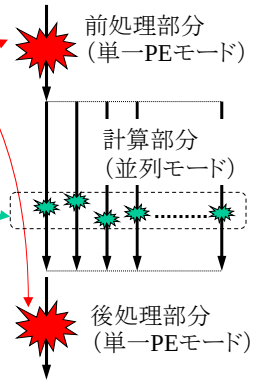
データ工学特論（7）

情報基盤センター
天野　浩文

1

おさらい（1）

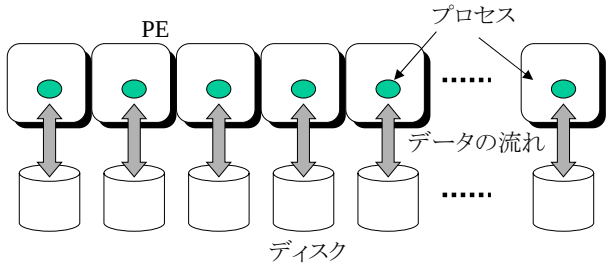
- 並列プログラムの入出力
  - 単一のPEが代表で入出力を行う
    ⇒大容量のデータの入出力には時間がかかる
  - 複数のPEが同時に入出力を行う
    ⇒プログラマが制御するのは面倒



2

おさらい（2）

- 並列入出力のイメージ
  - 複数のPE上で動作している各プロセスが，自分の必要とするデータを並列に入出力
  - あたかも，それぞれがローカルディスクにアクセスしているかのように（物理的にはそうでなくてもよい）



3

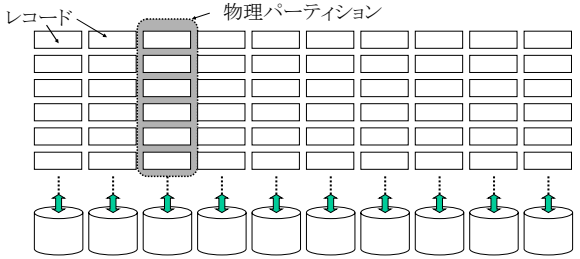
おさらい（3）

- **並列ファイルシステム**
  （どちらかというと）複数のディスクをまとめて1つのファイルシステムのイメージを提供
  - Vestaファイルシステム
  - Acaciaファイルシステム
- **並列入出力ライブラリ**
  （どちらかというと）すでにあるファイルシステムを並列に使うイメージを提供
  - Bordawekar らの並列入出力プリミティブ
  - Galbreath らの応用指向並列入出力パッケージ
- （既存のライブラリの）**並列入出力インタフェース**
  - MPIの集団型入出力（collective I/O）インタフェース

4

おさらい（4） Vesta

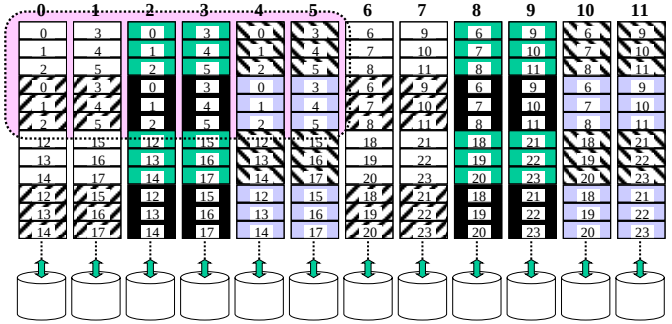
- Vestaファイルの基本構造…2次元
  ファイル⇒**物理パーティション**の集まり
  物理パーティション⇒レコードの線形並び
- 物理パーティション数とレコードサイズはファイル作成時に決定
  …その後は変化しない



5

おさらい（5） Vesta

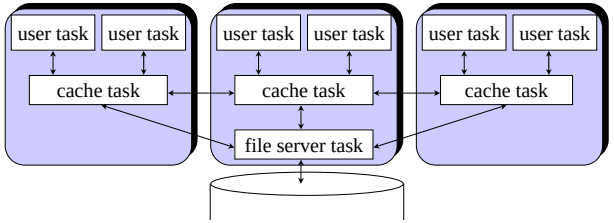
- 物理パーティション（下の例では12本）がカバーされるまでタイルパターンを繰り返す．レコード数が増大すると，これを下方向に繰り返す．



