CMSI配信講義B
第11回

大規模MD並列化の技術2

名古屋大学 工学研究科

CMSI 重点研究員

安藤 嘉倫

目次

- 分子動力学 (MD) 法
- 分子動力学計算の並列化特性
- 並列化技術 1　データ構造

第一回

- 並列化技術 2　MPI
- 並列化技術 3　OpenMP, SIMD

第二回

並列化技術 2

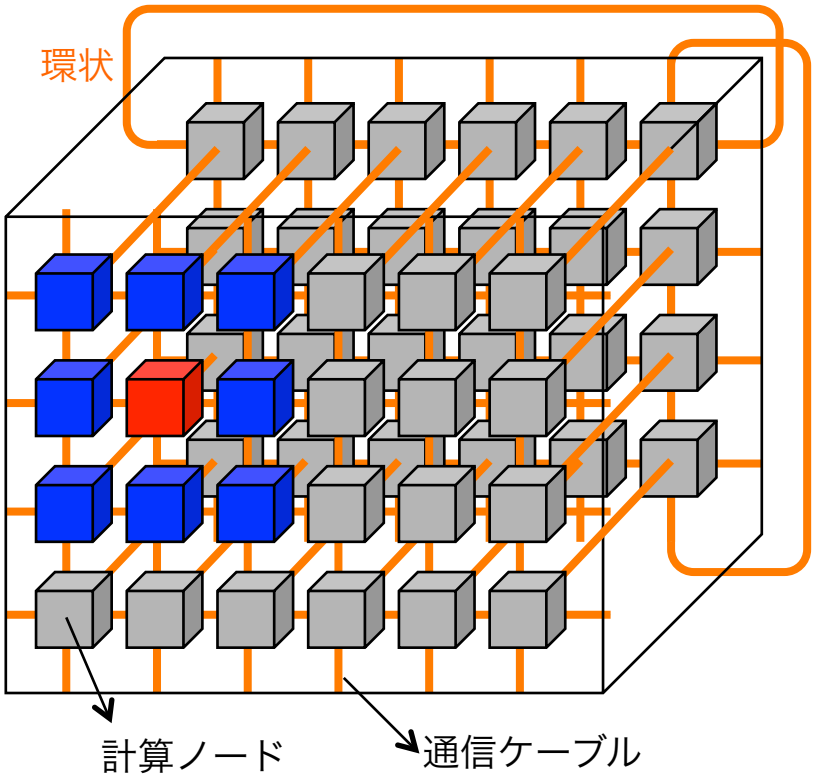
MPI 並列化技術 (特に 3 次元トーラスネットワークに最適化)

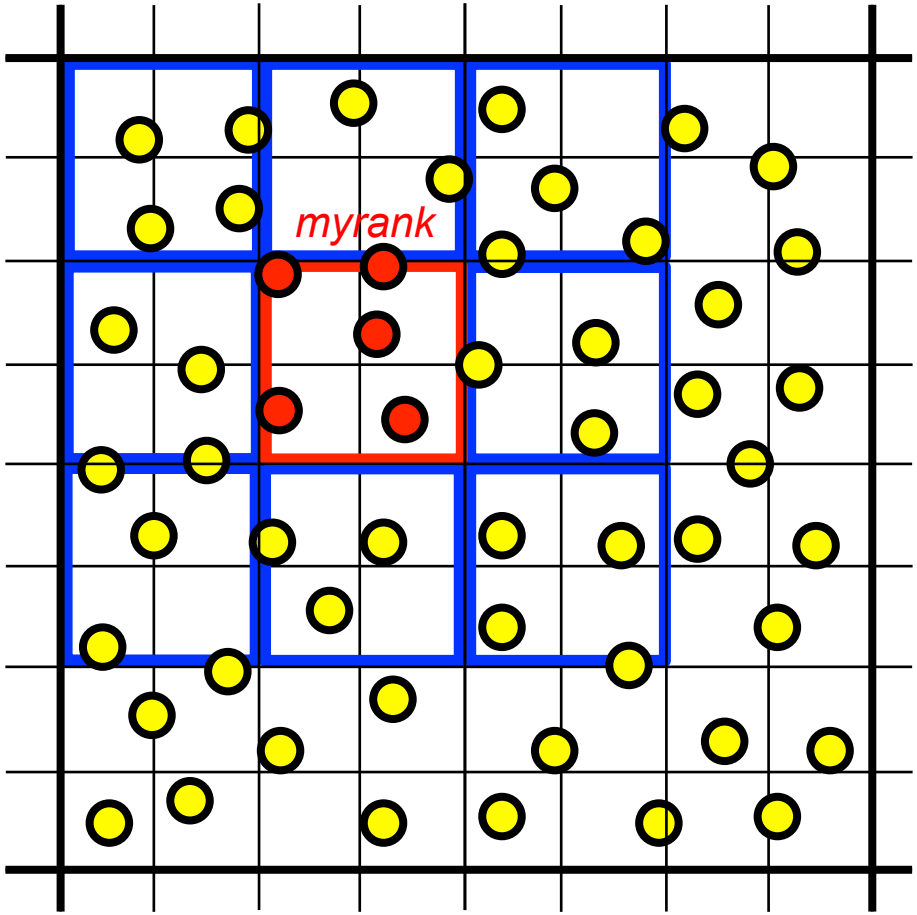
- 通信衝突の回避
- 通信前後での配列間コピーの消去
- 通信の演算による代用

ただし簡単のため, 各項目の説明は主に二次元平面上で行います

3 次元トーラスネットワーク

例) 京, FX10, Blue Gene, Anton





長所 : 空間分割によるMPI並列化との相性が良い.

短所 : 隣接ノードとの通信のみ可能. 斜め方向ノードとの通信が直にできないため
通信の回数 (ホップ数) が増加する.プロセス間通信の際に衝突が生じやすい.

mpi_sendrecv 関数

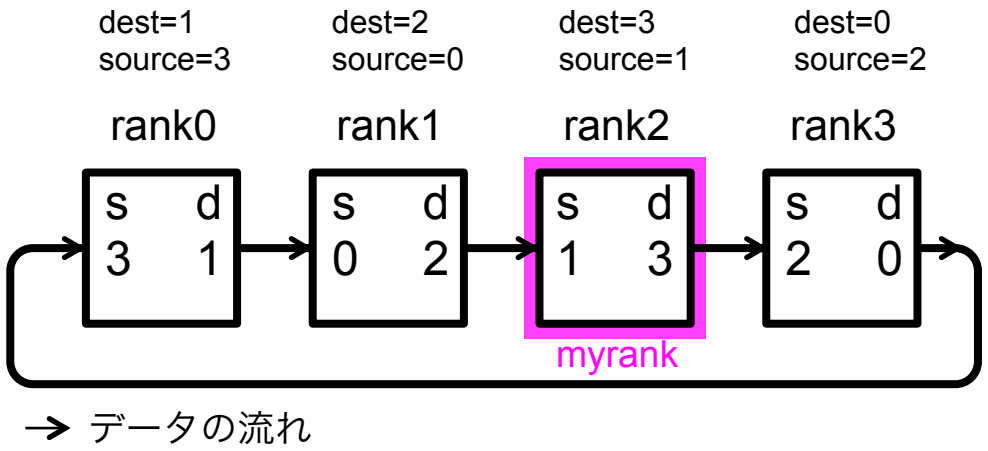
mpi_send + mpi_recv
送信　受信

call MPI_SENDRECV (
sendbuf,scount,stype,dest,stag,
recvbuf,rcount,rtype,source,rtag,comm,status,ierr)

sendbuf : 送信データ
scount : 送信データ数
stype : 送信データの型
dest : 送信先プロセス番号(rank)
stag : 送信タグ

recvbuf : 受信データ
rcount : 受信データ数
rtype : 受信データの型
source : 受信元プロセス番号
rtag : 受信タグ

comm : コミュニケータ
status : 状態
ierr : 完了コード



commで指定したプロセスグループ内で環状のシフト通信が可能.

通信相手は両隣のプロセス.

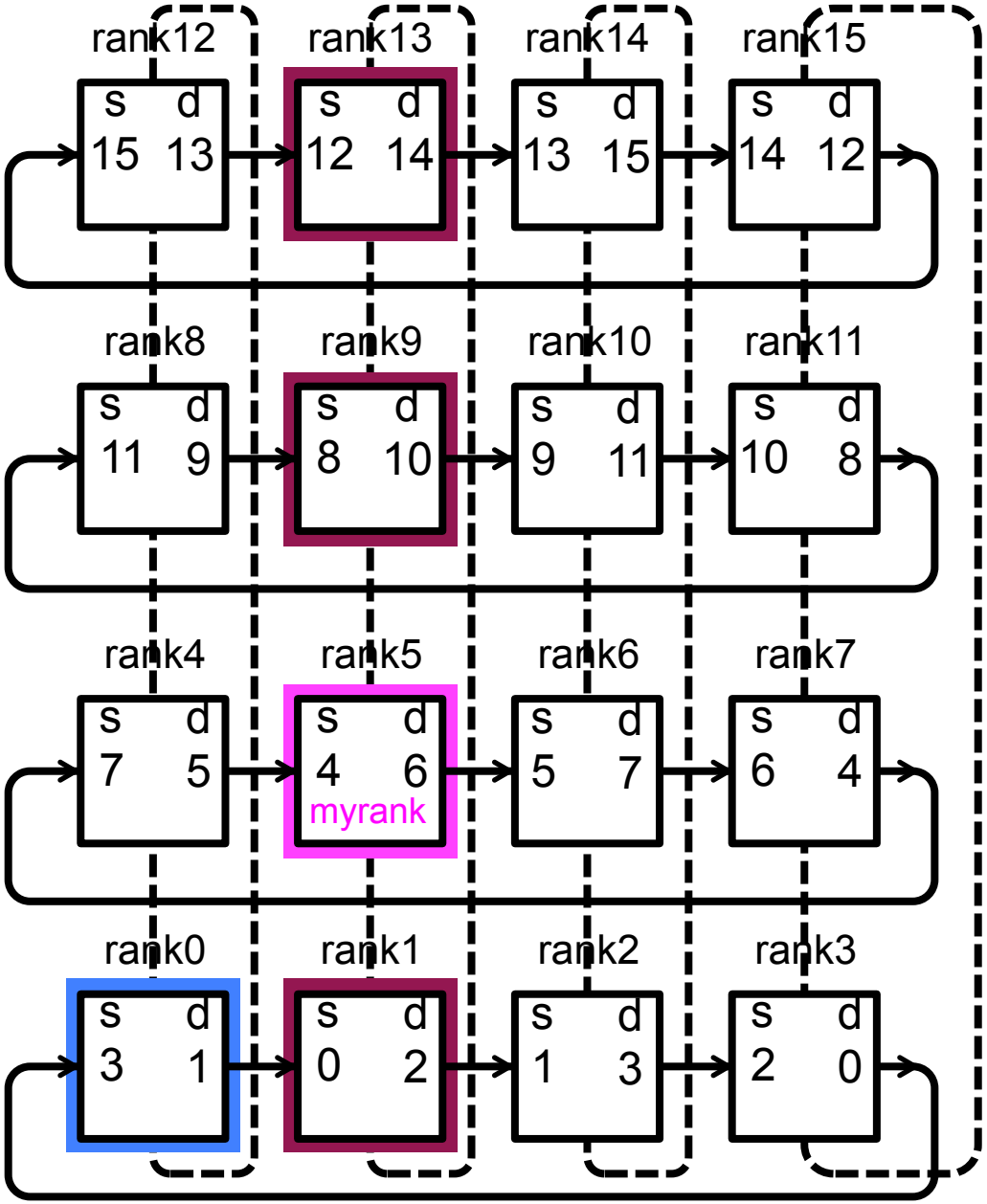
3Dトーラスネットワークでの通信に適する関数

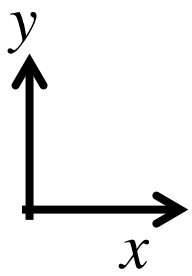
myrank からみると, call mpi_sendrecv() の結果
下流の rank のデータが受信される.

