KOBE HPC スプリングスクール 2021（中級）

MPI-IO：並列ファイルシステム

2021年3月10日

神戸大学大学院システム情報学研究科計算科学専攻
横川三津夫

参考資料： 堀 敦史，MPI-IO https://www.r-ccs.riken.jp/wordpress/wp-content/uploads/kogi5_v3.pdf
志田直之，MPIの基本的通信と入出力 https://www.ssken.gr.jp/MAINSITE/download/wg_report/hpct/report_hpct_3.3.1.pdf

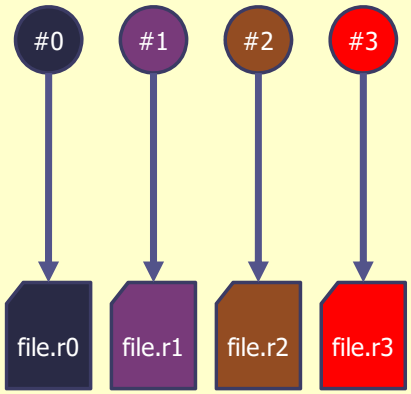
講義の内容

- MPI-IO
- 演習問題

【参考】並列ファイルシステム

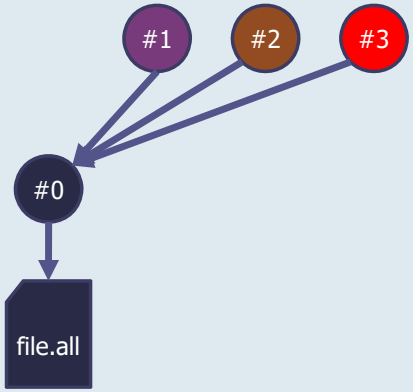
MPIプログラムにおけるファイル（入）出力

方式1：プロセスごとの個別ファイル



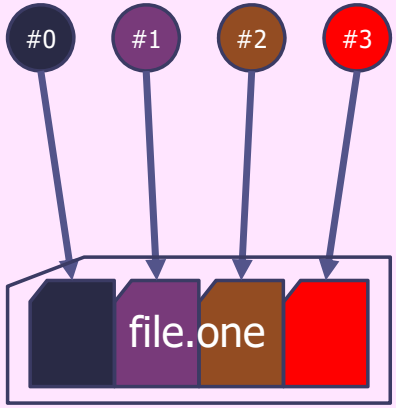
- MPIプロセス毎にファイルにランク番号など識別子を付けて出力
- 一般的なプログラムで記述できる.
- プロセス数の増加により，ファイル数が多くなる.

方式2：一つのファイル



- データを一つのプロセス #0 に集めて，一つのファイルに出力
- 事前にデータ転送が必要.
- プロセス #0 のメモリが不足するかも.
- プロセス #0 の負荷が高く，プロセス間の負荷バランスが悪い.