6

おさらい（6） Acacia

- **キャッシュタスク**
  - 各PEでデータをキャッシュ
  - ユーザタスク，ファイルサーバタスクとの通信
- **ファイルサーバタスク**
  - ディスクのあるPEにのみ配置
  - ディスクへのアクセスを担当



7

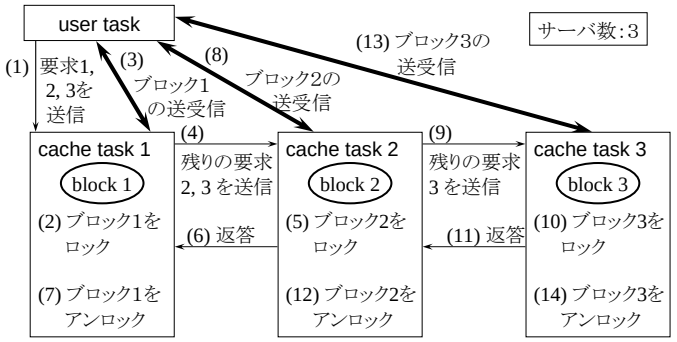
おさらい（7） Acacia

- **ファイルアクセスモード**
  - 複数のPEが同一のファイルへのアクセスするときのアクセスのしかたを制御
- 複数のモードをサポート
  - **独立モード**
  - **複製モード**
  - **共有モード**
    - 無指定
    - 固定順，固定長
    - 任意順，固定長
    - 任意順，可変長

8

おさらい（8） Acacia

- 分散ファイルキャッシング
  - キャッシュタスク間のロック委託による2ブロック逐次ロック



9

さて，今回は…

- **並列ファイルシステム**
  （どちらかというと）複数のディスクをまとめて1つのファイルシステムのイメージを提供
  - Vestaファイルシステム
  - Acaciaファイルシステム
- **並列入出力ライブラリ**
  （どちらかというと，すでにあるファイルシステムを並列に使うイメージを提供）
  - Bordawekar らの並列入出力プリミティブ
  - Galbreath らの応用指向並列入出力パッケージ
- （既存のライブラリの）**並列入出力インタフェース**
  - MPI-2の並列入出力インタフェース

10

Bordawekar らの並列入出力プリミティブ

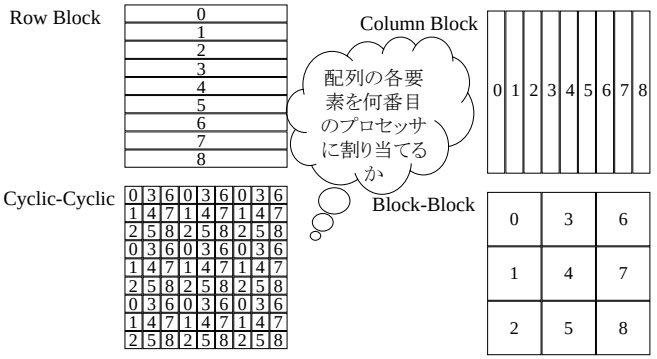
- Bordawekar, del Rosario, Choudhary (Univ. Syracuse, 1993)
  Intel Touchstone Delta 用にさまざまなデータの分散法をサポートする並列入出力プリミティブを実装
  - **二段階アクセス戦略**（Two-Phase Access Strategy）
    配列データのプロセッサへのマッピングとディスクへのマッピングは別のものとして扱う
  - SPMD（Single Program, Multiple Data）方式の並列プログラムに，逐次ファイルと同様のインタフェースを提供
    - **`popen, pclose`**
    - **`pread, pwrite`**
    - **`proc_map, array_map`**