mpi_sendrecv 関数

2次元

+x方向



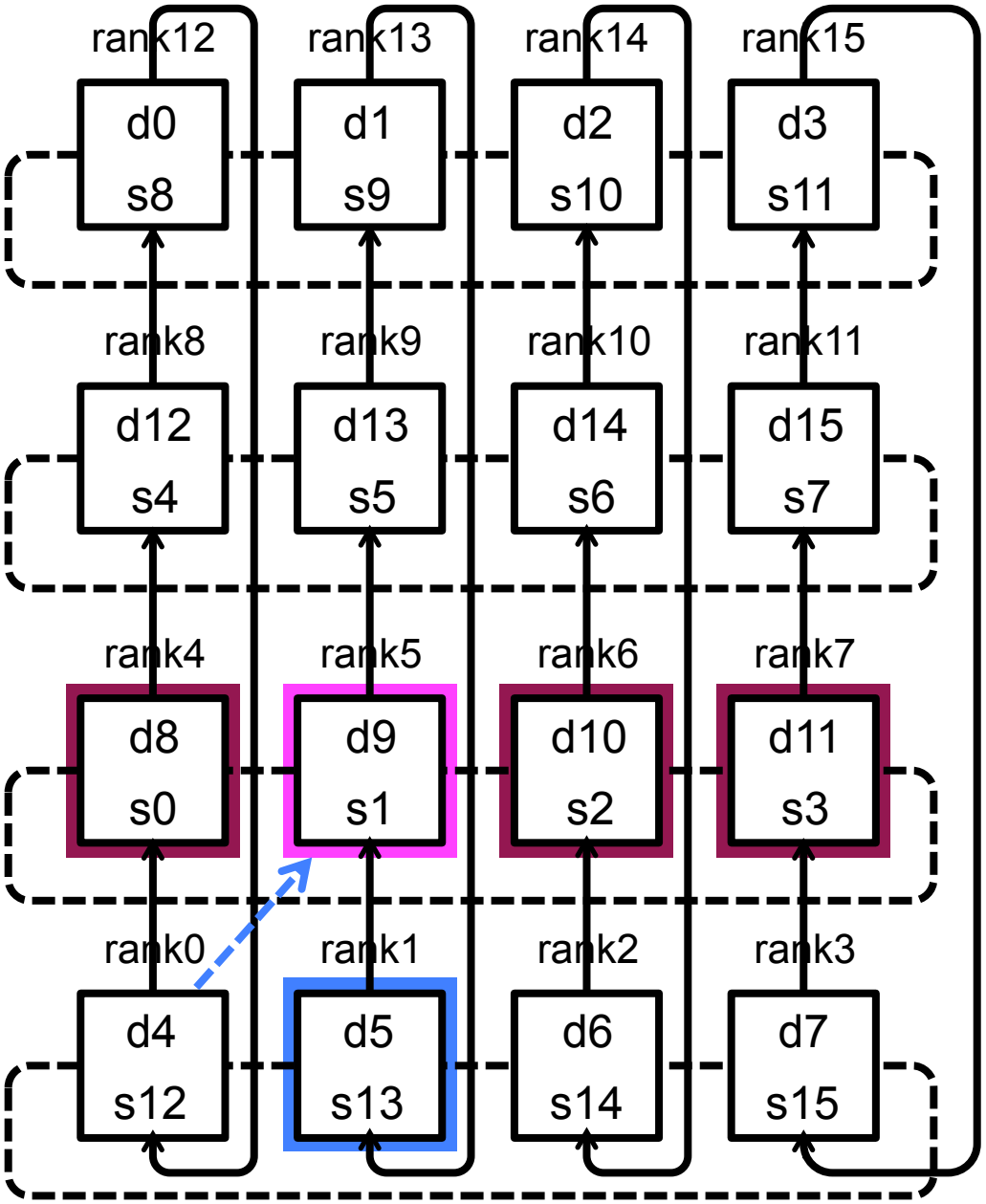


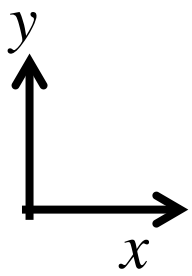
→ データの流れ

mpi_sendrecv 関数

2次元

+y方向

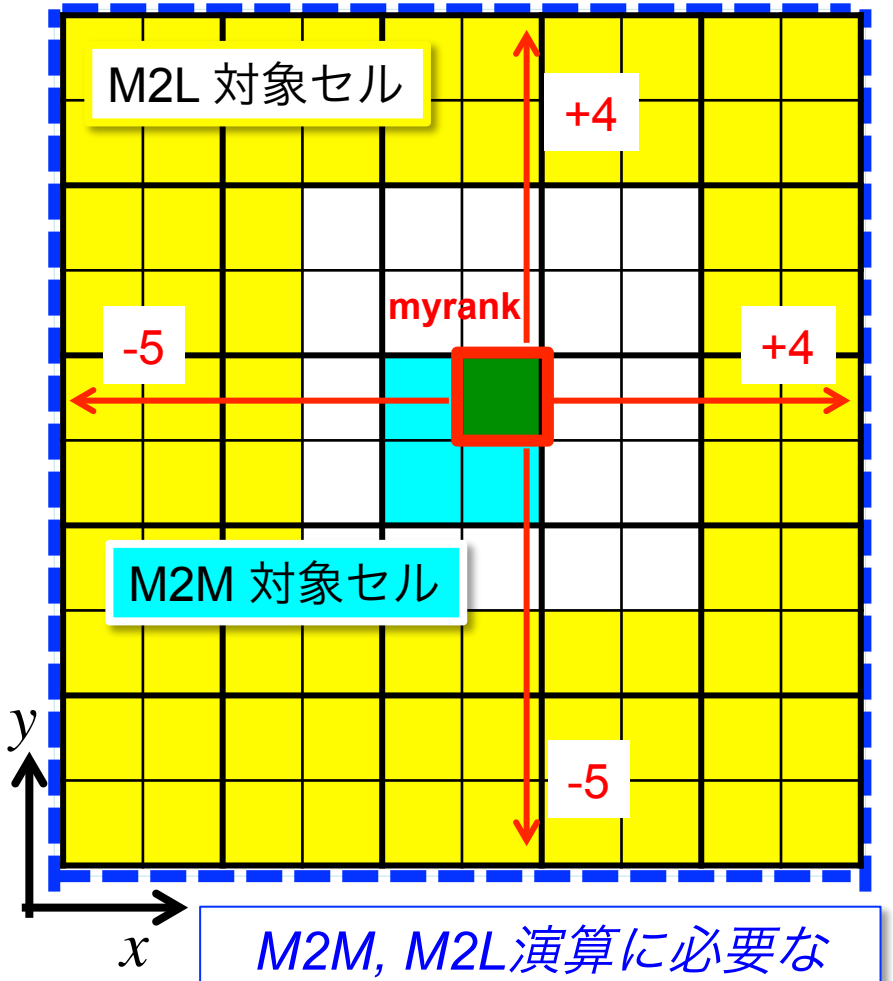




→ データの流れ

MPI 並列化技術: 通信衝突の回避

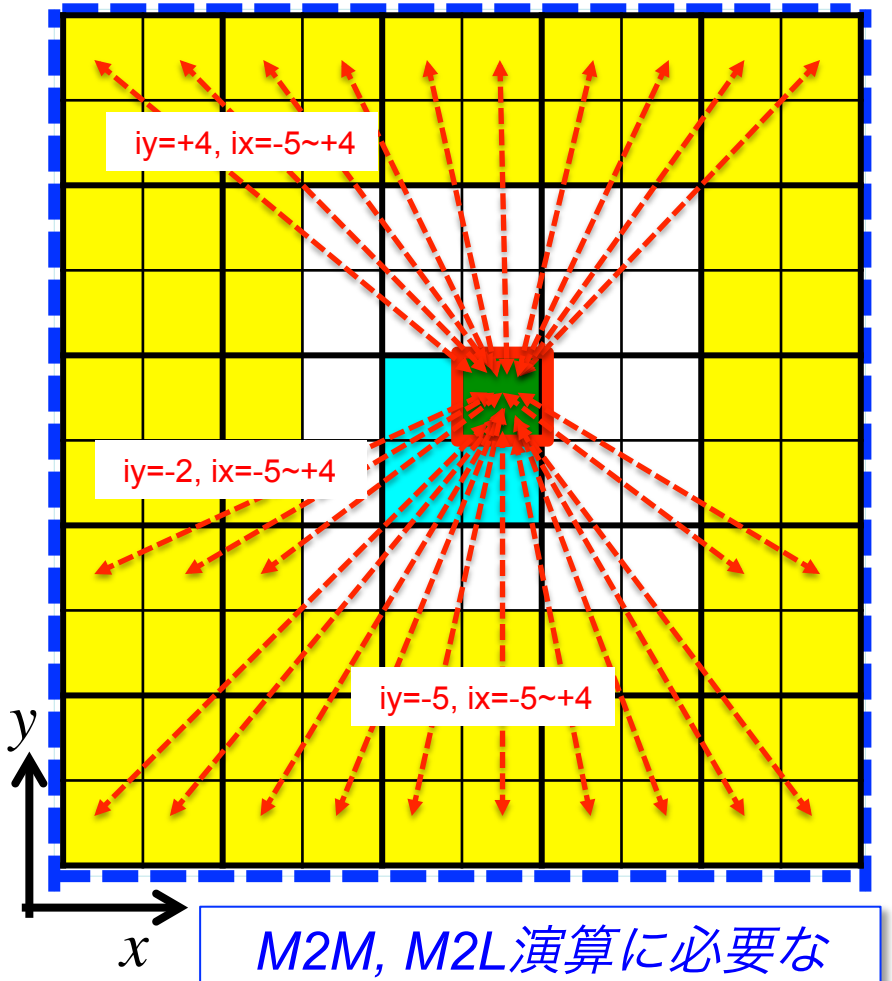
例) 多極子の通信 (1スーパーセル/プロセス)



M2M, M2L演算に必要な
データ範囲
(白抜き部分は除く)

MPI 並列化技術: 通信衝突の回避

例) 多極子の通信 (1スーパーセル/プロセス)



M2M, M2L演算に必要なデータ範囲 (白抜き部分は除く)

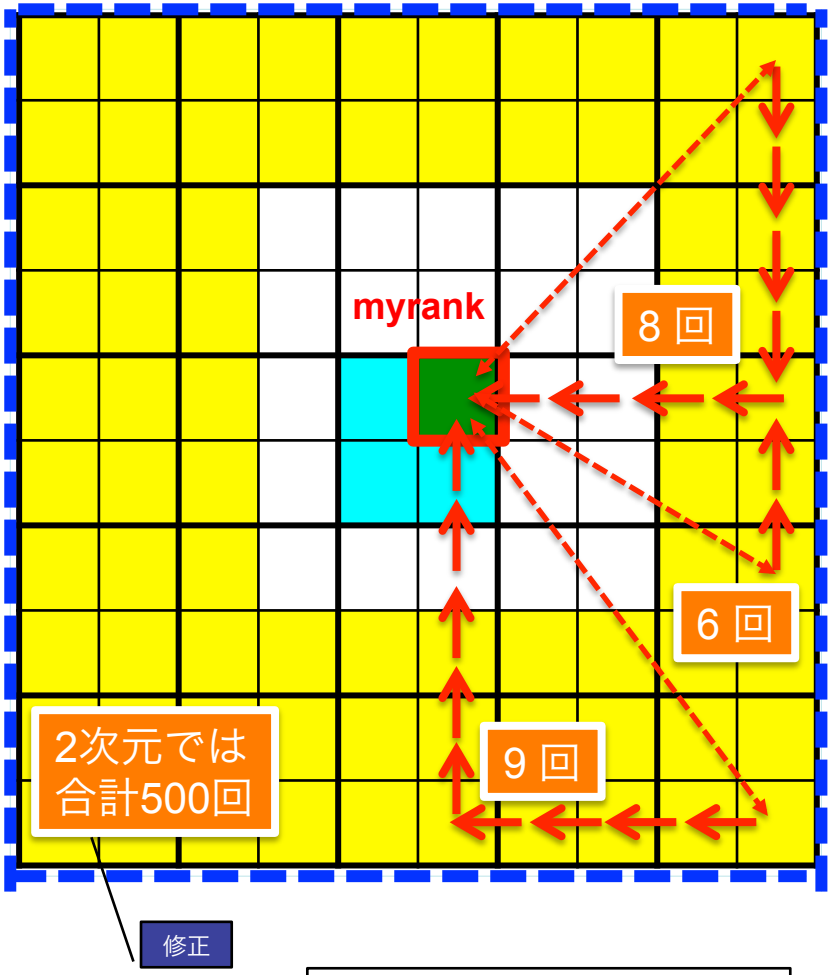
コーディングイメージ

```
do iy=-5,+4
do ix=-5,+4
   ip_dest   =送信先プロセス番号
   ip_src    =受信元プロセス番号
   IF( (ix, iy) が白抜き部分 ) cycle
   call mpi_sendrecv(..,ip_dest,..,ip_src,..)
enddo
enddo
```

このままだと$(10^3-5^3)+(2^3-1)$ =882 回のプロセス間通信 **[3次元]**

MPI 並列化技術: 通信衝突の回避

例) 多極子の通信 (1スーパーセル/プロセス)



用語の定義：

ホップ:
隣接ノードとの通信

コーディングイメージ

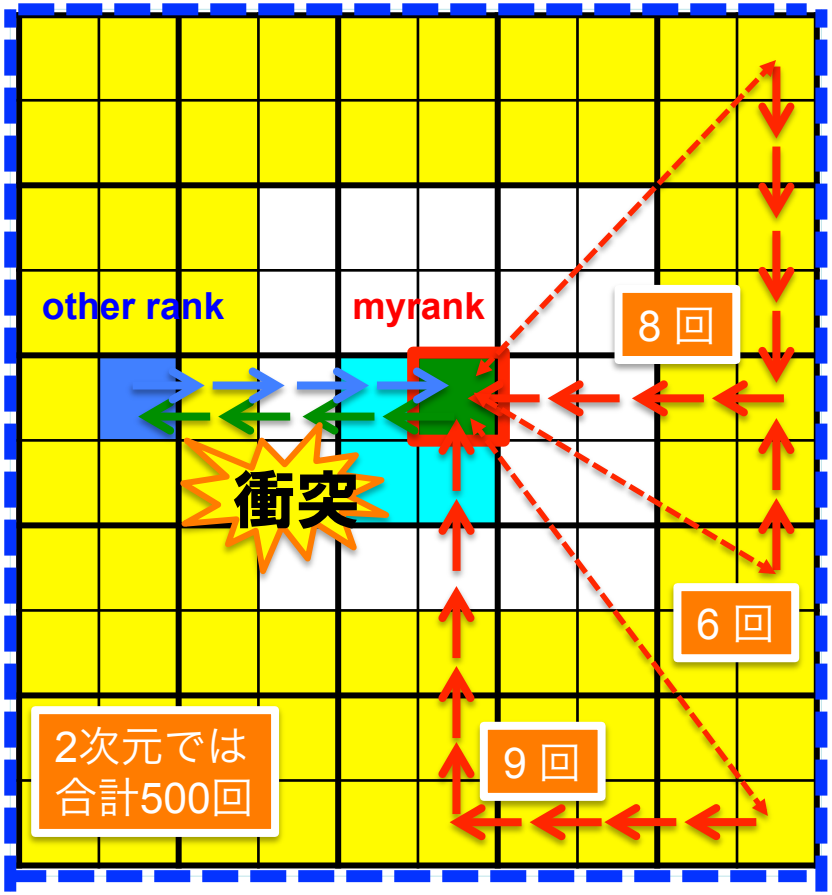
```
do iy=-5,+4
do ix=-5,+4
   ip_dest   =送信先プロセス番号
   ip_src    =受信元プロセス番号
   IF( (ix, iy) が白抜き部分 ) cycle
   call mpi_sendrecv(..,ip_dest,..,ip_src,..)
enddo
enddo
```