方式3：MPI-IOによる一つのファイル

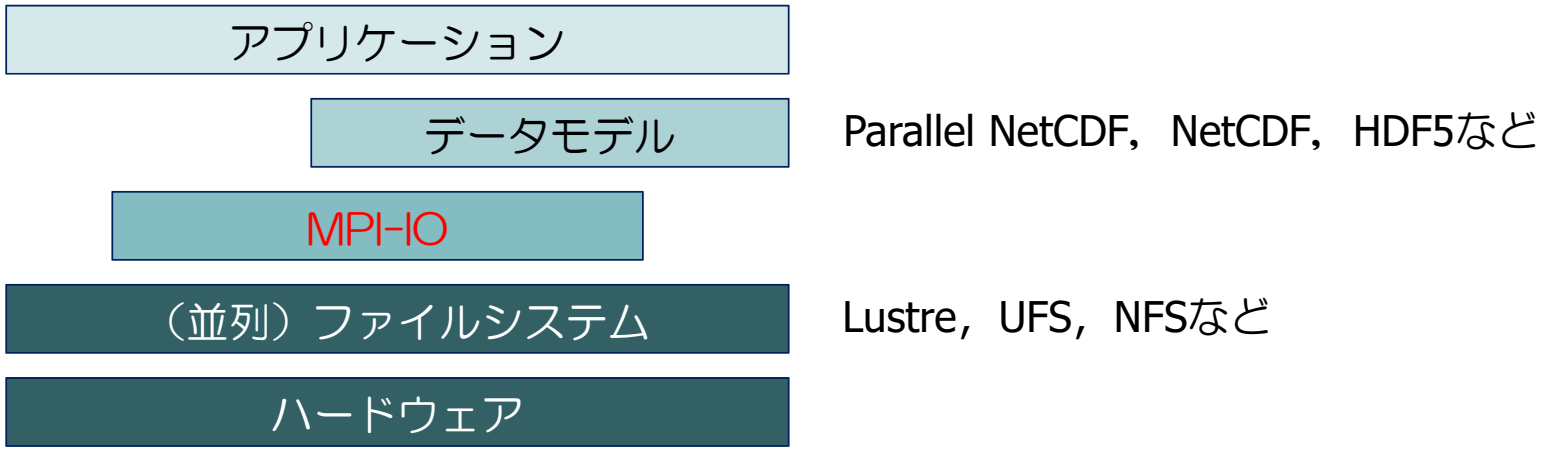


- MPI-IOを利用して一つのファイルに出力
- プロセス間の同期が必要ない.
- 多重プロセスからの書き込みが可能なファイルシステムが必要.

MPI-IO

- MPI-2規格に沿った並列I/Oを含むI/Oインターフェイス群
  - ROMIO：米国 Argonne National Laboratoryによって実装されたMPI-IO
    - MPI-IO実装のde facto standard

- IOアーキテクチャ



MPI-IO ファイルのオープン，クローズ

```
【C】int MPI_File_open(MPI_Comm comm, char *filename, int amode, MPI_Info info, MPI_File *fh)
【C】int MPI_File_close(MPI_File *fh)

【F】mpi_file_open( comm, filename, amode, info, fh, ierr )
【F】mpi_file_close(fh, ierr )
```

[Input]

- comm: コミュニケータ（集団操作のため，コミュニケータに属するプロセスで同じ関数を呼びだす必要あり）
- filename: ファイル名
- amode: ファイルアクセスモード
- info: MPI-IOに対するヒント
  - CではMPI_Info型， Fortranでは整数型
  - 情報を与えないときには，MPI_INFO_NULL とすればよい．
  - infoを使う場合には，MPI_Info_create 関数でinfo オブジェクトを作成する必要がある．

[Output]

- fh: ファイルハンドル（これを用いてファイル操作を行う）
- ierr: 戻りコード（整数型）

アクセスモード（amode）

以下の予約後のORでアクセスモードをセットする．

MPI_MODE_RDONLY	readのみ可能
MPI_MODE_RDWR	read/write可能
MPI_MODE_WRONLY	writeのみ可能
MPI_MODE_CREATE	ファイルがない場合，新規作成
MPI_MODE_EXCL	ファイルが存在してたらエラーを返す
MPI_MODE_DELETE_ON_CLOSE	ファイルクローズ時に消去
MPI_MODE_UNIQUE_OPEN	複数のプロセスで同時にファイルをオープンしない
MPI_MODE_SEQUENTIAL	複数のプロセスで逐次的にファイルをオープン
MPI_MODE_APPEND	すべてのファイルポインタをファイル終端にセット（追加）