  （SPMD: 同じプログラムのコピーをMIMD計算機の各プロセッサ上で同時に実行する形態のこと）

11

巨大配列のプロセッサへのマッピング

- Vienna Fortran, Fortran D, High Performance Fortran などが許すプロセッサへのマッピングの例



12

配列データのディスクへのマッピング（1）

●マッピングの例：列優先順序（Column-Major Order）

配列

0	3	6	9	12	15
1	4	7	10	13	16
2	5	8	11	14	17

ディスクブロック
0
1
2
3
4
5
6
7

13

配列データのディスクへのマッピング（2）

●マッピングの例：行優先順序（Row-Major Order）

配列

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14
15	16	17
18	19	20

ディスクブロック
0
1
2
3
4
5
6
7

14

直接アクセス方式

●並列計算機付属の入出力ライブラリでサポートされる低水準入出力関数
- ◆PE番号とメモリ上のアドレス
- ◆ディスク上のブロックアドレス

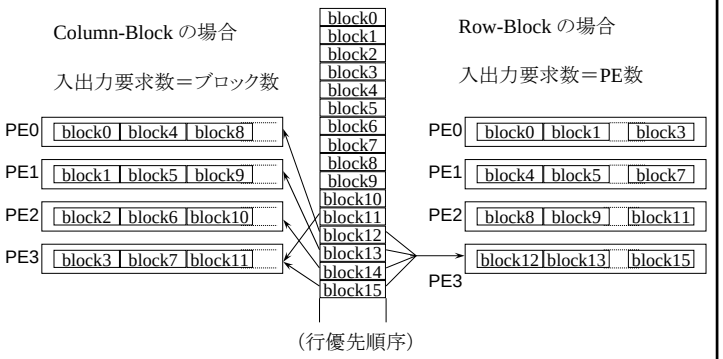
●個々の入出力関数は，上記の2つを指定して呼び出す

●この方式では，配列データのプロセッサへのマッピングとディスクへのマッピングが合わない場合に困ったことになる

15

2つのマッピングの干渉

●配列データのプロセッサへのマッピングとディスクへのマッピングが合わないとどうなるか



16

マッピングの種類と入出力要求の総数

●$N \times N$配列を Column-Major で格納するときの入出力要求数

$$R_{trans} = R_{dist} \times \frac{S_{dist}}{\min(S_{dist}, S_{stripe})}$$

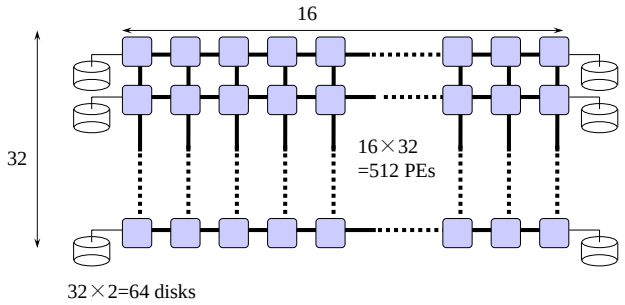
R_{dist}: ストライプサイズを無視したときの入出力要求数
S_{dist}: 連続してアクセスできる最大ブロック数
S_{stripe}: ストライプサイズ

分散法	R_{dist}	S_{dist}
Block-Block	$N*\sqrt{P}$	$N/\sqrt{P}$
Block-Cyclic	$N*\sqrt{P}$	$N/\sqrt{P}$
Cyclic-Block	N^2	1
Cyclic-Cyclic	N^2	1

17

直接アクセス方式の性能（1）

●実験に使用したシステム… Intel Touchstone Delta
- Intel 80860 (i860)プロセッサ採用
- 後の Intel Paragon のプロトタイプ



18

直接アクセス方式の性能

●Column-Block
- Column-Major（列優先）との相性がよい

●Column-Major に対する性能の比較
Column-Block＞Column-Cyclic＞Row-Block＞Row-Cyclic
- 最良のものと最悪のものでは，2桁以上の差がある

19

並列入出力プリミティブの設計

●直接アクセス方式の問題点
- マッピングの違いにより，性能に極端な差が出る

●2段階アクセス方式（読み出しの場合の例）
- ディスク上のデータ分散に最も適したマッピングでひとまずメモリに読み込む
- ユーザの指定するプロセッサへのマッピングが食い違う場合には，プロセッサ間でデータを再分配する