このままだと$(10^3-5^3)+(2^3-1)$ =882 回のプロセス間通信 **[3次元]**

＋ トーラスネットワークに特有の問題:
(1) 882個の通信先ごと多数のホップ回数

MPI 並列化技術: 通信衝突の回避

例) 多極子の通信 (1スーパーセル/プロセス)



コーディングイメージ

```
do iy=-5,+4
do ix=-5,+4
   ip_dest   =送信先プロセス番号
   ip_src    =受信元プロセス番号
   IF( (ix, iy) が白抜き部分 ) cycle
   call mpi_sendrecv(..,ip_dest,..,ip_src,..)
enddo
enddo
```

このままだと$(10^3-5^3)+(2^3-1)$ =882 回のプロセス間通信 **[3次元]**

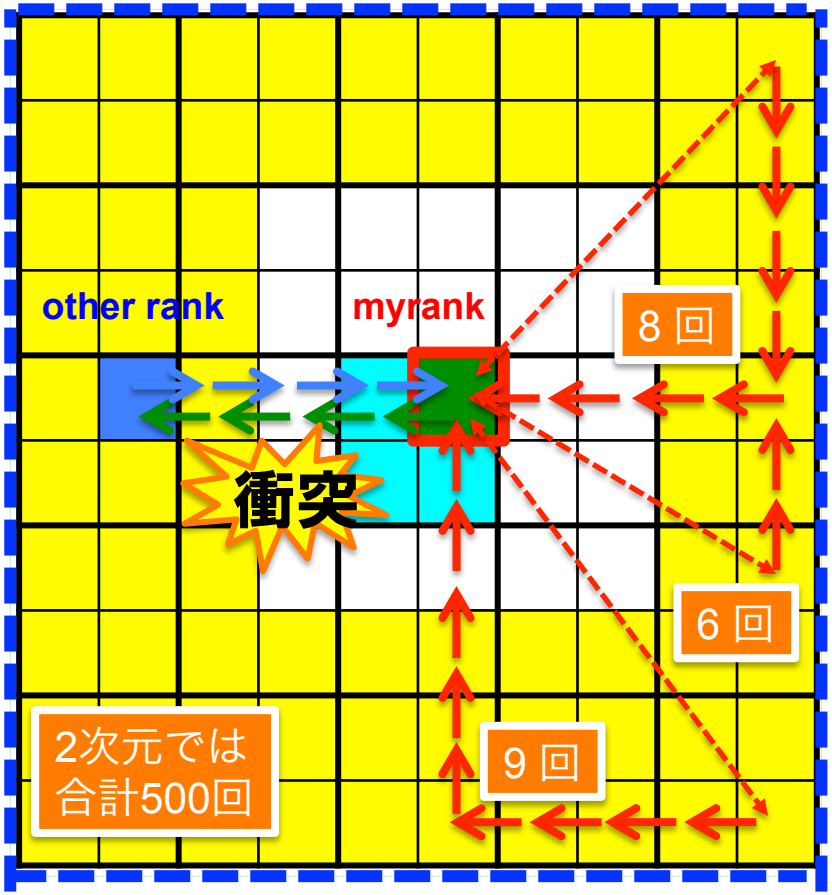
＋ トーラスネットワークに特有の問題:
(1) 882個の通信先ごと多数のホップ回数
(2) 通信の衝突が至る所で発生

用語の定義：

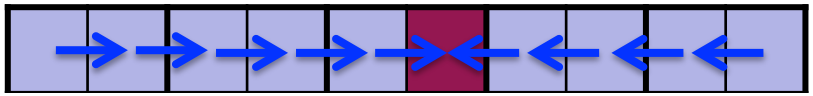
ホップ:
隣接ノードとの通信

MPI 並列化技術: 通信衝突の回避

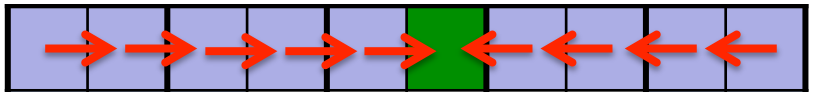
例) 多極子の通信 (1スーパーセル/プロセス)



新しい通信コード

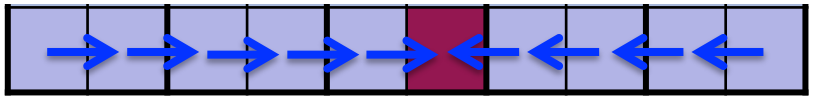


1) ± x 軸方向通信 9 回



全てのプロセスが
・同じタイミング
・同じ方向

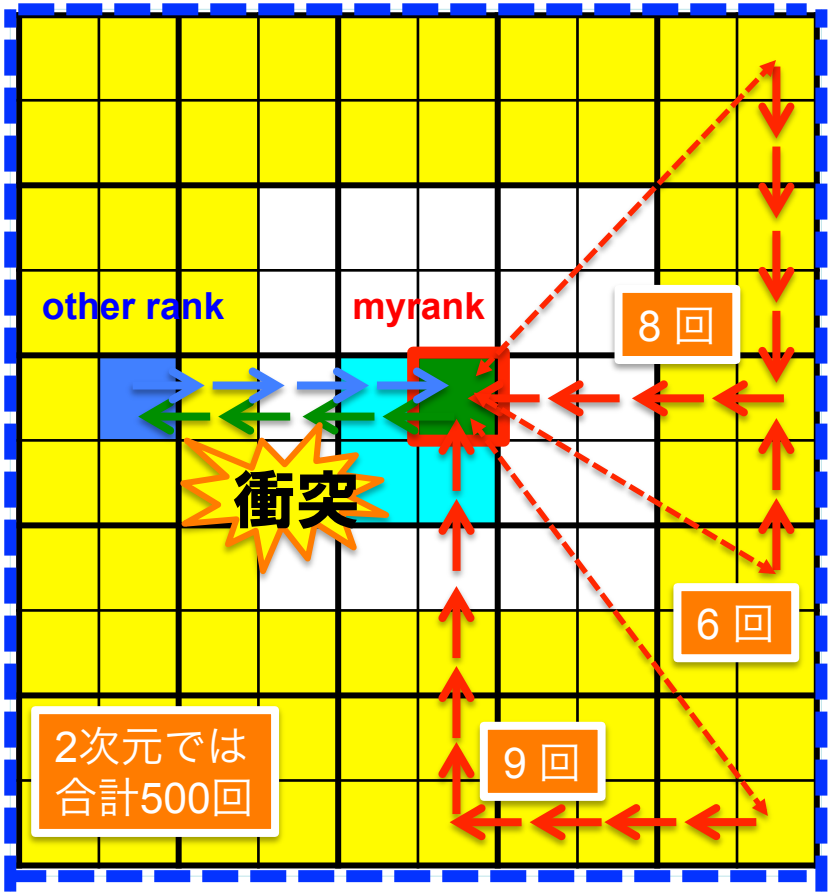
衝突回避



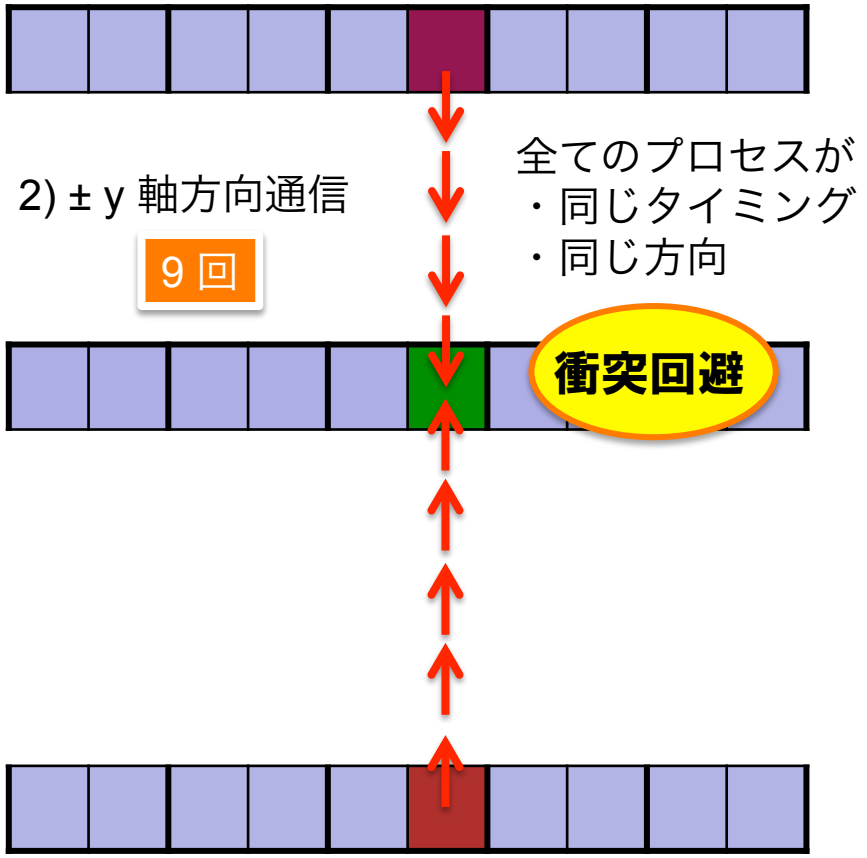
トーラスネットワークに特有の問題:
(1) 882個の通信先ごと多数のホップ回数
(2) 通信の衝突が至る所で発生

MPI 並列化技術: 通信衝突の回避

例) 多極子の通信 (1スーパーセル/プロセス)



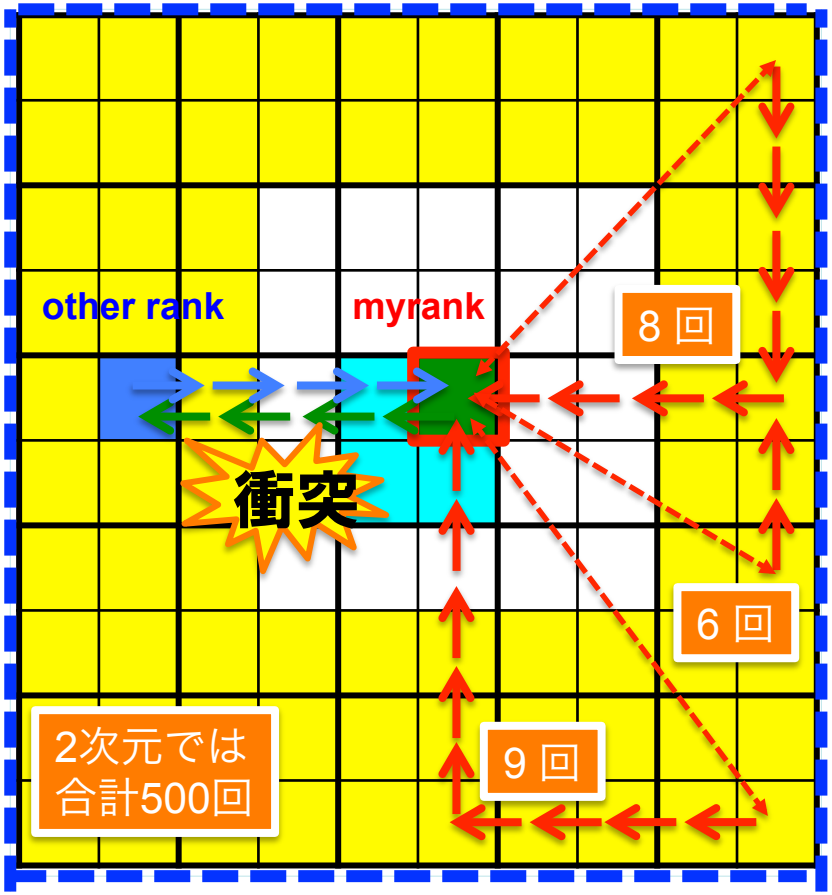
新しい通信コード



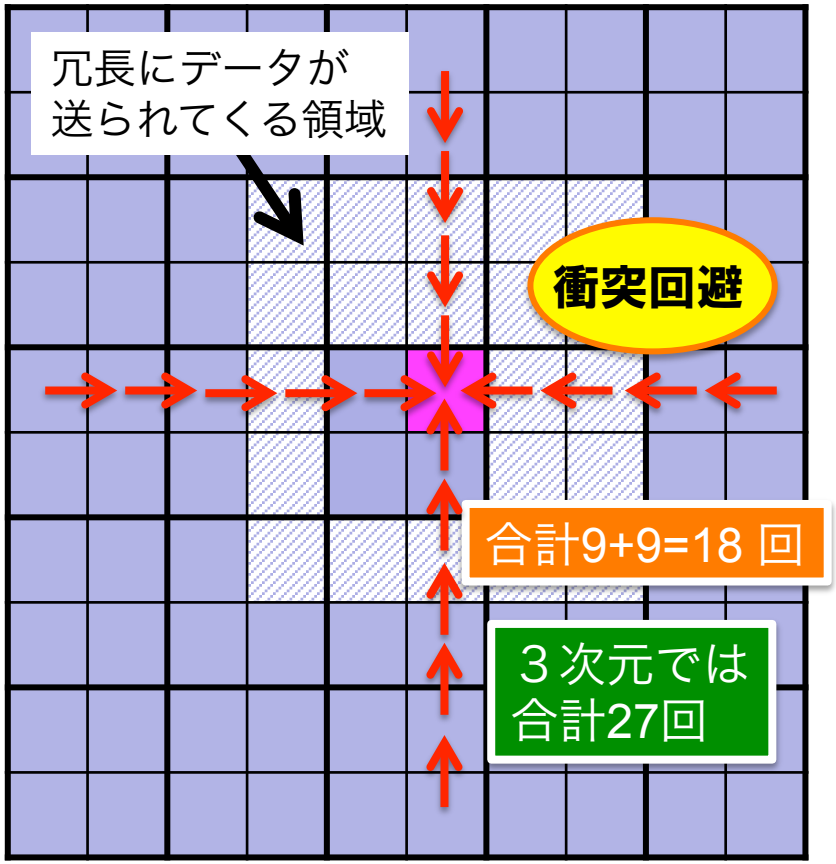
トーラスネットワークに特有の問題:
(1) 882個の通信先ごと多数のホップ回数
(2) 通信の衝突が至る所で発生

