムの追加が容易
  - 対応する豊富なファイルシステム
    - IBM PIOFS
    - Intel PFS
    - HP/Convex HFS
    - SGI XFS
    - NEC SFS
    - PVFS
    - NFS
    - UFS

ROMIOのソフトウェア・スタック





FUJITSU

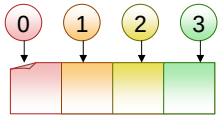
MPIでのファイル出力

- MPI-IO方式
- Split Files方式
- IOマスター方式

MPI-IO方式によるファイル転送

FUJITSU

- MPI-IOを利用して一つのファイルに転送(MPI-IO方式)



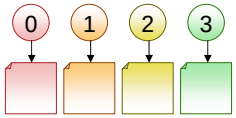

```
MPI_FILE_DELETE(fname);

MPI_File_open(MPI_COMM_WORLD, filename,
    MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
MPI_File_seek(fh, myrank * bufsize, MPI_SEEK_SET);
MPI_File_write(fh, buf, elements, MPI_DOUBLE, &status);
MPI_File_sync(fh);
MPI_File_close(&fh);
```

- ユーザーはプロセスの同期を取ることなく、一つのファイルを作成することができる。
- 多重書き込みをサポートする高速なネットワーク共有ファイルシステムが前提となる。

Split Files方式によるファイル転送

FUJITSU

- プロセスごとにファイルを転送(Split Files方式)



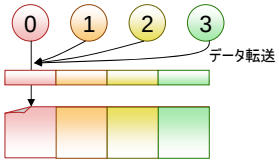

```
sprintf(filename, “%s.%03d", BASENAME, this_rank);
unlink(filename);

fd = open(filename, O_CREAT | O_WRONLY, S_IRUSR | S_IWUSR);
write(fd, (void *)buf, bufsize);
fsync(fd);
close(fd);
```

- 一般的なPOSIX-IOで実現できるため、わかりやすいプログラムで記述できる。
- ファイルがプロセス数個できるため、プロセス増加に伴い、ファイルシステムに負荷がかかる。

IOマスター方式によるファイル転送

FUJITSU

■ ルートに出力データを集めて、一つのファイルを転送(IOマスター方式)




```
unlink(filename);

MPI_Gather(buf, (int)elements, MPI_DOUBLE,
    allbuf, (int)elements, MPI_DOUBLE, 0, MPI_COMM_WORLD);
if(this_rank == 0){
    fd = open(filename, O_CREAT | O_WRONLY, S_IRUSR | S_IWUSR);
    write(fd, (void *)allbuf, allbufsize);
    fsync(fd);
    close(fd);
}
```

- 事前にデータ転送が必要なため、通信コストが高い。
- ルートプロセスの負荷が高く、プロセスの負荷バランスが悪くなる。

FUJITSU

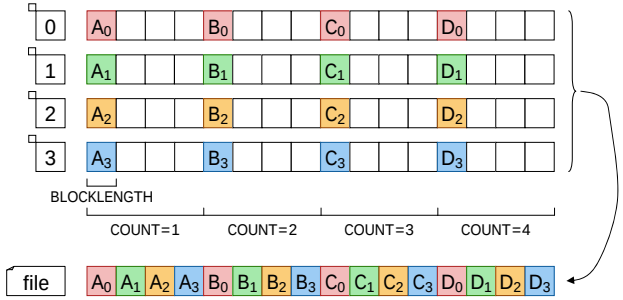
MPI-IOでのストライド転送

- ストライドデータの入出力
- Data Sieving

ストライドデータの入出力

FUJITSU

■ 非連続ベクトル型データから一つのファイルを作成。



ベクトル宣言型

```
size = MPI_Comm_size(MPI_COMM_WORLD, &size);

MPI_Type_contiguous(BLOCKLENGTH, MPI_INT, &type1);
MPI_Type_commit(&type1);

MPI_Type_vector(COUNT, BLOCKLENGTH, BLOCKLENGTH * size, MPI_INT, &type2);
MPI_Type_commit(&type2);
```


POSIX-IOとMPI-IOでの違い

FUJITSU

■ POSIX-IOでの書き込み

```
y = malloc(BLOCKLENGTH * size * COUNT * sizeof(int));
for(j = 0; j < COUNT; j++)
    MPI_Gather(x + size * j * BLOCKLENGTH, 1, type1, y + size * j * BLOCKLENGTH,
               1, type1, 0, MPI_COMM_WORLD);
if(rank == 0){
    fd = open(fname, O_CREAT | O_WRONLY, 0666);
    write(fd, y, BLOCKLENGTH * size * COUNT * sizeof(int));
    fsync(fd);
    close(fd);
}
free(y);
```

POSIX-IOは、IOマスター方式でなければ書き込めない

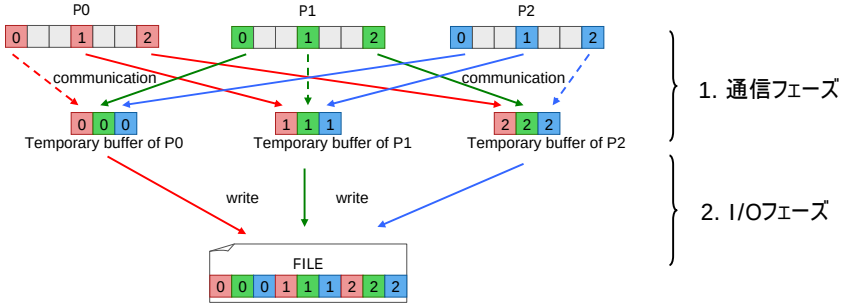
■ MPI-IOでの書き込み

```
MPI_File_open(MPI_COMM_WORLD, (char *)fname,
              MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
MPI_File_set_view(fh, BLOCKLENGTH * rank * sizeof(int), type1, type2,
                  "native", MPI_INFO_NULL);
MPI_File_write_all(fh, x, 1, type2, &status);
MPI_File_sync(fh);
MPI_File_close(&fh);
```

MPI-IOは、型を意識した入出力が可能

さらに、集団的MPI-IOを用いると、
Data Sievingを用いた効率的な入出力を行うことができる

Data Sievingとは

FUJITSU

- ROMIOの集団的MPI-IOに実装されたデータバッファリング
  - 不連続データの集団的MPI-IOで使用されます。
  - 一つ一つのブロックに対する小さいファイル要求でなく、バッファを使用した大きな領域でのファイル要求となるため、性能面で有利に働くことがあります。
  - 通信フェーズとI/Oフェーズの2つのフェーズがあります。writeの場合は、通信フェーズ→I/Oフェーズとなり、readの場合は、I/Oフェーズ→通信フェーズとなります。
  - Data Sievingを利用するかどうかとテンポラリバッファの大きさは、ヒントにより変更可能です。
    - romio_ds_read / romio_ds_write　　Data Sievingを利用するかどうか
    - ind_rd_buffer_size / ind_wr_buffer_size　　テンポラリバッファの大きさ