●さまざまなマッピングを許すことができるが，性能はそれほど下がらない

●**`popen, pclose, pread, pwrite`**
`proc_map, array_map`

20

並列ファイルのオープン・クローズ

●オープン

```
popen(ユニットナンバ,        …オープンする並列ファイルのID
      ファイル名,
      アクセスモード,        …sequential か direct か
      フォーマット,          …書式付きか, 書式なしか
      モード,                …新規, 既存, etc.
      ディスク数,            …-1なら全部
      I/Oを行う PE 数 )
```

●クローズ

```
pclose(ユニットナンバ)
```

21

配列のプロセッサへのマッピング

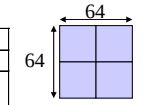
```
array_map(データを格納する配列,
          サイズ情報配列,          …各次元ごとに記述
          分散情報配列,            …  〃
          ブロックサイズ配列,      …  〃
          プロセッサ数配列 )       …  〃
```

戻り値…アレイディスクリプタ (integer). アレイディスクリプタテーブルのID.

●例:64×64配列を Block-Block で4プロセッサに分配するときのアレイディスクリプタテーブル(Array Descriptor Table, ADT)

	1	2	3	4	5	6	7
Global Size	64	64	-1	-1	-1	-1	-1
Distr. Code	1	1	-1	-1	-1	-1	-1
Block Size	-1	-1	-1	-1	-1	-1	-1
nprocs	2	2	-1	-1	-1	-1	-1

← Block Sizeは Cyclicのときのみ使用する



(-1は don't care)

22

実プロセッサと論理プロセッサのマッピング

●プロセッサ集合を
　物理レベル
　論理レベル
に分けて定義できる

```
proc_map(次元ごとのプロセッサ数の配列,   … ADT と同じもの
         パラメータ,                     …  0 なら変更なし
         ユーザ定義マップ情報配列 )
```

23

Read/Write

●並列読み出し

```
pread(データ配列名,
      読み込むデータのサイズ, . . .
      アレイディスクリプタ,
      ユニットナンバ,
      バッファ配列名,
      バッファサイズ )
```

●並列書き込み… **pwrite**も**pread**と同様

●配列の実プロセッサへのマップとディスクとのマップの相性が悪いときには, 自動的に中間的なマッピングを用いて入出力を行う.

24

2段階アクセス方式の性能

●Column-Majorで格納された10K*10K配列を64PEに分配する場合

	最適 read	再配分	計	直接アクセス	速度向上
Column-Block	11395	-	11395	11395	1
Column-Cyclic	11395	2478	13873	63400	4.57
Row-Block	11395	1028	11623	78767	6.87
Row-Cyclic	11395	3092	14487	*	>248

（単位は msec.　*は時間のオーダになってしまったもの）

◆Column-Blockに読み込むときが最も効率がよい（11395ms）.

◆Column-Cyclic, Row-Block, Row-Cyclic は，Column-Block でアクセスしてから再配分する時間を合わせても，直接アクセス方式よりも数倍～数百倍効率がよい.

25

Galbreath らの応用指向並列入出力パッケージ

●Galbreath, Gropp, Levine (Argonne Nat. Lab., 1993) PETSc/Chameleon パッケージ
◆応用レベルの要求に適合した高水準の並列入出力(PIO)パッケージ

●実際の科学技術計算で必要となる入出力を分析
…使用されているプログラムコードではなく，要求記述を分析
◆入力（初期値）
◆デバッグ用
◆作業ファイル (scratch file)
◆チェックポイント／リスタート
◆出力（計算結果）

26

科学技術計算プログラムにおける入出力

●入力
◆大域的パラメータ…各PEに複製が置かれる
◆各格子点の初期状態…各PEに分割して置かれる

●デバッグ用
途中の状態を記録して，逐次版の結果と比較
…通常の逐次ファイルと同じ形式で書き込まれる必要がある.

●作業ファイル
再計算を省略したり，メモリに入りきれないデータを格納する.

●チェックポイント／リスタート
各PEの状態を書き込み，あとで元通りに読み込むことが必要.

●出力
いくつかのパラメータだけ出力するものから，各PEの値をすべて出力するものまで，さまざま.