例えば，ファイルを新規作成，writeのみ可能なファイルにしたかったら

```
MPI_MODE_CREATE|MPI_MODE_WRONLY
```

MPI-IO関数の分類

Positioning	Synchronism	Coordination	
		Non-collective	Collective（集団型）
Explicit offsets ファイルポインタを持たず，ファイル先頭からのオフセットで各プロセスの位置を指定	Blocking	MPI_File_read_at MPI_File_write_at（次ページ）	MPI_File_read_at_all MPI_File_write_at_all
	Nonblocking	MPI_File_iread_at MPI_File_iwrite_at	MPI_File_iread_at_all MPI_File_iwrite_at_all
	Split collective	N/A	MPI_File_iread_at_all_begin/end MPI_File_iwrite_at_all_begin/end
Individual file pointers 各プロセスが独立したファイルポインタを持つ	Blocking	MPI_File_read MPI_File_write	MPI_File_read_all MPI_File_write_all（次ページ）
	Nonblocking	MPI_File_iread MPI_File_iwrite	MPI_File_iread_all MPI_File_iwrite_all 注）MPI 3.1 の追加仕様．サポートされていない場合がある．
	Split collective	N/A	MPI_File_iread_all_begin/end MPI_File_iwrite_all_begin/end
Shared file pointer すべてのプロセスで共通のファイルポインタを持つ． 処理が遅いので使わない方が良い．	Blocking	MPI_File_read_shared MPI_File_write_shared	MPI_File_read_ordered MPI_File_write_ordered
	Nonblocking	MPI_File_iread_shared MPI_File_iwrite_shared	N/A
	Split collective	N/A	MPI_File_read_ordered_begin/end MPI_File_write_ordered_begin/end

MPI-IO 出力関数の例：MPI_File_write_all, MPI_File_write_at

```
【C】int MPI_File_write_all(MPI_File fh, void *buf, int count, MPI_Datatype datatype,
                            MPI_Status *status)
【F】mpi_file_write_all(fh, buf, count, datatype, status ierr )
```

- fh: ファイルハンドル
- buf: 出力する変数の先頭アドレス
- count: 出力する変数の個数
- datatype: 変数のデータ型
- status: 通信の状態を格納するオブジェクト.
- ierr: 戻りコード（整数型）

```
【C】int MPI_File_write_at(MPI_File fh, MPI_Offset offset, void *buf, int count,
                           MPI_Datatype datatype, MPI_Status *status)
【F】mpi_file_write_at(fh, offset, buf, count, datatype, status ierr )
```

- offset: ファイル先頭からのオフセット

※ 他の引数は，MPI_File_write_all と同じ

Blocking vs. Nonblocking

- Blocking I/O
  - I/O要求が完了するまで制御が戻らない．ただし，書き込みバッファを利用する記憶デバイスでは，記憶デバイスへの書き込みは保証しない．書き込み保証のためには，MPI_File_sync 関数で確認をすること．

```
【C】int MPI_File_Sync(MPI_File *fh)
```

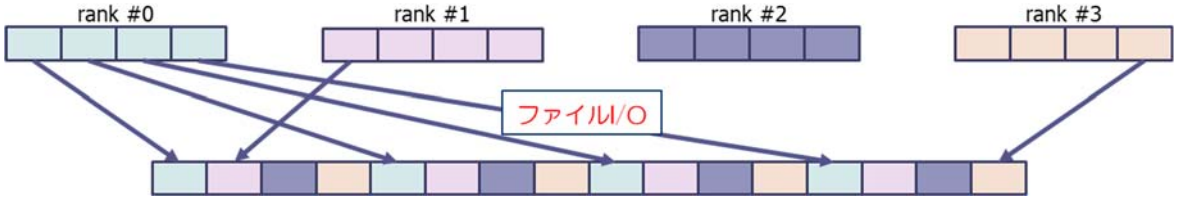
- Nonblocking I/O
  - I/O操作のルーチンを呼び出した後，完了まで待たない．計算処理とオーバラップさせることが出来る．動作が完了しているかどうか（ユーザバッファが再利用できるかどうか）は，MPI_Wait，MPI_Test で確認する必要がある．