MPI 並列化技術: 通信衝突の回避

例) 多極子の通信 (1スーパーセル/プロセス)



新しい通信コード



トーラスネットワークに特有の問題:

(1) 882個の通信先ごと多数のホップ回数
(2) 通信の衝突が至る所で発生

⇒

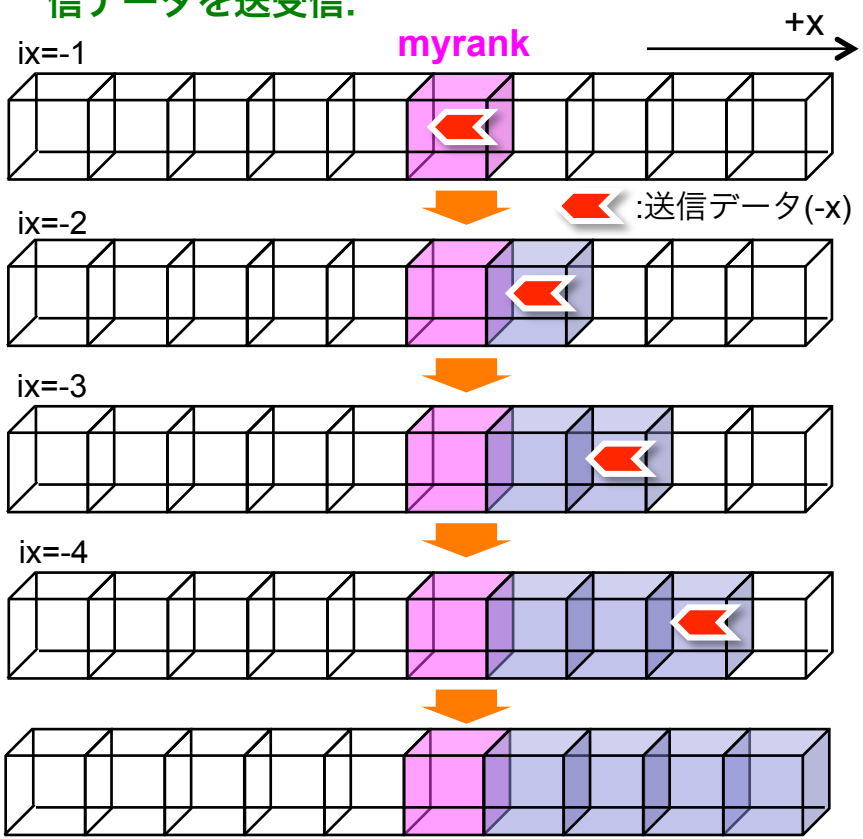
(1) ホップ回数を最小限に
(2) 通信の衝突を回避

MPI 並列化技術: 通信衝突の回避

x方向通信コーディングイメージ

```
ipx_dest =送信先プロセス番号(-x)
ipx_src  =受信元プロセス番号(-x)
do ix= -1, -4, -1
   call mpi_sendrecv(..,ipx_dest,..,ipx_src,..)
enddo
```

袖部つき局所化されたデータ構造の所定位置に受信データを格納.　次回通信では前回の受信データを送受信.



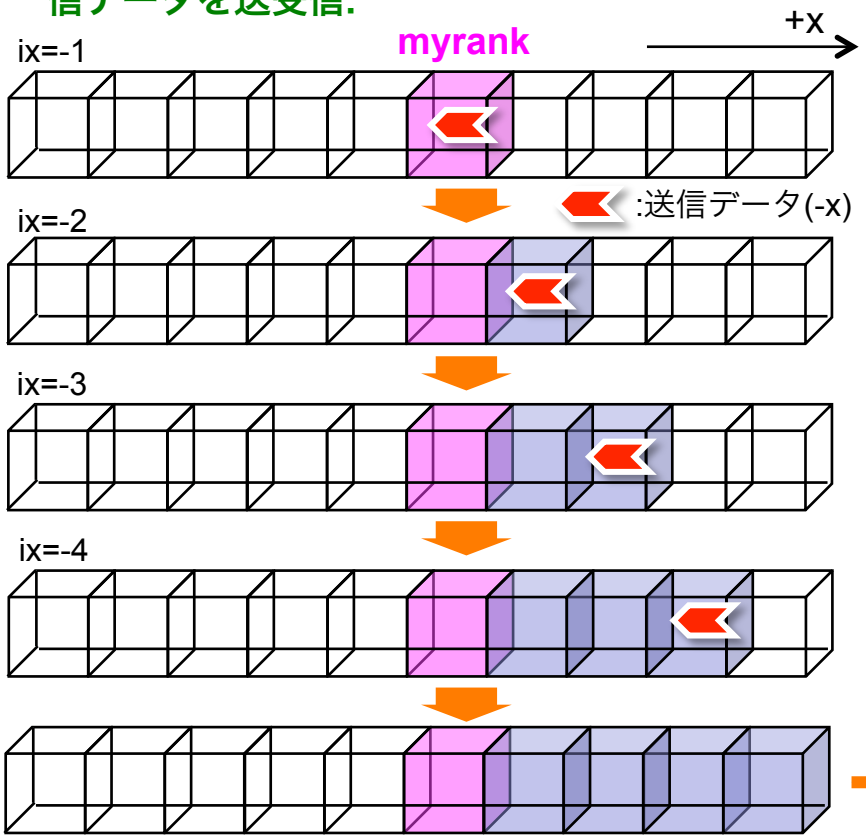
注) call mpi_sendrecv() の結果, myrank からみた -x 方向通信により +x 方向のデータを受信

MPI 並列化技術: 通信衝突の回避

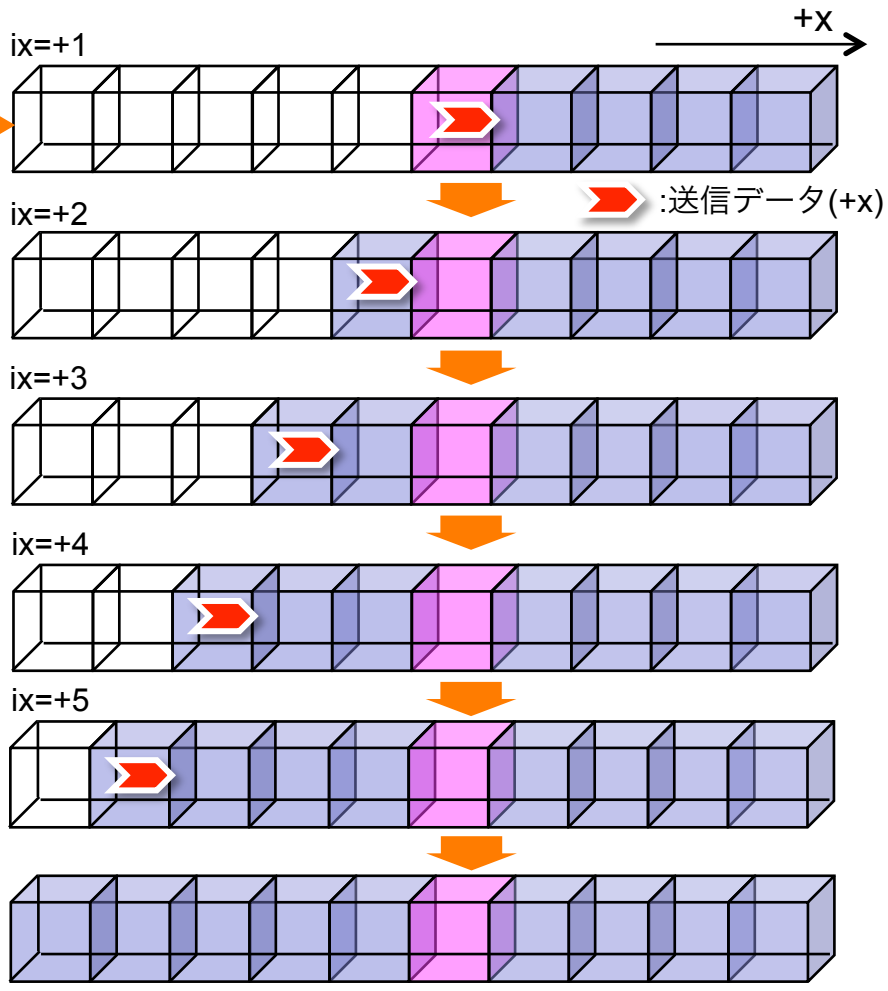
x方向通信コーディングイメージ

```
ipx_dest =送信先プロセス番号(-x)
ipx_src   =受信元プロセス番号(-x)
do ix= -1, -4, -1
   call mpi_sendrecv(..,ipx_dest,..,ipx_src,..)
enddo
```

袖部つき局所化されたデータ構造の所定位置に受信データを格納.　次回通信では前回の受信データを送受信.




```
ipx_dest =送信先プロセス番号(+x)
ipx_src   =受信元プロセス番号(+x)
do ix= +1, +5, +1
   call mpi_sendrecv(..,ipx_dest,..,ipx_src,..)
enddo
```

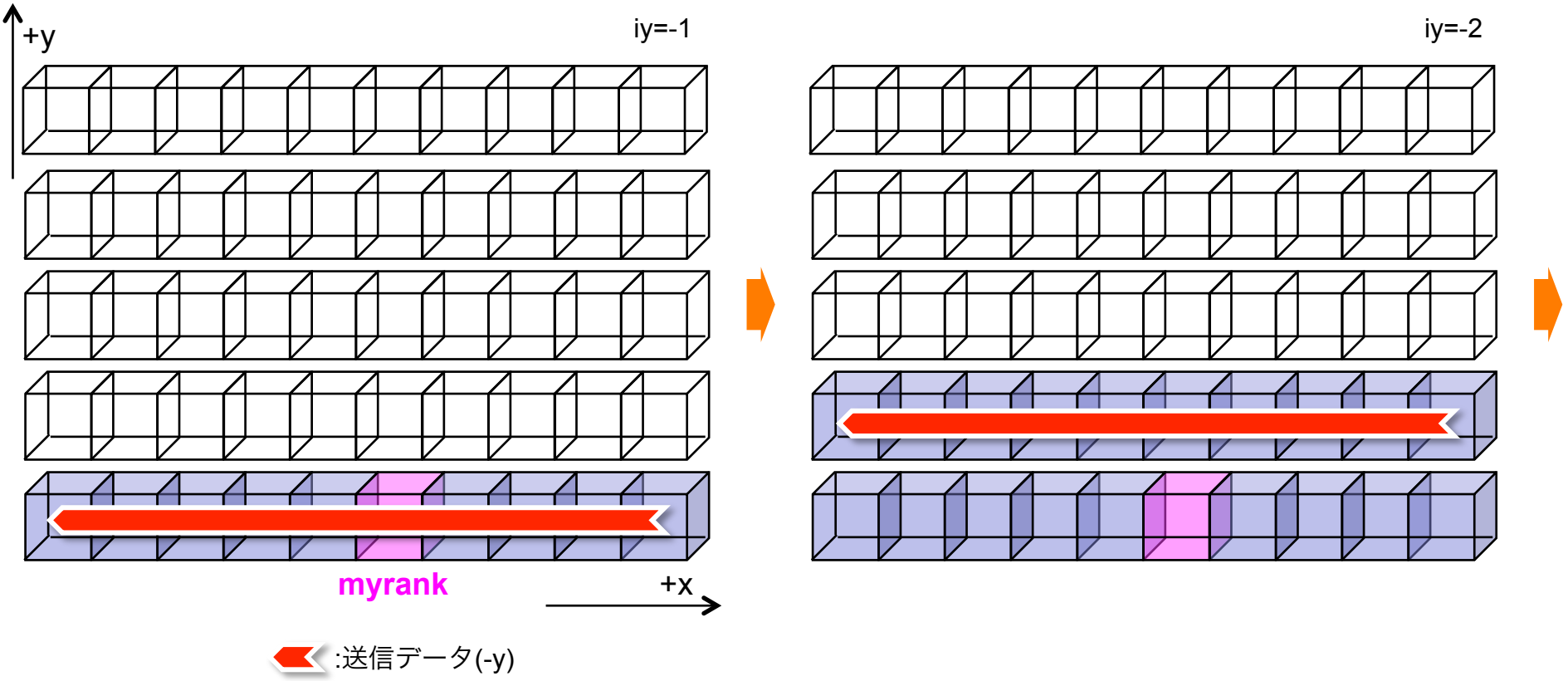


MPI 並列化技術: 通信衝突の回避

y方向通信コーディングイメージ

```
ipy_dest =送信先プロセス番号(-y)
ipy_src   =受信元プロセス番号(-y)
do iy= -1, -4, -1
   call mpi_sendrecv(..,ipy_dest,..,ipy_src,..)
enddo
```

袖部つき局所化されたデータ構造の所定位置に棒状の受信データを格納.　次回通信では前回の棒状の受信データを送受信.