27

PETSc/Chameleonパッケージの設計方針

●SPMDプログラムで使用するための高水準入出力関数ライブラリ
◆共通のスカラー値（あるいは小配列）を読み書き
◆分散配列を各プロセスが一斉に読み書き
◆特定のプロセスがまとまったデータを読み書き

●従来型のストリーム入出力とのアナロジを残す
◆`fopen, fclose, fread, fwrite, ...`

●抽象化のレベルを高く
◆分散配列の物理的な格納方法は意識しない
◆異なるアーキテクチャへの移植や性能の向上は関数の実装レベルで吸収，応用プログラムには影響を与えないように

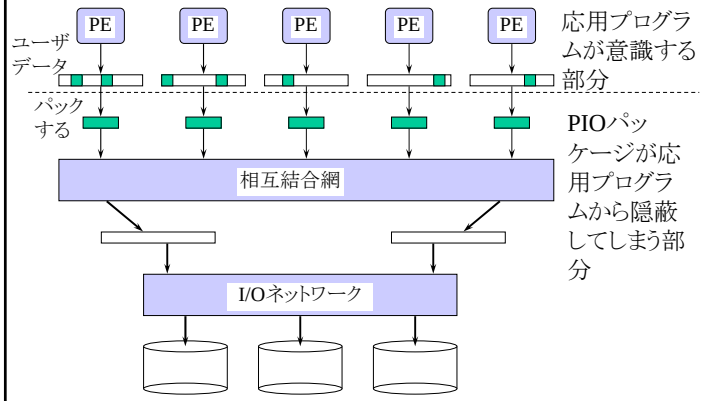
28

主な入出力関数

- **PIFopen, PIFclose**
  - 並列ファイルのオープン・クローズ
  - 並列ファイルポインタ(PIFile構造体へのポインタ)を返す
- **PIFReadCommon, PIFWriteCommon**
  - 各PEに複製されるデータの読み書き
- **PIFReadDistributedArray, PIFWriteDistributedArray**
  - 各PEに分割して置かれる配列データを**単一ファイルから**読み書き
- **PIFRead, PIFWrite**
  - 各PEが大量データを読み書きするとき

29

システムのイメージ



30

並列ファイルモードと逐次ファイルモード

- PIFopen のパラメータを1個書き換えるだけで，並列ファイルを通常の UNIX ファイルと互換の逐次ファイルに書き出せる …デバッグに便利

- ワークステーションクラスタにおける性能比較 (ファイルサイズ:64KB)

ファイル形式	PE 数	Read MB/sec	Write MB/sec
並列	2	6.8	3.7
逐次	2	0.8	0.1

31

実機での性能

- IBM SP-1 (ファイルサイズ:16MB)
  - 各PEにディスクがついている
  - PE数とブロックサイズを変化させる

PE 数	分割のサイズ	性能が最も低下するブロックサイズ	最小の Read 性能 MB/sec	最小の Write 性能 MB/sec
4	4MB	8KB	6	26
8	2MB	64KB	12	40
16	1MB	16KB	26	30

32

MPI

- 並列計算機の問題点
  - それぞれのメーカ・開発者が独自の構想で設計・実装
    - 固有のアーキテクチャ
    - 固有のプログラミング言語
    - 固有のプログラミング環境
  - 次の並列機が登場するたびに，プログラムの書き換えが必要
- **MPI** (Message Passing Interface)
  - プラットフォームが変わっても，同じプログラムを動作させることはできないか？　⇒標準化への動き
    - 仮想的な分散メモリ型並列計算機の上のプロセッサ間通信ライブラリを提供
    - プログラミング言語は既存のものを活用
    - 並列計算機ごとにライブラリのほうを実装あるいは変更

33

MPIの概要（1）

- （PVMと並ぶ）並列プログラミングライブラリの国際標準
- CまたはFortranから呼び出すことのできる関数群
  - 並列プロセスの初期化：**`MPI_Init()`**
  - 自分のプロセスIDの取得：**`MPI_Comm_rank()`**
  - プロセスの総数の取得：**`MPI_Comm_size()`**
  - 宛先のIDを指定してメッセージを送信：**`MPI_Send()`**
  - 送信もとを指定して受信：**`MPI_Recv()`**
  - 並列処理の終了：**`MPI_Finalize()`**

