



計算科学演習 MPI 発展

学術情報メディアセンター
情報学研究科・システム科学専攻
中島 浩

© 2009 H. Nakashima



目次

- 序論
 - 1次元分割 vs 2次元分割、実装のポイント
- プロセスの2次元配置
 - 直交プロセス空間
 - rank ⇔ 座標の変換
- 2次元配列の宣言・割付・参照
- 不連続データの通信
 - 派生データ型、東西通信用派生データ型
 - 通信回数削減との関係
- 2次元分散データの出力
 - 良くない方法2種
 - 個別出力:MPI File I/O 機能の利用

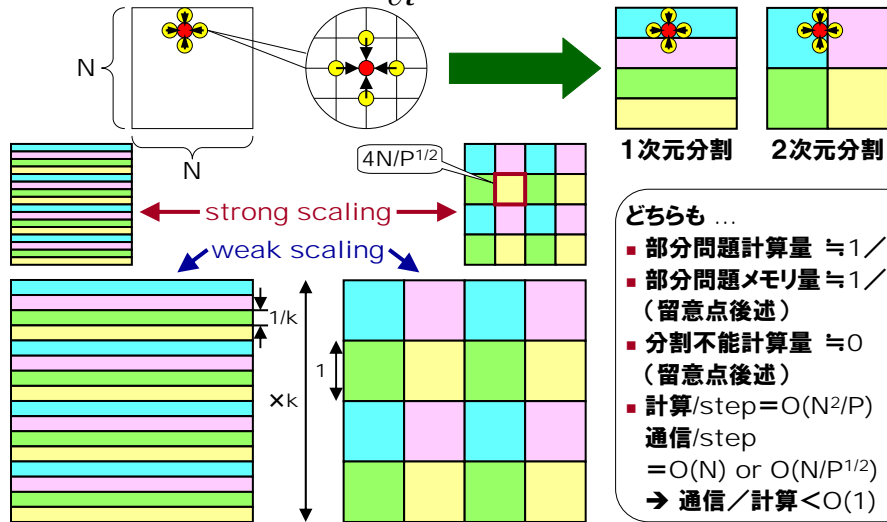
© 2009 H. Nakashima



序論

1次元分割 vs 2次元分割

■ 拡散方程式 $\nabla^2 \phi = \frac{\partial \phi}{\partial t}$ の初期値問題求解 by 陽解法

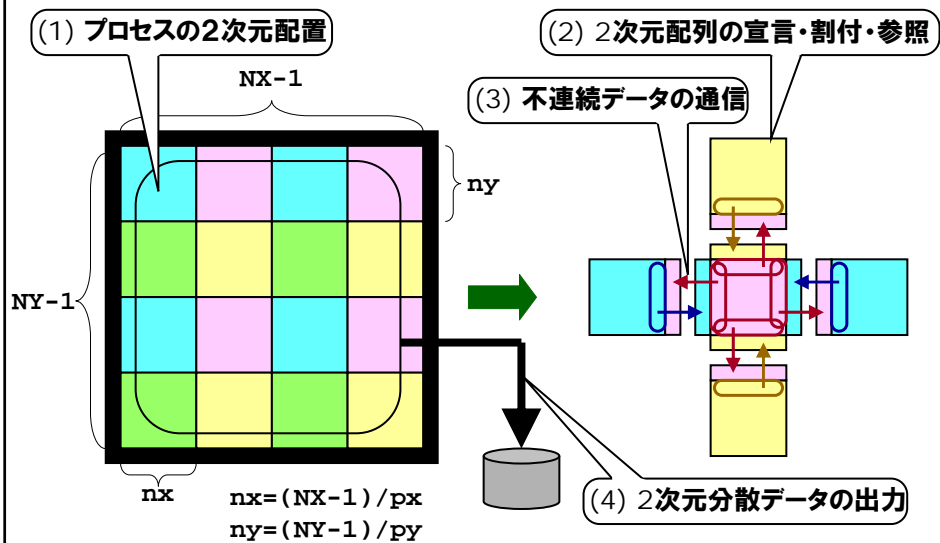


© 2009 H. Nakashima



序論

実装のポイント



© 2009 H. Nakashima

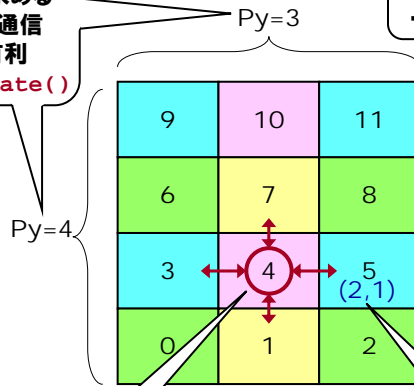


プロセスの2次元配置 概論

■ プロセス数 P (e.g. 12) について...