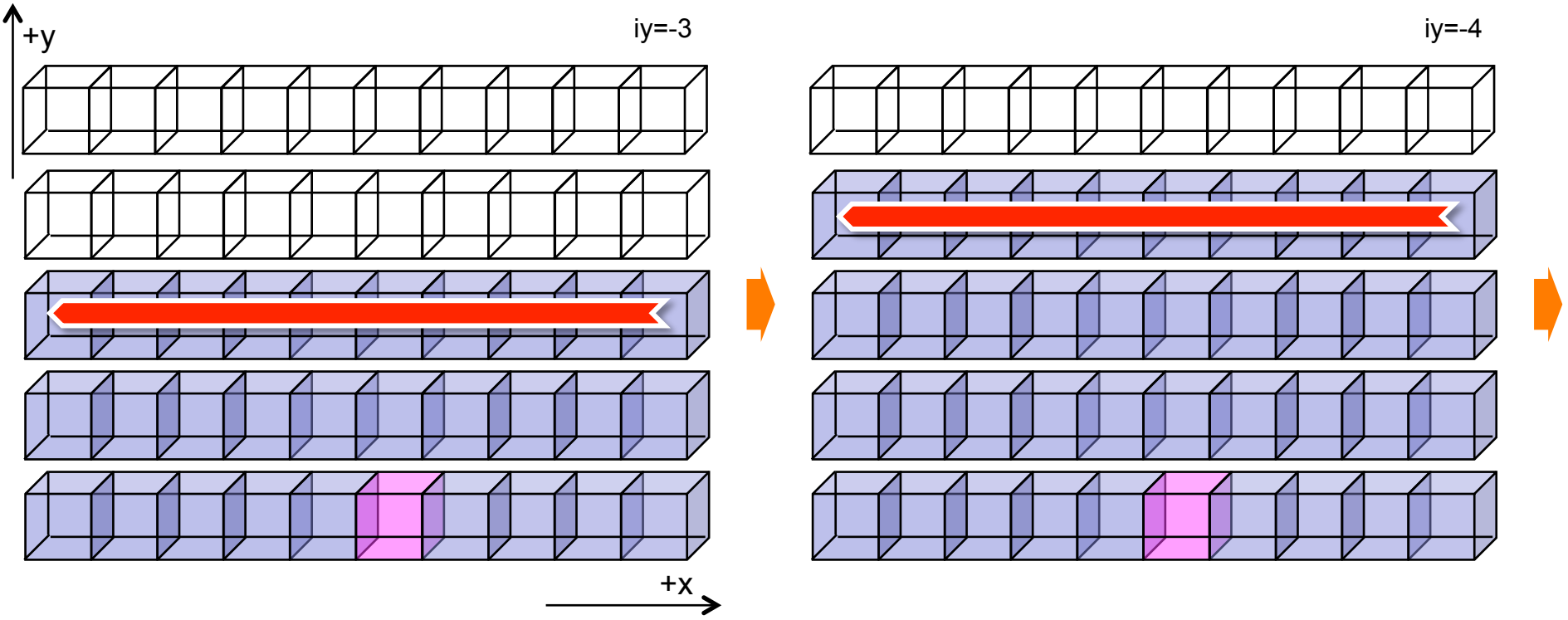


MPI 並列化技術: 通信衝突の回避

y方向通信コーディングイメージ

```
ipy_dest =送信先プロセス番号(-y)
ipy_src   =受信元プロセス番号(-y)
do iy= -1, -4, -1
   call mpi_sendrecv(..,ipy_dest,..,ipy_src,..)
enddo
```

袖部つき局所化されたデータ構造の所定位置に棒状の受信データを格納.　次回通信では前回の棒状の受信データを送受信.

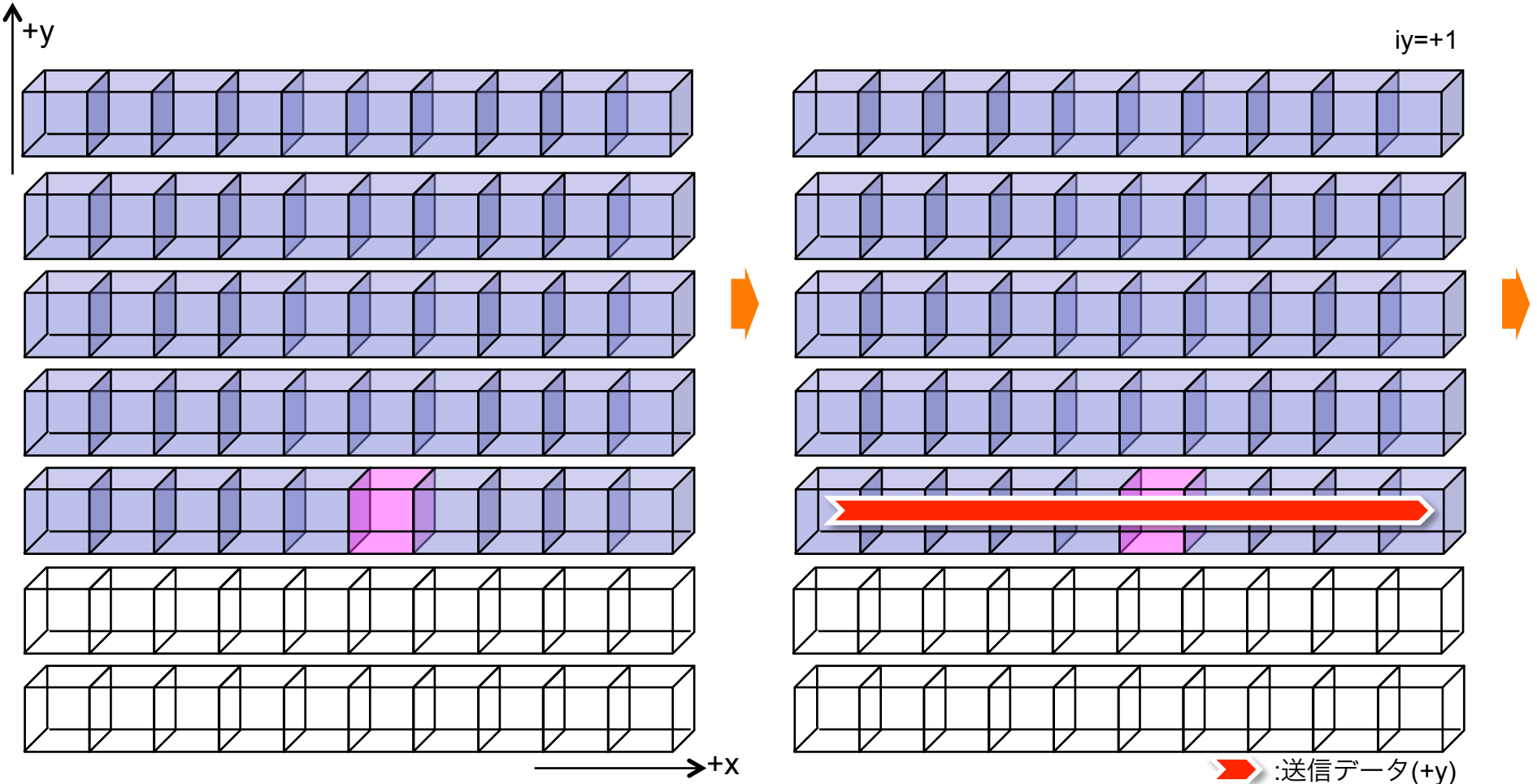


MPI 並列化技術: 通信衝突の回避

y方向通信コーディングイメージ

```
ipy_dest =送信先プロセス番号(-y)
ipy_src   =受信元プロセス番号(-y)
do iy= -1, -4, -1
   call mpi_sendrecv(..,ipy_dest,..,ipy_src,..)
enddo
```

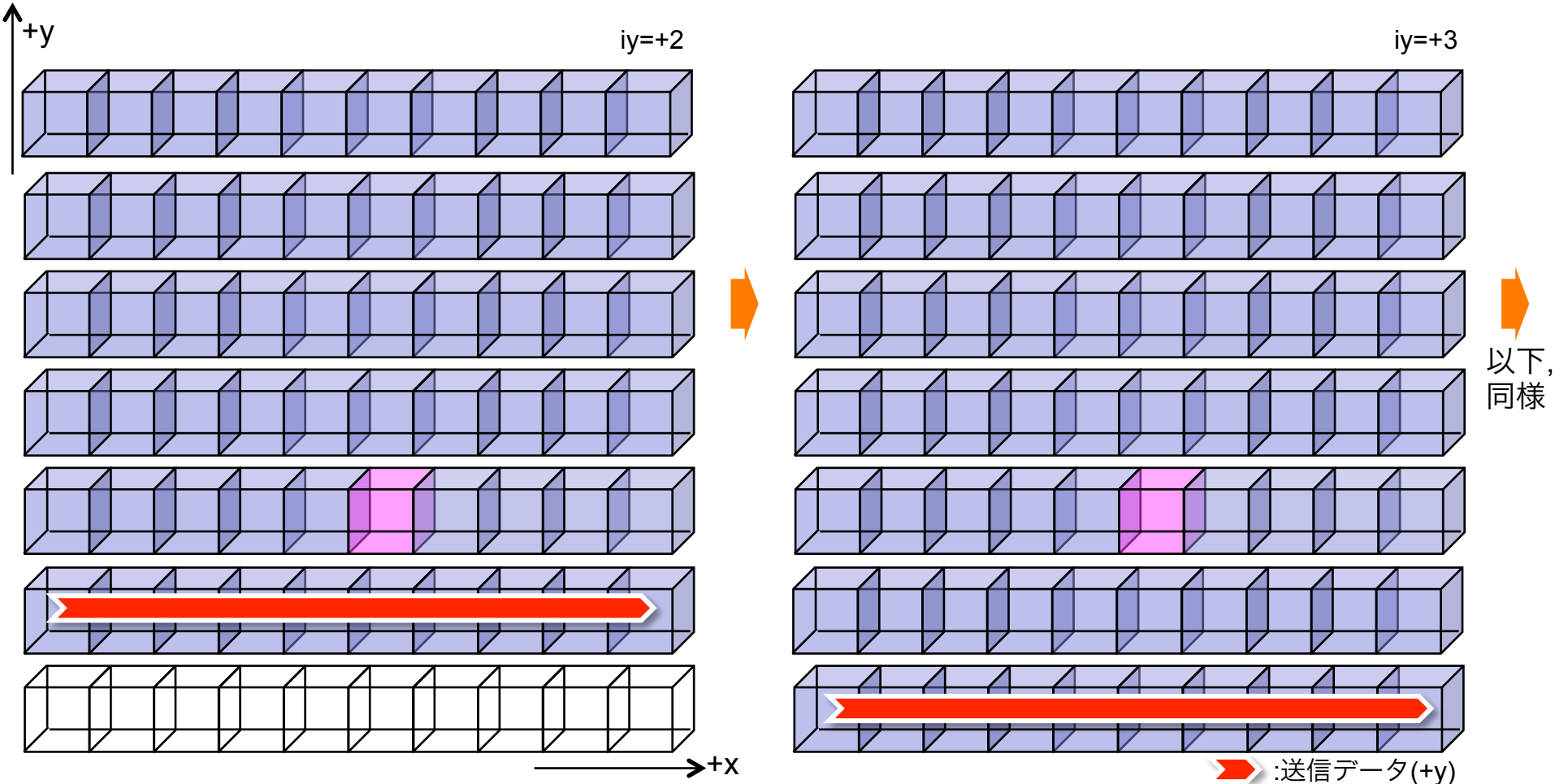
```
ipy_dest =送信先プロセス番号(+y)
ipy_src   =受信元プロセス番号(+y)
do iy= +1, +5, +1
   call mpi_sendrecv(..,ipy_dest,..,ipy_src,..)
enddo
```



MPI 並列化技術: 通信衝突の回避

y方向通信コーディングイメージ

```
ipy_dest =送信先プロセス番号(+y)
ipy_src   =受信元プロセス番号(+y)
do iy= +1, +5, +1
   call mpi_sendrecv(..,ipy_dest,..,ipy_src,..)
enddo
```



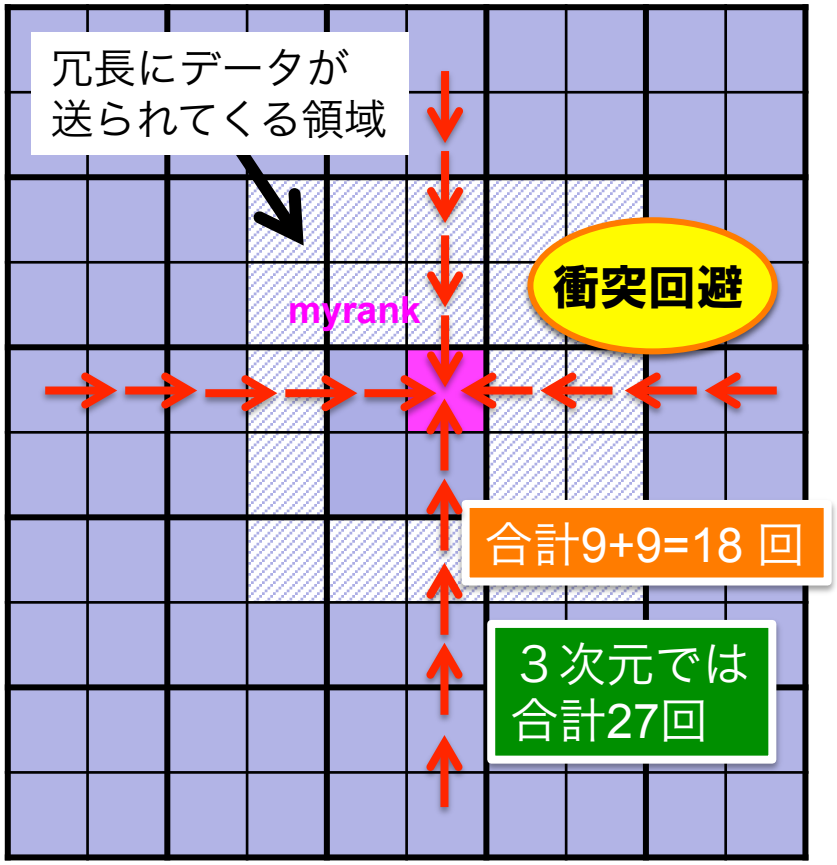
MPI 並列化技術: 通信衝突の回避

コーディングイメージ

```
ipx_dest =送信先プロセス番号(-x)
ipx_src  =受信元プロセス番号(-x)        } 固定
do ix= -1, -4, -1
   call mpi_sendrecv(..,ipx_dest,..,ipx_src,..)
enddo

ipx_dest =送信先プロセス番号(+x)
ipx_src  =受信元プロセス番号(+x)        } 固定
do ix= +1, +5, +1
   call mpi_sendrecv(..,ipx_dest,..,ipx_src,..)
enddo

ipy_dest =送信先プロセス番号(-y)
ipy_src  =受信元プロセス番号(-y)        } 固定
do iy=-1, -4, -1
   call mpi_sendrecv(..,ipy_dest,..,ipy_src,..)
enddo

ipy_dest =送信先プロセス番号(+y)
ipy_src  =受信元プロセス番号(+y)        } 固定
do iy=+1, +5, +1
   call mpi_sendrecv(..,ipy_dest,..,ipy_src,..)
enddo
```