  などなど．
- SPMD (single-program, multiple-data)
  - 同一のプログラムのコピーを複数のプロセッサで走らせる実行形態

34

MPIの概要（2）

- プログラマはMPIの関数を呼び出すようなプログラムを1つ書く．実行時にプログラムのコピーを配布して実行．

```
#include "mpi.h"
...
void main(int argc, char* argv[]) {
  ... /* 通常の逐次処理を行う部分 */
  MPI_Init(&argc, &argv);

    ...
    /* 各プロセスで並列に行う処理 */
    ...

  MPI_Finalize();
  ... /* 通常の逐次処理を行う部分 */
} /* main の終わり */
```

35

MPIの概要（3）

- プロセスによって処理の内容を変えたいときは…
  - プロセスIDによる条件分岐で対処

```
...
MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
...
if (my_rank==0) {

  /* プロセス0の処理内容 */

} else {

  /* 他のプロセスの処理内容 */

}
...
```

36

MPIの概要（4）

●メッセージの送受の際に，宛先（送信元），メッセージのサイズ・種別，同時に集団型（collective）通信に参加するプロセス集合などを指定する．

```
int MPI_Recv(
     void*          message   /* out */,
     int            count     /* in */,
     MPI_Datatype   datatype  /* in */,
     int            source    /* in */,
     int            tag       /* in */,
     MPI_Comm       comm      /* in */,
     MPI_Status     status    /* out */
)
```

メッセージ
サイズ
送信元
種別
プロセス集合
（ステータス）

●派生データ型（derived data type）のデータを含むメッセージも扱える

37

MPIプログラムにおけるファイルI/Oの問題点

●UNIXのファイル入出力インタフェース **read()**, **write()**を呼び出すことはできる．しかし．．．
- ◆もともとは非並列プログラムでのI/Oを想定
- ◆並列プロセスが同一ファイルに同時アクセスするための機能を持っていない
  - 同一ファイルの異なる部分にアクセスする機能がない
  - 不連続なデータを単一の操作でアクセスする機能がない
- ◆多数のファイルアクセス要求をまとめて効率的に処理する機能を持っていない
- ◆非同期アクセスの機能を持っていない

38

MPI-2の並列入出力インタフェース（1）

●当初MPI-IOとして設計→MPI規格2.0で取り入れ
●ファイル入出力を，メッセージ通信と見なす
- ◆read ⇒ receive
- ◆write ⇒ send

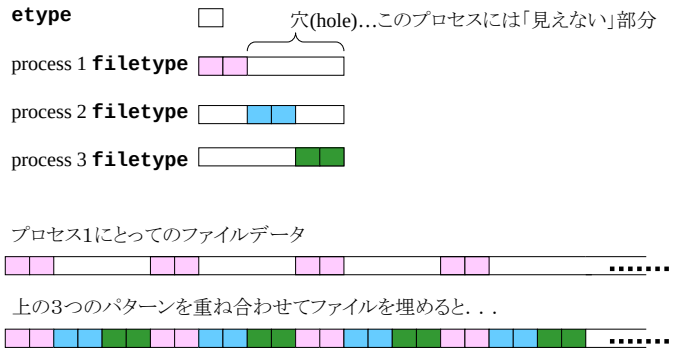
●各プロセスが入出力要求を出す
●新たに3つの引数を追加…いずれもMPIの派生データ型
- ◆**filetype**　ファイル上のデータ配置パターン
- ◆**buftype**　ユーザバッファ上のデータ配置パターン
- ◆**etype**　要素型 (elementary datatype)

●**filetype** と **etype** はファイルをオープンするときに指定する
●**buftype** は比較的自由に設定可能
●ファイル上のデータ配置は，**filetype**のパターンの繰り返し