(1) $P = P_x \cdot P_y$, $P_x \leq P_y$
なる P_x, P_y を求める
← 南北通信 > 東西通信
にするのがやや有利
→ `MPI_Dims_create()`

(2) $P_x \cdot P_y$ の配置に対する
communicator を得る
→ `MPI_Cart_create()`



(3) 自分の東西・南北の rank を得る
→ `MPI_Cart_shift()`

(4) 自分のプロセス座標を得る
→ `MPI_Cart_coord()`

© 2009 H. Nakashima



プロセスの2次元配置 直交プロセス空間 (1/2)

■ $n = n_0 \times n_1 \times \dots \times n_{k-1}$

($n_0 \geq n_1 \geq \dots \geq n_{k-1}$ かつ $\sum n_i$ 最小) の算出

■ `MPI_Dims_create(int n, int k, int *dims)`

`dims={0,...,0}` or `dims=(/0,...,0/)` で呼出し

→ `dims[0...k-1]=dims(1...k)= n_0, n_1, \dots, n_{k-1}`

cf) 呼出し前の非ゼロ要素は「尊重」される

■ 例

```
int dims[2]={0,0};
MPI_Comm_size(MCW,&np);
MPI_Dims_create(np,2,dims);
```

```
integer::dims(2)=(/0,0/)
call MPI_Comm_size(MCW,np,err)
call MPI_Dims_create(np,2,dims,err)
```

© 2009 H. Nakashima



プロセスの2次元配置 直交プロセス空間 (2/2)

- プロセス座標系(p_0, \dots, p_{k-1}) ($0 \leq p_i < n_i$) 生成
 - `MPI_Cart_create(MPI_Comm comm, int k, int *dims, int *periods, int reorder, MPI_Comm *cart)`
 - `dims = n0, n1, ..., nk-1`
 - `periods[i] = periods[i+1] =`

$$\begin{cases} 0 & \text{次元 } i \text{ は非周期境界} \\ 1 & \text{次元 } i \text{ は周期境界} \end{cases}$$
 - `reorder == 0` ? rank不変 : rank変更あり

- 例

```
int periods[2]={0,0}; MPI_Comm cart;
MPI_Dims_create(np,2,dims);
MPI_Cart_create(MCW,2,dims,periods,0,&cart);
```

```
integer::periods(2)=(/0,0/),cart
call MPI_Dims_create(np,2,dims,err)
call MPI_Cart_create(MCW,2,dims,periods,0,card,err)
```

© 2009 H. Nakashima



プロセスの2次元配置 rank ⇔ 座標の変換 (1/3)

- (p_0, \dots, p_{k-1}) の rank (row major)
 - $r_0 = p_0$ $r_i = n_i r_{i-1} + p_i$ $r_{k-1} = \text{rank}(p_0, \dots, p_{k-1})$
 - `MPI_Cart_rank(MPI_Comm cart, int *coord, int *rank)`

`coord = p0, ..., pk-1` → `rank = rank(p0, ..., pk-1)`

- 例

```
int c[2];
for(c[0]=0;c[0]<dims[0];c[0]++){
  for(c[1]=0;c[1]<dims[1];c[1]++){
    MPI_Cart_rank(cart,c,r); MPI_Recv(...,r,...);
  }
}
```

```
integer::c(2)
do c(1)=0,dims(1); do c(2)=0,dims(2)
  call MPI_Cart_rank(cart,c,r,err)
  call MPI_Recv_rank(...,r,...)
end do; end do
```

© 2009 H. Nakashima



プロセスの2次元配置 rank ⇔ 座標の変換 (2/3)

- 逆変換 $(p_0, \dots, p_{k-1}) = \text{rank}^{-1}(r)$
 - `MPI_Cart_coords(MPI_Comm cart, int rank, int k, int *coord)`

- 例

```
int c[2];
MPI_Cart_coords(cart,me,2,c);
for(i=...) for(j=...) {
    y=c[0]*ny+i+1; x=c[1]*nx+j+1;
    ...
}
```

```
integer::c(2)
call MPI_Cart_coords(cart,me,2,c,err)
do i=...; do j=...;
    y=c(1)*ny+i+1; x=c(2)*nx+j+1
    ...
end do; end do
```

© 2009 H. Nakashima



プロセスの2次元配置 rank ⇔ 座標の変換 (3/3)