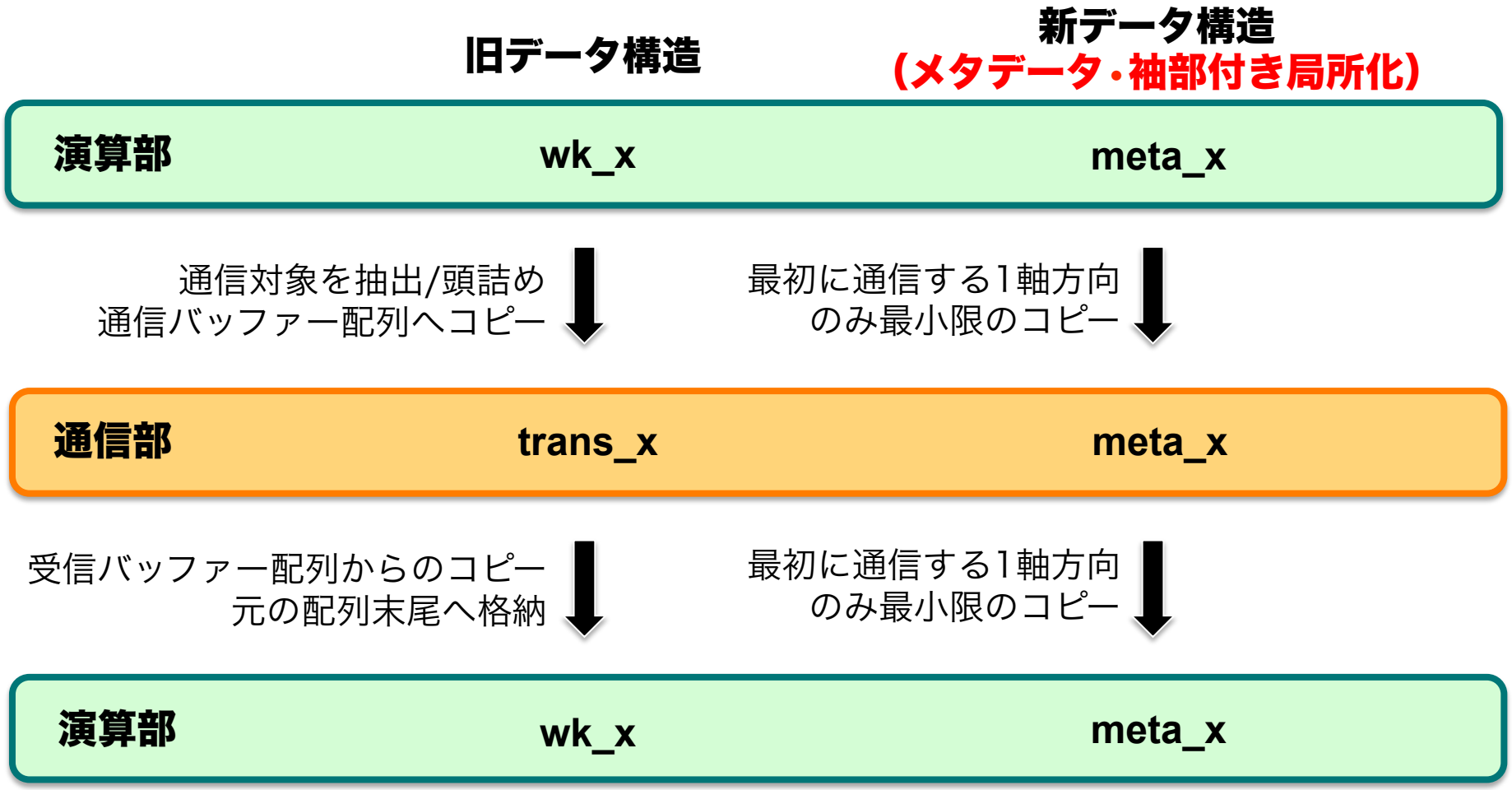
残るz方向も同様

新しい通信コード



３軸逐次隣接通信による局所的収集処理

MPI 並列化技術:通信前後での配列間コピーの消去



MPI 並列化技術:通信前後での配列間コピーの消去

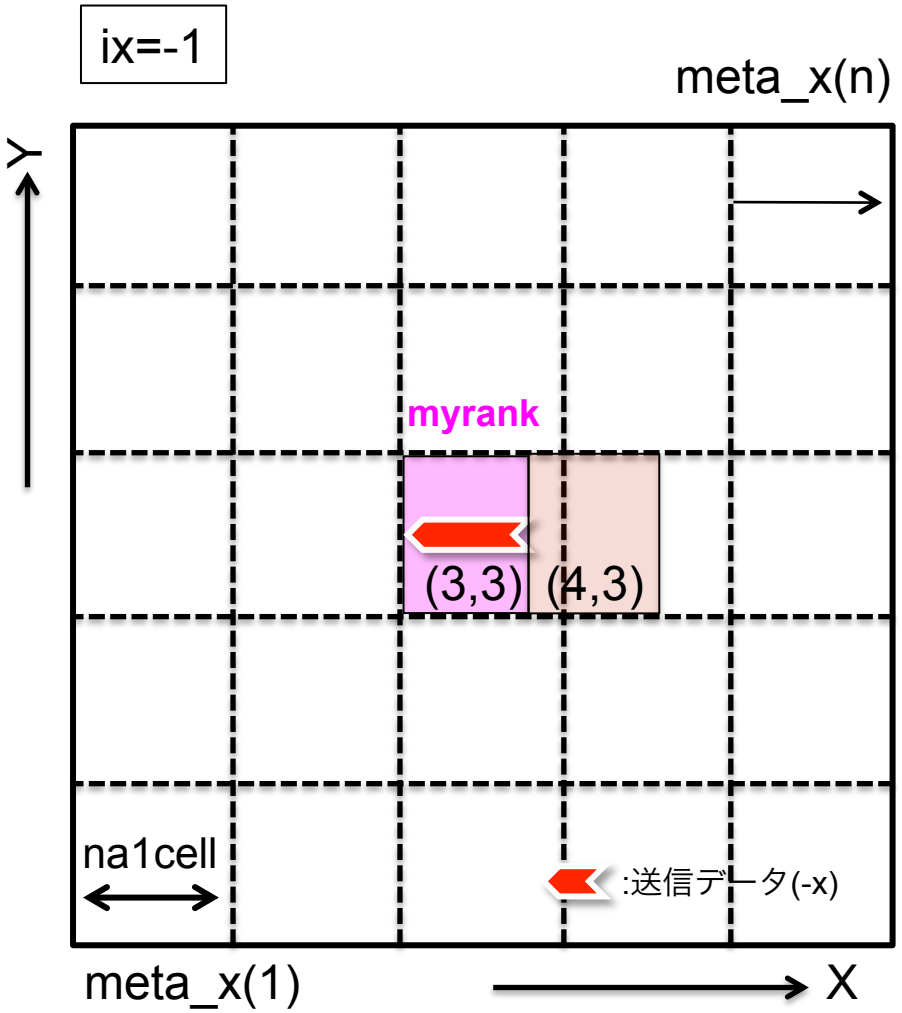
コーディングイメージ 最初の-x 軸方向通信 [**サイズ不定**の**セル単位データ**を送受信]

```
ipx_mdest  =送信先プロセス番号(-x)
ipx_msrc=受信元プロセス番号(-x)
do ix=-1,-2,-1
c 原子数情報の送受信
    icbufm( ) = na_per_cell( 3, 3 )
    call mpi_sendrecv(icbufm, ...,ipx_mdest, ...,
                      ircbufm,..., ipx_msrc, ...)
    na_per_cell( 4, 3 )=ircbufm( )
    受信データのtag(4,3)を作成
c 座標情報の送受信
    buffm( ) = meta_x( )
    call mpi_sendrecv(buffm,..,ipx_mdest,..,
                      rbuffm, ...,ipx_msrc,..)
    meta_x( ) = rbuffm( )
enddo
```

送信バッファーへコピー

データを左づめにしつつ元配列にコピー

注) myrank からみると -x 方向通信により +x 方向の座標情報を受信



MPI 並列化技術:通信前後での配列間コピーの消去

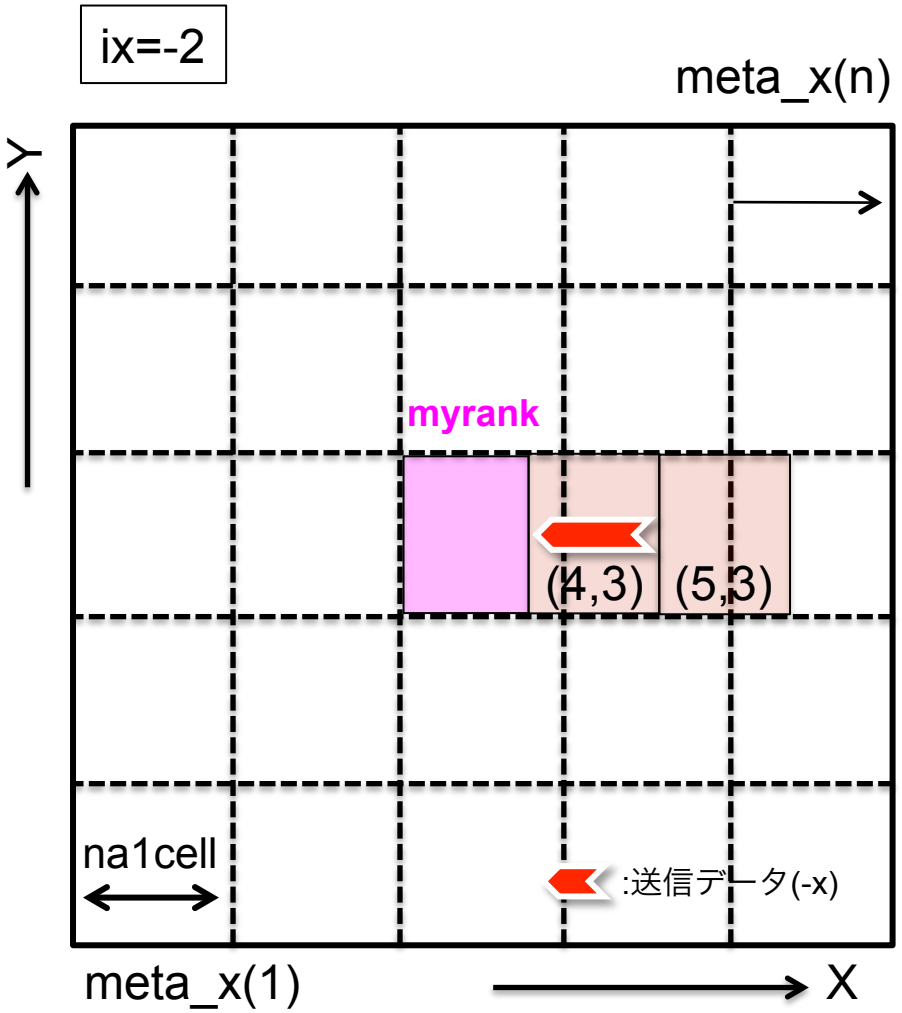
コーディングイメージ　最初の-x 軸方向通信 [**サイズ不定**の**セル単位データ**を送受信]

```
ipx_mdest  =送信先プロセス番号(-x)
ipx_msrc=受信元プロセス番号(-x)
do ix=-1,-2,-1
c 原子数情報の送受信
   icbufm( ) = na_per_cell( 4, 3 )
   call mpi_sendrecv(icbufm, ...,ipx_mdest, ...,
                              ircbufm,..., ipx_msrc, ...)
   na_per_cell( 5, 3 )=ircbufm( )
   受信データのtag(5,3)を作成
c 座標情報の送受信
   buffm( ) = meta_x( )
   call mpi_sendrecv(buffm,..,ipx_mdest,..,
                              rbuffm, ...,ipx_msrc,..)
   meta_x( ) = rbuffm( )
enddo
```