MPI-IOのプログラム例（抜粋）： ファイルpfile.bin に出力

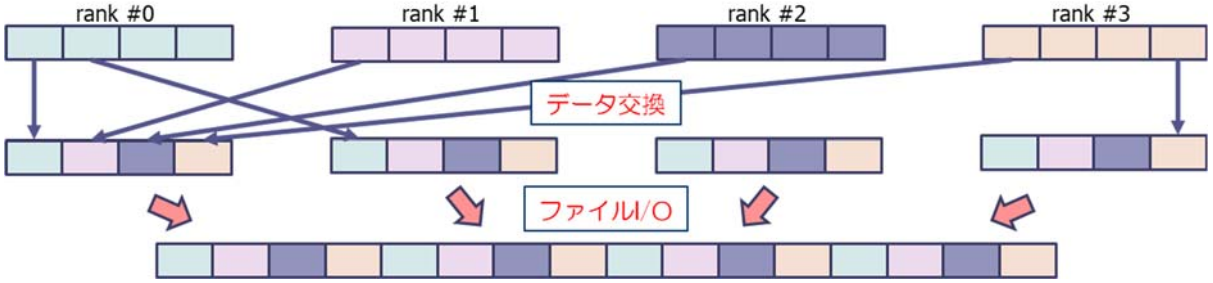
```
int stripe=16;
int var[16];

int nprocs, myrank, tag, disp;
MPI_Status status;

MPI_File   fh;                                          // ファイルハンドラ
char       *filename="./pfile.bin";                     // ファイル名

MPI_Init( &argc, &argv );
MPI_Comm_size( MPI_COMM_WORLD, &nprocs );
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

var <-                                                  // varに適当な値をセット

MPI_File_open( MPI_COMM_WORLD, filename, MPI_MODE_CREATE|MPI_MODE_RDWR, MPI_INFO_NULL, &fh );  //  ファイルオープン

disp = myrank*stripe*sizeof(int);                       // 各プロセスのオフセット（disp），バイトで指定
MPI_File_write_at( fh, disp, var, stripe, MPI_INT, &status );   // 各プロセス毎に出力

MPI_File_sync( fh );                                    // ファイル入出力の同期
MPI_File_close( &fh );                                  // ファイルのクローズ

MPI_Finalize();
```

Non-collective I/O vs. Collective I/O（集団的I/O）

- Non-collective I/O：各プロセスがファイルの位置に直接書き込む.



- Collective I/O
  - すべてのプロセスが同時にI/Oすることで，細かいデータを取りまとめることができ，結果としてI/Oのデータ量を大きくすることが出来る.
  - 派生データ型を用いて，不連続なデータ並びに対し，一括したI/O処理が可能
  - I/Oを行うデータをバッファ上で連続に並べることにより，高いI/O性能が期待できる.



MPI-IO ファイルビューの定義

```
【C】int MPI_File_set_view(MPI_File fh, MPI_Offset disp, MPI_Datatype etype, MPI_Datatype ftype,
                           char *datarep, MPI_Info info)
【F】mpi_file_set_view(fh, disp, etype, ftyep, datarep, info, ierr)
```

- fh: ファイルハンドル
- disp: オフセット値
- etype: ファイルビューを表すftypeを生成する元となるデータ型
- ftype: ファイルビューを表すデータ型
- datarep: 以下の3種類の内のいずれかを指定
  - native: メモリ内のバイナリデータ表現と同じ表現
  - internal: 同一システム内で互換するデータ表現
  - external32: 異なるシステム間でも互換性のあるデータ表現．最も一般的で可搬性のあるバイナリ表現
- info: ファイルシステムに与えるヒント情報（CではMPI_Info型， Fortranでは整数型）
  - 情報を与えないときには，MPI_INFO_NULL とすればよい．
- ierr: 戻りコード（整数型）

※ ファイルビュー生成は，MPI_File_open 関数によりファイルハンドルを取得後に行う．

MPI-IO ファイルの操作（削除，ファイルサイズ取得）

```
【C】int MPI_File_delete(char *filename, MPI_Info info)
【F】mpi_file_delete(filename, info, ierr)
```

- filename:　削除するファイル名
- info:　　MPI-IOに対するヒント（CではMPI_Info型，　Fortranでは整数型）
  - 情報を与えないときには，MPI_INFO_NULL とすればよい．
- ierr:　　戻りコード（整数型）

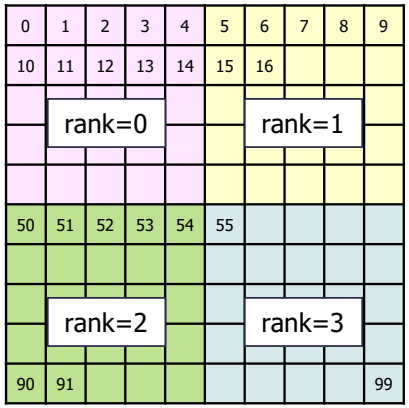
どのプロセスもファイルをオープンしてない場合にファイルを削除する．ファイルがオープンされていれば，エラーとなる．

```
【C】int MPI_File_get_size(MPI_File fh, MPI_Offset *size)
【F】mpi_file_get_size(fh, size, ierr)
```