- 隣接プロセスの rank
 - `MPI_Cart_shift(MPI_Comm cart, int d, int dir, int *src, int *dst)`
 - $me = \text{rank}(p_0, \dots, p_{k-1}) \rightarrow \text{src} = \text{rank}(p_0, \dots, p_{d \mp 1}, \dots, p_{k-1})$
 - $\text{dst} = \text{rank}(p_0, \dots, p_{d \pm 1}, \dots, p_{k-1})$

非周期境界の端では
MPI_PROC_NULL

dir>0 → -/+
dir<0 → +/-

- 例

```
MPI_Cart_shift(cart,0,1,&south,&north);
MPI_Cart_shift(cart,1,1,&west,&east); ...
MPI_Sendrecv(...,north,...,south,...); ...
MPI_Sendrecv(...,east,...,west,...); ...
```

```
call MPI_Cart_shift(cart,0,1,south,north,err)
call MPI_Cart_shift(cart,1,1,west,east,err); ...
call MPI_Sendrecv(...,north,...,south,...); ...
call MPI_Sendrecv(...,east,...,west,...); ...
```

© 2009 H. Nakashima



2次元配列の宣言・割付・参照 (1/2)

- $[X0, x1] \times [Y0, y1]$ なる配列 a の宣言 & 割付
($X0, Y0$ は定数、 $x1, y1$ は可変)

- Fortran は相変わらず簡単

```
double precision, allocatable :: a(:, :)  
allocate(a(x0:x1, y0:y1))
```

- C で $a[j][i]$ に固執 → 良くない方法しかない

```
double **a; int h=y1-Y0+1, w=x1-X0+1;  
a=(double**)malloc(h*sizeof(double))-Y0;  
a[Y0]=(double*)malloc(h*w*sizeof(double))-X0;  
for(j=Y0+1; j<=y1; j++) a[j]=a[j-1]+w;
```

© 2009 H. Nakashima



2次元配列の宣言・割付・参照 (2/2)

- C での良い方法 = 2次元配列の1次元化

```
double *a;  
a=(double*)malloc(h*w*sizeof(double))-Y0*w-X0;  
for(i=...) for(j=...)  
...=...a[i*w+j]...a[(i+1)*w+j]...;
```

→ 内側ループのループ不変式 (e.g. $i*w$) の追い出し

```
for(i=...) {  
double *ai=a+i*w, *aip=ai+(i+1)*w;  
for(j=...)  
...=...ai[j]...aip[j]...;  
}
```

→ 演算子強度低減 (strength reduction) 乗算 → 加算

```
for(i=..., ai=..., aip=...; ...; ..., ai+=w, aip+=w) {  
for(j=...)  
...=...ai[j]...aip[j]...;  
}
```

© 2009 H. Nakashima

不連続データの通信 概論

- 南北通信: 送受信データ=連続 → OK
- 東西通信: 送受信データ=不連続
→ 連続するようにバッファにコピー?
 - 問題点1: 面倒くさい・間違いやすい
 - 問題点2: 非効率かもしれない

→ 派生データ型を使って直接通信

とりあえず角は必要ない(後述)

```

MPI_Sendrecv(esbuf, ny, MPI_DOUBLE, east, 0,
             wrbuf, ny, MPI_DOUBLE, west, 0, MCW, &st);
MPI_Sendrecv(wsbuff, ny, MPI_DOUBLE, west, 0,
             esbuf, ny, MPI_DOUBLE, east, 0, MCW, &st);

```

© 2009 H. Nakashima

不連続データの通信 派生データ型

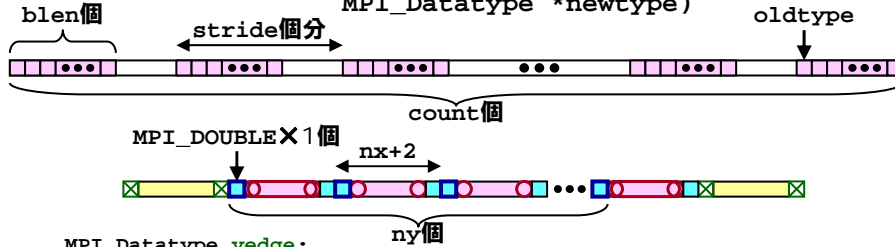
- 派生データ型 (derivative data type)
 - 基本型(MPI_DOUBLE など)・他の派生型を組み合わせで作られる送受信(およびファイルI/O用)の型
 - 内部に穴がある不連続データ型も定義可能
→ 送受信MPI関数が必要に応じて pack/unpack
 - 規則的な派生データ型生成関数
 - 東西通信 MPI_Type_contiguous() 同一型の単純な連続
 - MPI_Type_vector() 同一型の規則的な連続+不連続
 - MPI_Type_hvector() 同一型の規則的な連続+不連続
 - 不規則な派生データ型生成関数
 - 結果出力 MPI_Type_indexed() 同一型の不規則な連続+不連続
 - MPI_Type_hindexed() 同一型の不規則な連続+不連続
 - MPI_Type_struct() 非同一型の不規則配置