送信バッファーへコピー

データを左づめにしつつ元配列にコピー

注) myrank からみると -x 方向通信により +x 方向の座標情報を受信



MPI 並列化技術:通信前後での配列間コピーの消去

コーディングイメージ

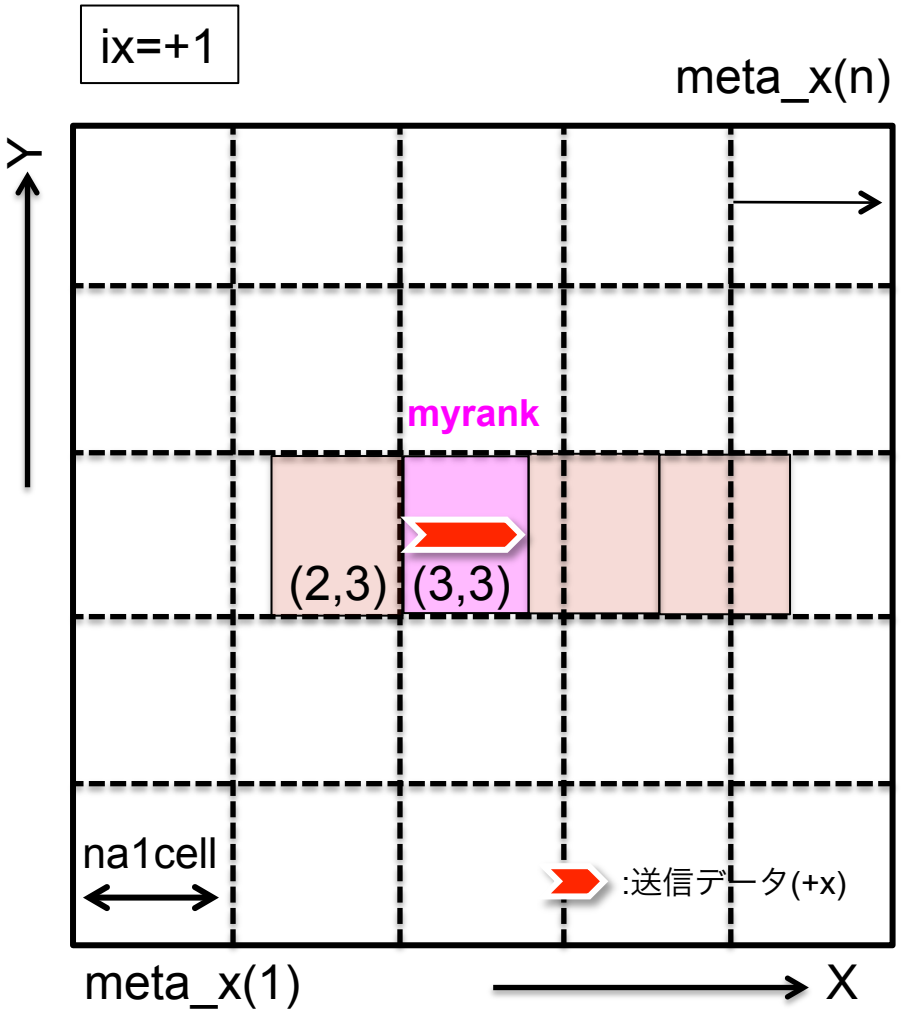
+x 軸方向通信　[**サイズ不定**の**セル単位データ**を送受信]

```
ipx_pdest   =送信先プロセス番号(+x)
ipx_psrc =受信元プロセス番号(+x)
do ix=+1,+2,+1
c 原子数情報の送受信
    icbufp( ) = na_per_cell( 3, 3 )
    call mpi_sendrecv(icbufp, ...,ipx_pdest, ...,
                                ircbufp,..., ipx_psrc, ...)
    na_per_cell( 2, 3 )=ircbufp( )
    受信データのtag(2,3)を作成
c 座標情報の送受信
    buffp( ) = meta_x( )
    call mpi_sendrecv(buffp,..,ipx_pdest,..,
                                rbuffp, ...,ipx_psrc,..)
    meta_x( ) = rbuffp( )
enddo
```

送信バッファーへコピー

データを右づめにしつつ元配列にコピー

注) myrank からみると +x 方向通信により –x 方向の座標情報を受信



MPI 並列化技術:通信前後での配列間コピーの消去

コーディングイメージ

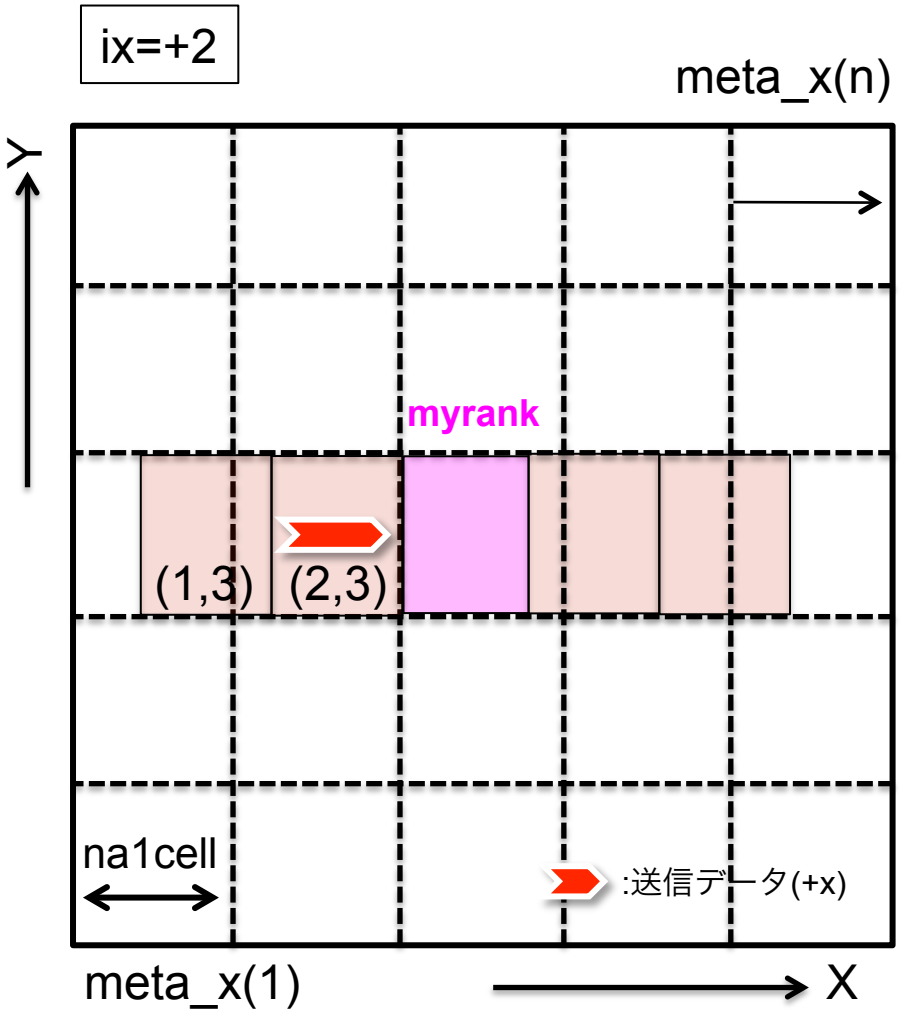
+x 軸方向通信　[**サイズ不定**の**セル単位データ**を送受信]

```
ipx_pdest   =送信先プロセス番号(+x)
ipx_psrc =受信元プロセス番号(+x)
do ix=+1,+2,+1
c 原子数情報の送受信
    icbufp( ) = na_per_cell( 2, 3 )
    call mpi_sendrecv(icbufp, ...,ipx_pdest, ...,
                                ircbufp,..., ipx_psrc, ...)
    na_per_cell( 1, 3 )=ircbufp( )
    受信データのtag(1,3)を作成
c 座標情報の送受信
    buffp( ) = meta_x( )
    call mpi_sendrecv(buffp,..,ipx_pdest,..,
                                rbuffp, ...,ipx_psrc,..)
    meta_x( ) = rbuffp( )
enddo
```

送信バッファーへコピー

データを右づめにしつつ元配列にコピー

注) myrank からみると +x 方向通信により –x 方向の座標情報を受信



MPI 並列化技術:通信前後での配列間コピーの消去

コーディングイメージ

つづく -y 軸方向通信　[**サイズ一定**の**棒状データ**を送受信]

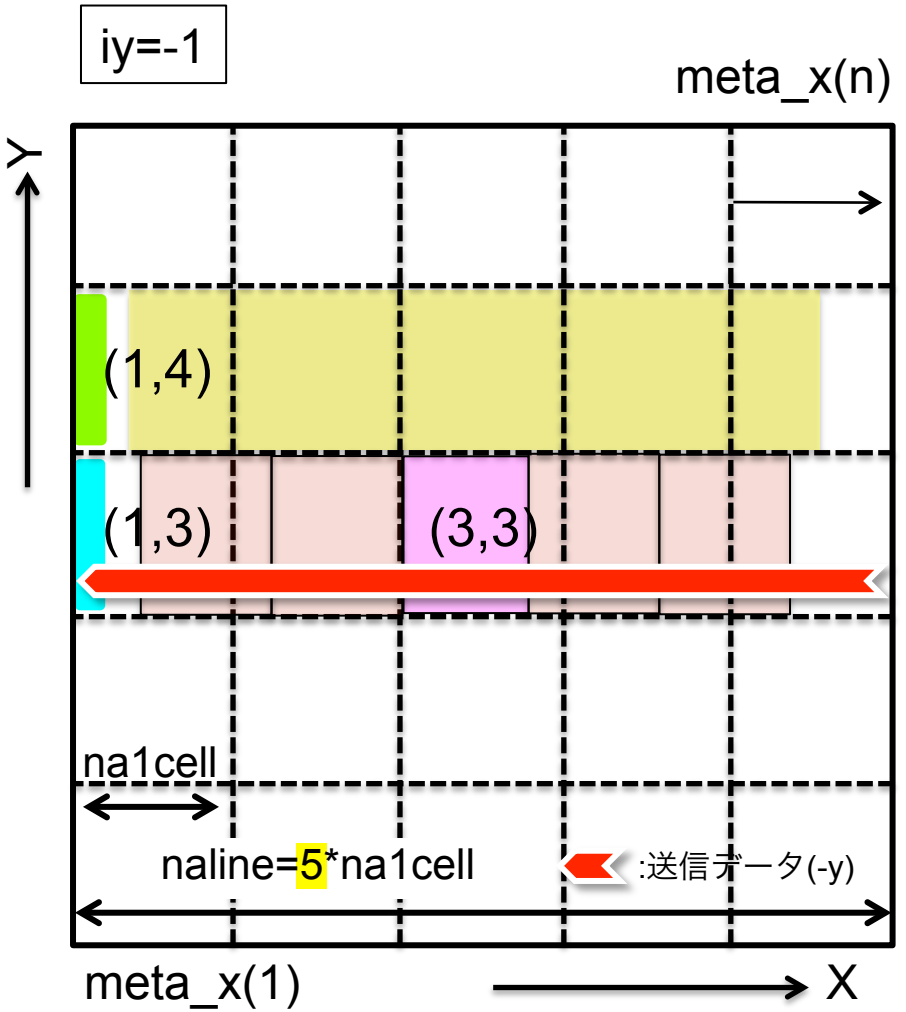
```
ipy_mdest  =送信先プロセス番号(-y)
ipy_msrc=受信元プロセス番号(-y)
do iy=-1,-2,-1
c 原子数情報の送受信

   call mpi_sendrecv(na_per_cell(1,3),5,...,ipy_mdest, ...,
                     na_per_cell(1,4),5,..., ipy_msrc, ...)
   受信データのtag(1:5,4)を作成
c 座標情報の送受信

   call mpi_sendrecv(meta_x(1,3),naline,..,ipy_mdest,..,
                     meta_x(1,4),naline,...,ipy_msrc,..)

enddo
```

棒状データを構成する
サブセル数を指定

(1,3)セルの先頭
アドレスを指定

(1,4)セルの先頭
アドレスを指定

注) myrank からみると -y 方向通信により
+y 方向の座標情報を受信



MPI 並列化技術:通信前後での配列間コピーの消去

コーディングイメージ

つづく -y 軸方向通信　[**サイズ一定**の**棒状データ**を送受信]

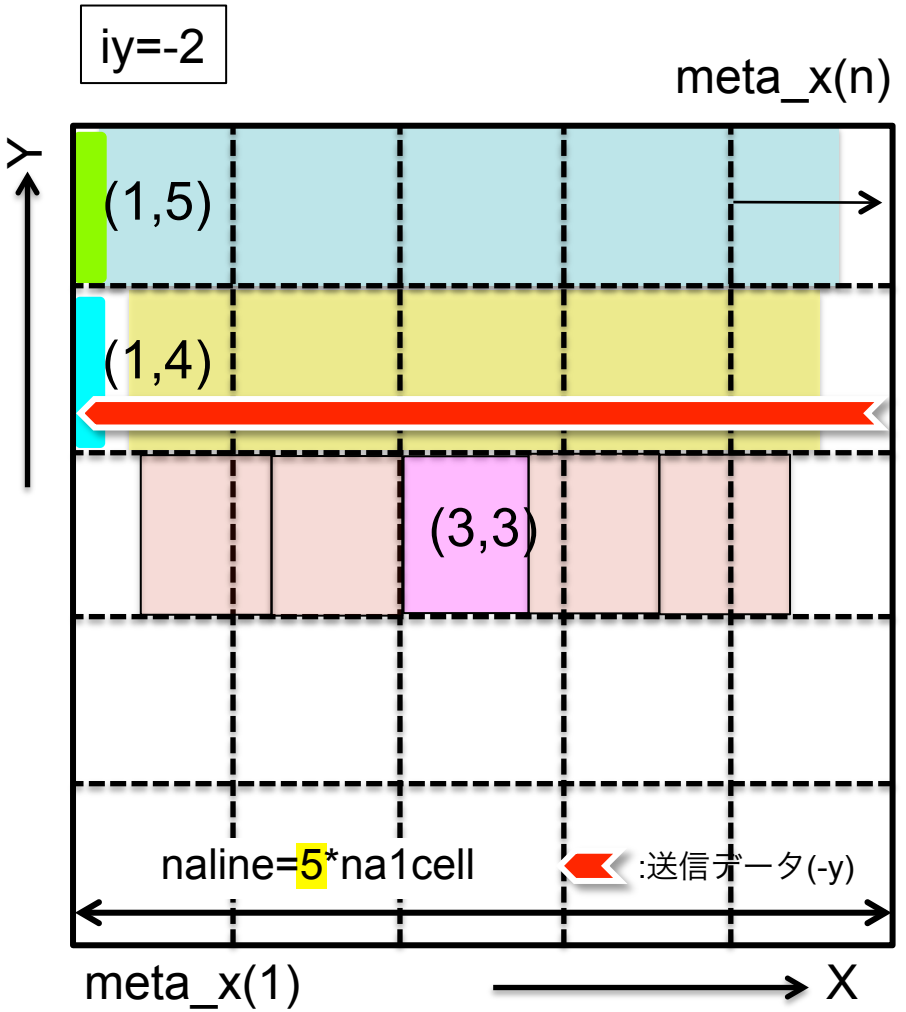
```
ipy_mdest  =送信先プロセス番号(-y)
ipy_msrc=受信元プロセス番号(-y)
do iy=-1,-2,-1
c 原子数情報の送受信

   call mpi_sendrecv(na_per_cell(1,4),5,...,ipy_mdest, ...,
                     na_per_cell(1,5),5,..., ipy_msrc, ...)
   受信データのtag(1:5,5)を作成
c 座標情報の送受信

   call mpi_sendrecv(meta_x(1,4),naline,..,ipy_mdest,..,
                     meta_x(1,5),naline,...,ipy_msrc,..)

enddo
```