- fh:　　ファイルハンドル
- size:　　ファイルサイズ（整数型），バイト単位で値が返る．
- ierr:　　戻りコード（整数型）

演習問題

- 4プロセスのMPIプログラムにおいて，それぞれのプロセスが下図のような部分配列を持っているとする時，各プロセスが持つデータを図のファイル上の並び（図の番号順）になるように出力せよ．



ファイル上の並び

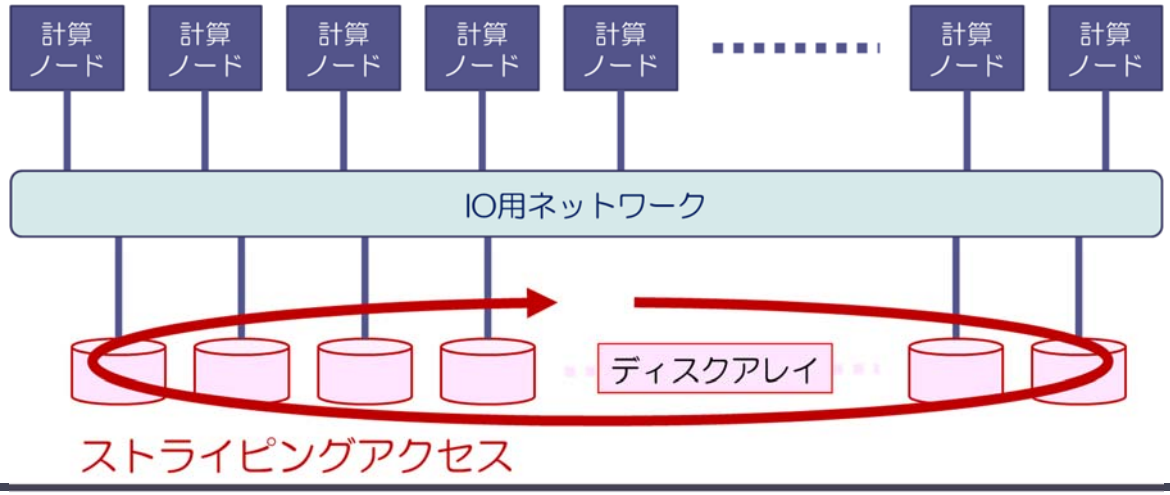
- 各プロセスの持つ部分配列は，n × n とする．したがって，配列全体の大きさは 2n × 2n.
- 配列のデータ型は何でも良い．例えば int型.

【ヒント】

1. ファイルビューを作るためのMPI_Datatype を設定
2. MPI_File_open でMPI-IOファイルをオープン
3. MPI_File_set_view で出力するファイルビューを作成
4. MPI_File_write_all により各プロセスが持つデータを出力

【参考】並列ファイルシステム

- 複数のストレージデバイスを仮想的に１ボリューム化し，共有ファイルシステムとしてサービスするファイルシステム
  - ネットワークを介して複数のストレージデバイスにストライピングアクセス
  - 分散格納されたデータを一つのファイルとして扱うことが可能
  - 高いI/O性能
- 代表的並列ファイルシステム：Lustre，GPGS，FEFSなど



例：Lustreファイルシステム

- 構成
  - １台のメタデータサーバ（Meta Data Server：MDS）
    - メタデータ領域（Meta Data Target：MDT）で，ファイルシステム全体の内容を管理
  - 複数のオブジェクトストレージサーバ（Object Storage Server：OSS）
    - 一つ以上のデータ領域（Object Storage Target：OST）から成り，全体を仮想的に１ボリュームとして見せる.
- 動作
  - 一つのファイルをストライプサイズ（デフォルト 4 KB）に分割し，OSTに分散配置する.
- MDSにアクセスが集中すると性能が低下する.