© 2009 H. Nakashima



不連続データの通信 東西通信用派生データ型

- `MPI_Type_vector(int count, int blen, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)`



```

MPI_Datatype vedge;
MPI_Type_vector(ny, 1, nx+2, MPI_DOUBLE, &vedge);
MPI_Type_commit(&vedge);
MPI_Sendrecv(&u[...], 1, vedge, east, 0, &u[...], 1, vedge, west, 0, MCW, &st);

```

作ったデータの使用開始宣言

```

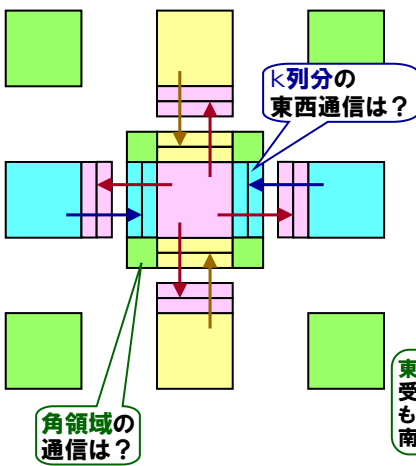
integer::vedge;
call MPI_Type_vector(ny, 1, nx+2, MPI_DOUBLE_PRECISION, vedge, err);
call MPI_Type_commit(vedge, err);
call MPI_Sendrecv(u(...), 1, vedge, east, 0, u(...), 1, vedge, west, 0, MCW, st, err);

```

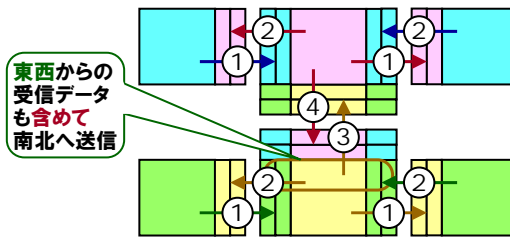
© 2009 H. Nakashima



不連続データの通信 通信回数削減への対応



- k列分の東西通信**
 - `blen=k` の `Type_vector()`
 $\neq 1$ 列分 $\times k$ 個
- 角領域通信**
 - 南西・南東 \rightarrow 南 \rightarrow 自分
 北西・北東 \rightarrow 北 \rightarrow 自分
 \neq 南西など \rightarrow 自分



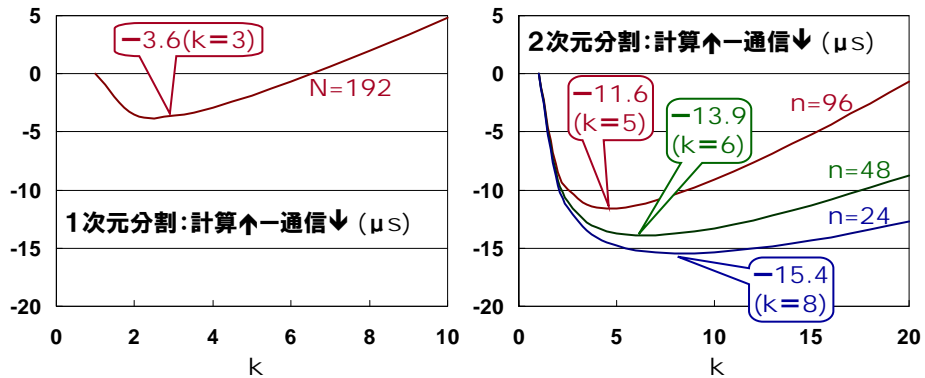
© 2009 H. Nakashima



不連続データの通信 通信回数削減の効果

| 分割 | 1次元 | 2次元 |
|-----|--------------|-------------------------------|
| 計算↑ | $k(k-1)Nt/k$ | $2k(k-1)(n+(2k-1)/3)t/k$ |
| 通信↓ | $2(k-1)L/k$ | $(4(k-1)L-4(k-1)(k-1+n)/B)/k$ |