39

MPI-2の並列入出力インタフェース（2）

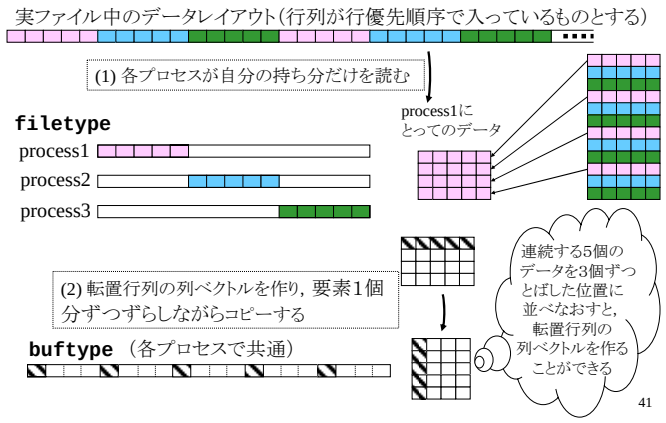
●プロセス間のデータ分配の例



40

MPI-2の並列入出力インタフェース（3）

●行列データを読み出してメモリ上で転置する例



41

集団型入出力（Collective I/O）

●MPIプログラムは，SPMD動作をする
- ◆各プロセスからの入出力関数の呼び出しは，一致とは言わないまでも，比較的まとまっている
- ◆したがって，これらの関数呼び出しを，それぞれバラバラのものと扱うこともできるが，各プロセスがいっせいに呼び出すものとして扱うこともできる．

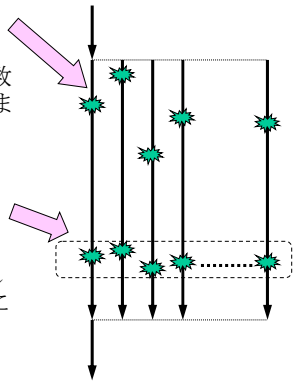
●このため，MPIの並列入出力関数は，同じ操作に対して，独立型と集団型の2種類のインタフェースが用意されている
- ◆独立型は，バラバラに動作する（実行の終了したプロセスは先に進んでよい）
- ◆集団型は，同期して動作する（他のプロセスでの実行が終了するまで，全プロセスが待ち合わせに入る）

42

集団型入出力の重要性

●独立型では，各プロセスからの入出力要求が独立の入出力として扱われる
- ◆各プロセスからの要求を少数の入出力システムコールにまとめることが困難
- ◆効率の低下

●集団型では，各プロセスからの入出力要求が同期して実行される
- ◆各プロセスがバラバラに発した入出力要求をひとまとめにすることができる
- ◆効率化の可能性



43

位置決め	同期	プロセス間協調	
		独立型	集団型
Explict offset	Sync	MPI_File_read_at MPI_File_write_at	MPI_File_read_at_all MPI_File_write_at_all
	Async	MPI_File_iread_at MPI_File_iwrite_at	MPI_File_read_at_all_begin MPI_File_read_at_all_end MPI_File_write_at_all_begin MPI_File_write_at_all_end
Individual offset	Sync	MPI_File_read MPI_File_write	MPI_File_read_all MPI_File_write_all
	Async	MPI_File_iread MPI_File_iwrite	MPI_File_read_all_begin MPI_File_read_all_end MPI_File_write_all_begin MPI_File_write_all_end
Shared offset	Sync	MPI_File_read_shared MPI_File_write_shared	MPI_File_read_ordered MPI_File_write_ordered
	Async	MPI_File_iread_shared MPI_File_iwrite_shared	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，ノンブロッキングととらえたほうがわかりやすいかも

44

まとめ

- **並列入出力ライブラリ**
  （どちらかというと，すでにあるファイルシステムを並列に使うイメージを提供）
  - Bordawekar らの並列入出力プリミティブ
    - 二段階アクセス戦略
    - 行優先順序と列優先順序
  - Galbreath らの応用指向並列入出力パッケージ
    - 物理的なデータ構成をユーザから隠蔽
    - 逐次型のストリーム入出力インタフェースを並列版に拡張
- （既存のライブラリの）**並列入出力インタフェース**
  - MPI-2の並列入出力インタフェース
    - 単一ファイル上のどの部分にアクセスするかを，プロセスごとに filetype によって指定

45