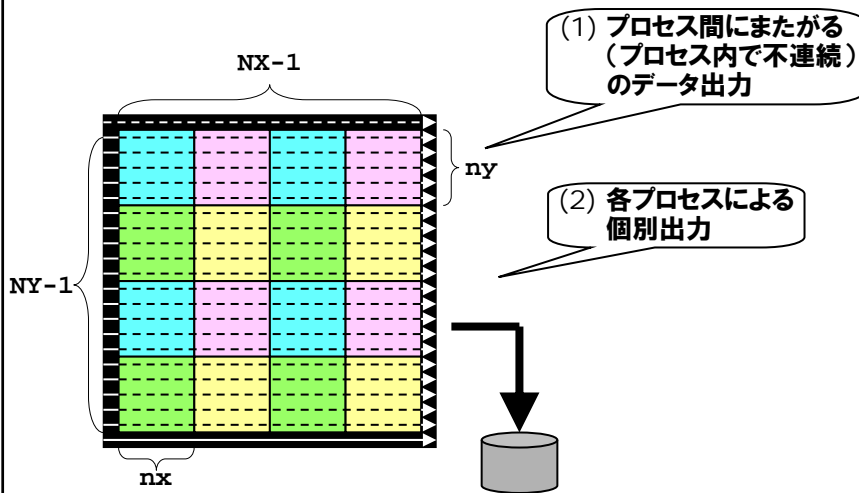
$t=4\text{ns/格子点}$ $L=5\mu\text{s}$ $B=2\text{GB/s}$



© 2009 H. Nakashima



2次元分散データの出力 概論

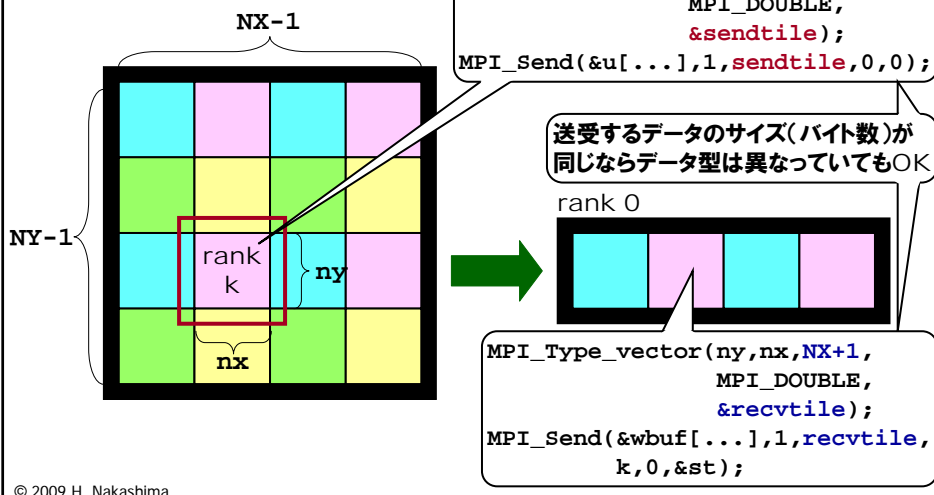


© 2009 H. Nakashima



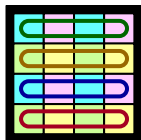
2次元分散データの出力 rank0 による出力

- プロセス配列1行分を rank 0 にまとめる
 良くない方法



2次元分散データの出力 西端プロセスによる出力

- プロセス配列1行分を西端にまとめる **良くない方法**
 - 同じ行のプロセスからなる communicator 生成



- `MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm *newcomm)`
- `newcomm=color` が自分と同じプロセスの集合
- `newcomm` での `rank=key` の昇順

- 行プロセス集合の中で西端に `MPI_Gather()`

- 例:

```

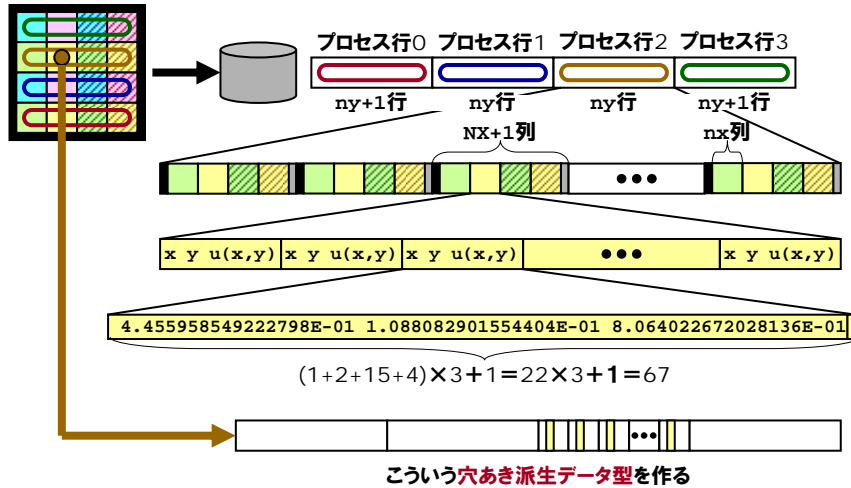
MPI_Comm row;
MPI_cart_coords(cart, me, 2, c);
MPI_Comm_split(cart, c[0], c[1], &row);
MPI_Gather(&u[...], ..., wbuf, ..., 0, row);

```

© 2009 H. Nakashima

2次元分散データの出力 個別出力: 概論

- やりたいこと = プロセスの位置により定まるファイル中の不連続領域への分散書込み



© 2009 H. Nakashima

2次元分散データの出力 個別出力: ファイルの open/close (1/2)

- open等のMPI関数: 全プロセスが同一引数で実行

- `MPI_File_open(MPI_Comm comm, char *name, int mode, MPI_Info info, MPI_File *fh)` ファイルに関するヒント
 - comm で共有する名前が name のファイルを開く
 - アクセスモード (mode) = 以下(など)の定数の論理和
 - `MPI_MODE_RDONLY`: 読み専用
 - `MPI_MODE_RDWR`: 読み & 書き込み
 - `MPI_MODE_WRONLY`: 書き込み専用
 - `MPI_MODE_APPEND`: 追加(書き込み)
 - `MPI_MODE_CREATE`: ファイルがなければ生成
- `MPI_File_set_size(MPI_File fh, int size)`
 - ファイルの(初期)サイズを設定
 - 書き込みオープンでも既存ファイルの初期サイズは0ではない!!
- `MPI_File_close(MPI_File *fh)`

© 2009 H. Nakashima



2次元分散データの出力 個別出力:ファイルの open/close (2/2)

■ C の例

```

MPI_File udata;
MPI_File_open(cart, "...",
               MPI_MODE_WRONLY | MPI_MODE_CREATE,
               MPI_INFO_NULL, &udata);
MPI_File_set_size(udata, 0);
...
MPI_File_close(&udata);

```

書込み & 存在しなければ生成

特にヒントはなし

■ Fortran の例

```

integer :: udata
call MPI_File_open(cart, '...', &
                  ior(MPI_MODE_WRONLY, MPI_MODE_CREATE), &
                  MPI_INFO_NULL, udata, err)
call MPI_File_set_size(udata, 0, err)
...
call MPI_File_close(udata, err)

```

存在した場合初期サイズを0にしないと元より小さいファイルになった場合にゴミが残る

© 2009 H. Nakashima



2次元分散データの出力 個別出力:個別書込みパターンの設定

■ ファイルの view の設定

```

MPI_File_set_view(MPI_File fh, MPI_Offset disp,
                  MPI_Datatype etype, MPI_Datatype ftype,
                  char *drep, MPI_Info info)

```

ファイル形式情報
京大スパコンでは
"native"

etype

ファイルアクセスの
最小単位の型



プロセス固有の
アクセスパターン
を示すデータ型
(filetype)



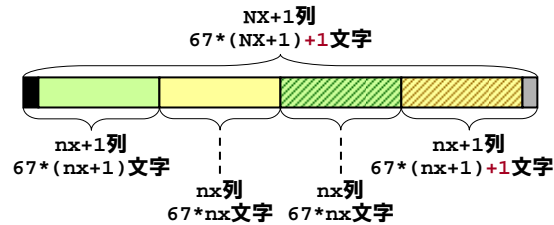
1つの ftype を超えると
ftype の繰り返し

© 2009 H. Nakashima

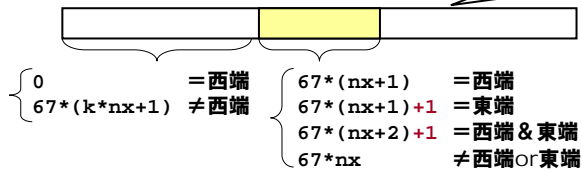


2次元分散データの出力 個別出力: filetype の生成 (1/7)

- k 列目のプロセスの配列1行分のアクセスパターン



こういう派生データ型を
`MPI_Type_struct()` で作る

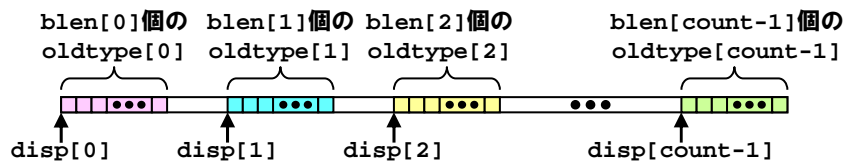


© 2009 H. Nakashima

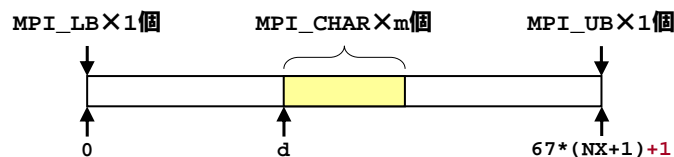


2次元分散データの出力 個別出力: filetype の生成 (2/7)

- `MPI_Type_struct(int count, int *blen, MPI_Aint *disp, MPI_Datatype *oldtype, MPI_Datatype *newtype)`



- `MPI_LB`: 派生型の下端を定めるための仮想データ型
- `MPI_UB`: 派生方の上端を定めるための仮想データ型



© 2009 H. Nakashima



2次元分散データの出力 個別出力: filetype の生成 (3/7)

■ C の例

```

■ #define LW 67
...
MPI_Datatype rtype;
MPI_Aint disp[3]; int blen[3];
MPI_Datatype types[3]={MPI_LB,MPI_CHAR,MPI_UB};
...
MPI_Dims_create(np,2,dims);
MPI_Cart_coords(cart,me,2,c);
...
disp[0]=0; disp[2]=LW*(NX+1)+1;
disp[1]=(c[1]==0) ? 0 : LW*(c[1]*nx+1);
blen[0]=blen[2]=1;
blen[1]=LW*nx;
if(c[1]==0) blen[1]+=LW;
if(c[1]==dims[1]-1) blen[1]+=LW+1;
MPI_Type_struct(3,blen,disp,types,&rtype);

```

両方とも
成立する
ことがある

© 2009 H. Nakashima



2次元分散データの出力 個別出力: filetype の生成 (4/7)

■ Fortran の例

```

■ integer,parameter::LW=67
integer::rtype;
integer::disp(3),blen(3)
integer::types(3)=&
    (/MPI_LB,MPI_CHARACTER,MPI_UB/)
...
call MPI_Dims_create(np,2,dims,err)
call MPI_Cart_coords(cart,me,2,c,err)
...
disp(1)=0; disp(3)=LW*(NX+1)+1
disp(2)=0; if (c(2) /= 0) disp(2)=LW*(c(2)*nx+1)
blen(1)=1; blen(3)=1; blen(2)=LW*nx
if(c(2)==0) blen(2)=blen(2)+LW
if(c(2)==dims(2)-1) blen(2)=blen(2)+LW+1
call MPI_Type_struct(3,blen,disp,types,&
    rtype,err)

```

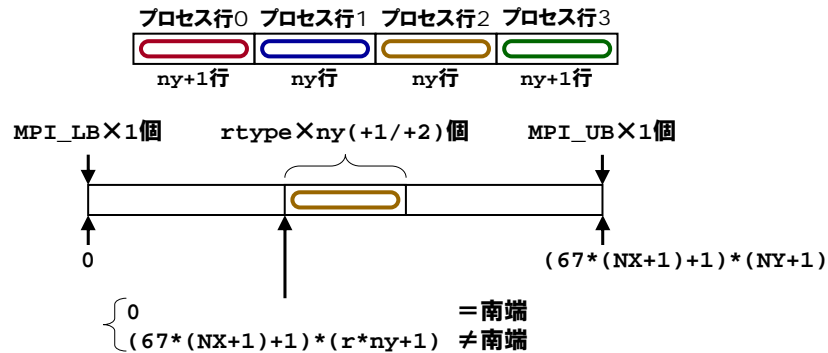
両方とも
成立する
ことがある

© 2009 H. Nakashima



2次元分散データの出力 個別出力: filetype の生成 (5/7)

■ r 行目のプロセスのアクセスパターン



© 2009 H. Nakashima



2次元分散データの出力 個別出力: filetype の生成 (6/7)

■ C の例

```

■ MPI_Datatype ftype;
...
rowwidth=LW*(NX+1)+1
disp[2]=rowwidth*(NY+1);
disp[1]=(c[0]==0) ? 0 : rowwidth*(c[0]*ny+1);
blen[1]=ny;
if(c[0]==0) blen[1]++;
if(c[0]==dims[0]-1) blen[1]++;
types[1]=rtype;
MPI_Type_struct(3,blen,disp,types,&ftype);
MPI_Type_commit(&ftype);
MPI_File_set_view(udata,0,MPI_CHAR,ftype,
                  "native",MPI_INFO_NULL);

```

両方とも
成立する
ことがある

© 2009 H. Nakashima



2次元分散データの出力 個別出力: filetype の生成 (7/7)

■ Fortran の例

```

■ integer::ftype;
...
rowwidth=LW*(NX+1)+1
disp(3)=rowwidth*(NY+1)
disp(2)=0
if (c(1)!=0) disp(2)=rowwidth*(c(1)*ny+1)
blen(2)=ny
if(c(1)==0) blen(2)=ny+1
if(c(1)==dims(1)-1) blen(2)=ny+1
types(2)=rtype
call MPI_Type_struct(3,blen,disp,types,&
                    filetype,err)
call MPI_Type_commit(ftype,err)
call MPI_File_set_view(udata,0,MPI_CHARACTER,&
                    ftype,"native",MPI_INFO_NULL,err)

```

両方とも
成立する
ことがある

© 2009 H. Nakashima



2次元分散データの出力 個別出力: ファイルの書込み (1/3)

- MPI_File_write(MPI_File fh, void *buf, int count, MPI_Datatype dtype, MPI_Status *st)
 - buf から count 個の dtype データを fh に書込み
 - dtype のデータがある部分を繋いだもの = etype を並べたもの

■ あまり良くない(かもしれない)例

```

char wbuf[LW+1];
...
for(j=...){
  for(i=...) {
    sprintf(wbuf,"...",...,u[...]);
    MPI_File_write(udata,wbuf,LW,MPI_CHAR,&st);
  }
  if(c[1]==dims[1]-1)
    MPI_File_write(udata,"%n",1,MPI_CHAR,&st);
}

```

MPI_File_write() を
沢山実行すると、とりあえず
京大スパコンでは結構な時間
がかかった

© 2009 H. Nakashima



2次元分散データの出力 個別出力:ファイルの書込み (2/3)

- どのマシンでもそこそこ良い(はずの)例 (C版)

```
char *wbuf;
...
wbuf=(char*)malloc((LW*(nx+2)+2)*sizeof(char));
for(j=...){
    for(i=...,k=0;...;...,k+=LW)
        sprintf(wbuf+k,"...",...,u[...]);
    }
    if(c[1]==dims[1]-1)
        sprintf(wbuf+(k++),"%n");
    MPI_File_write(udata,wbuf,k,MPI_CHAR,&st);
}
```

© 2009 H. Nakashima



2次元分散データの出力 個別出力:ファイルの書込み (3/3)

- どのマシンでもそこそこ良い(はずの)例 (Fortran版)

```
subroutine print_u(u,...,bufsize,eastmost)
integer::bufsize
logical::eastmost
character(len=bufsize)::wbuf
...
do j=...; k=1
    do i=...
        write(wbuf(k:k+LW-1),'(...)')...,u(...); k=k+LW
    end do
    if(eastmost) then
        write(wbuf(k:k+LW),'(.../)'...),u(...)
        call MPI_File_write(udata,wbuf,k+LW,MPI_CHAR,st,err);
    else
        call MPI_File_write(udata,wbuf,k-1,MPI_CHAR,st,err);
    end do
end subroutine
```

© 2009 H. Nakashima



2次元分散データの出力 個別出力:まとめ (1/2)

■ Cのまとめ

```
MPI_File udata;
MPI_Datatype rtype,ftype;
...
MPI_File_open(&udata,...);
MPI_File_set_size(udata,0);
/* rtype の生成 */
/* ftype の生成 */
MPI_Type_commit(&ftype);
MPI_File_set_view(udata,...,ftype,...);
for(j=...){
    for(i=...){...} ...;
    MPI_File_write(udata);
}
MPI_File_close(&udata);
```

© 2009 H. Nakashima



2次元分散データの出力 個別出力:まとめ (2/2)

■ Fortran のまとめ

```
integer::udata,rtype,ftype
...
call MPI_File_open(udata,...)
call MPI_File_set_size(udata,0,err)
/* rtype の生成 */
/* ftype の生成 */
call MPI_Type_commit(ftype,err)
call MPI_File_set_view(udata,...,ftype,...)
do j=...
    do i=...
        ...
    end do
    call print_u(u,...,LW*(nx+2)+1,c(1)==dims(1)-1)
end do
MPI_File_close(&udata);
```

© 2009 H. Nakashima