棒状データを構成する
サブセル数を指定

(1,4)セルの先頭
アドレスを指定

(1,5)セルの先頭
アドレスを指定

注) myrank からみると -y 方向通信により
+y 方向の座標情報を受信



MPI 並列化技術:通信前後での配列間コピーの消去

コーディングイメージ

+y 軸方向通信

[サイズ一定の棒状データを送受信]

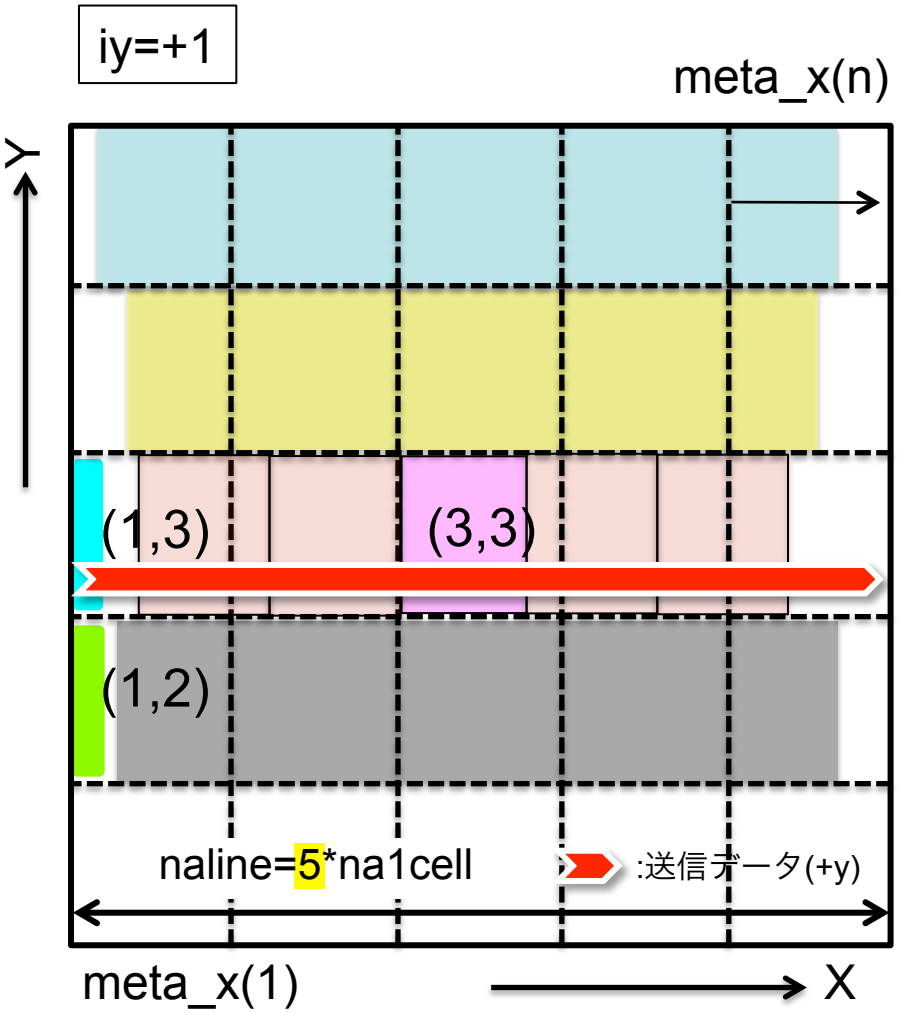
```
ipy_pdest   =送信先プロセス番号(+y)
ipy_psrc =受信元プロセス番号(+y)
do iy=+1,+2,+1
c 原子数情報の送受信

   call mpi_sendrecv(na_per_cell(1,3),5,...,ipy_pdest, ...,
                     na_per_cell(1,2),5,..., ipy_psrc, ...)
   受信データのtag(1:5,2)を作成
c 座標情報の送受信

   call mpi_sendrecv(meta_x(1,3),naline,..,ipy_pdest,..,
                     meta_x(1,2),naline,...,ipy_psrc,..)

enddo
```

棒状データを構成するサブセル数を指定

(1,3)セルの先頭アドレスを指定

(1,2)セルの先頭アドレスを指定

注) myrank からみると +y 方向通信により
-y 方向の座標情報を受信



MPI 並列化技術:通信前後での配列間コピーの消去

コーディングイメージ

+y 軸方向通信

[サイズ一定の棒状データを送受信]

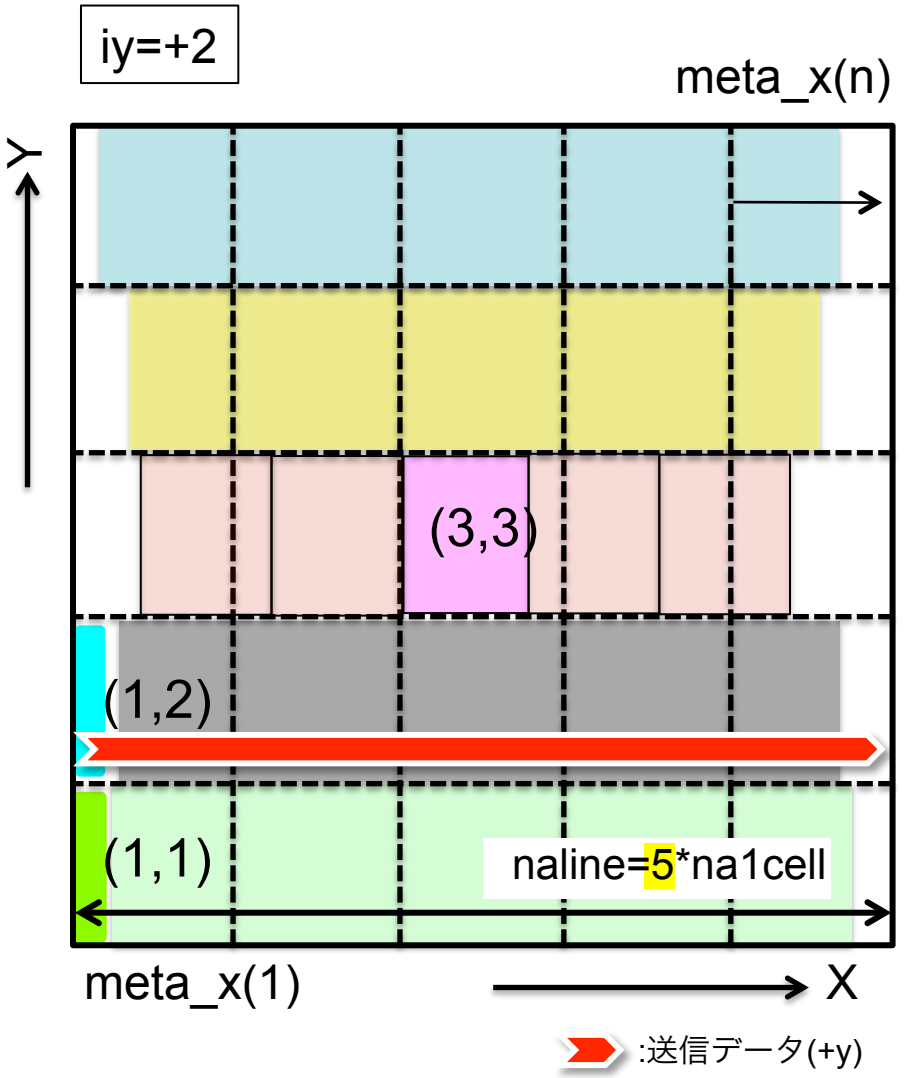
```
ipy_pdest   =送信先プロセス番号(+y)
ipy_psrc =受信元プロセス番号(+y)
do iy=+1,+2,+1
c 原子数情報の送受信

   call mpi_sendrecv(na_per_cell(1,2),5,...,ipy_pdest, ...,
                     na_per_cell(1,1),5,..., ipy_psrc, ...)
   受信データのtag(1:5,1)を作成
c 座標情報の送受信

   call mpi_sendrecv(meta_x(1,2),naline,..,ipy_pdest,..,
                     meta_x(1,1),naline,...,ipy_psrc,..)

enddo
```

棒状データを構成するサブセル数を指定

(1,2)セルの先頭アドレスを指定

(1,1)セルの先頭アドレスを指定

注) myrank からみると +y 方向通信により
-y 方向の座標情報を受信



MPI 並列化技術:通信前後での配列間コピーの消去

コーディングイメージ

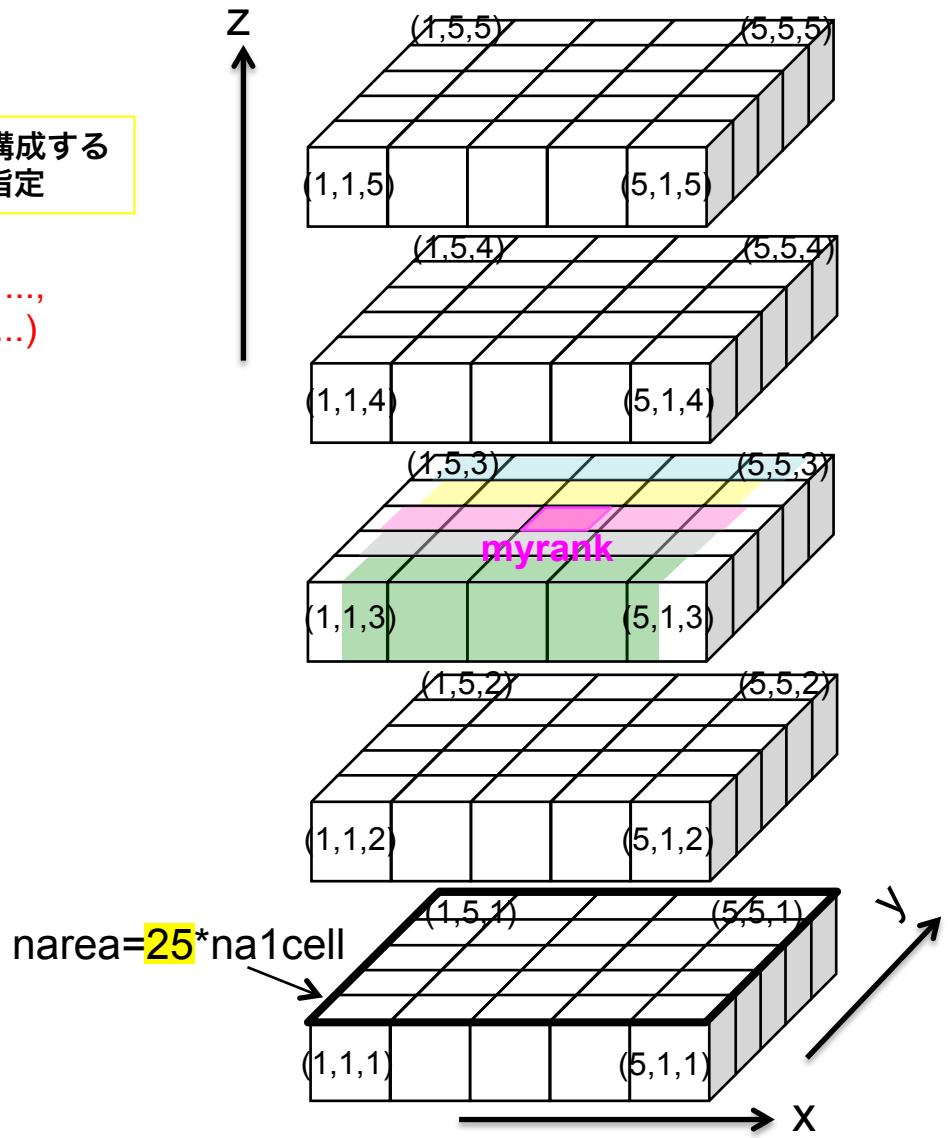
-z 軸方向通信

[**サイズー定**の**面状データ**を送受信]

```
ipz_mdest  =送信先プロセス番号(-z)
ipz_msrc=受信元プロセス番号(-z)
do iz=-1,-2,-1
c 原子数情報の送受信

   call mpi_sendrecv(na_per_cell( ),25,...,ipz_mdest, ...,
                     na_per_cell( ),25,..., ipz_msrc, ...)
   受信データのtagを作成
c 座標情報の送受信

   call mpi_sendrecv(meta_x( ),narea,..,ipz_mdest,..,
                     meta_x( ),narea,...,ipz_msrc,..)

enddo
```

面状データを構成する
サブセル数を指定

注) myrank からみると -z 方向通信により
+z 方向の座標情報を受信



MPI 並列化技術:通信前後での配列間コピーの消去

コーディングイメージ

-z 軸方向通信

[サイズ一定の面状データを送受信]

ipz_mdest =送信先プロセス番号(-z)
ipz_msrc=受信元プロセス番号(-z)
do iz=-1,-2,-1
c 原子数情報の送受信

面状データを構成する
サブセル数を指定

```
call mpi_sendrecv(na_per_cell(1,1,3),25,...,ipz_mdest, ...,
                  na_per_cell(1,1,4),25,..., ipz_msrc, ...)
```

受信データのtag(1:5,1:5,4)を作成

c 座標情報の送受信

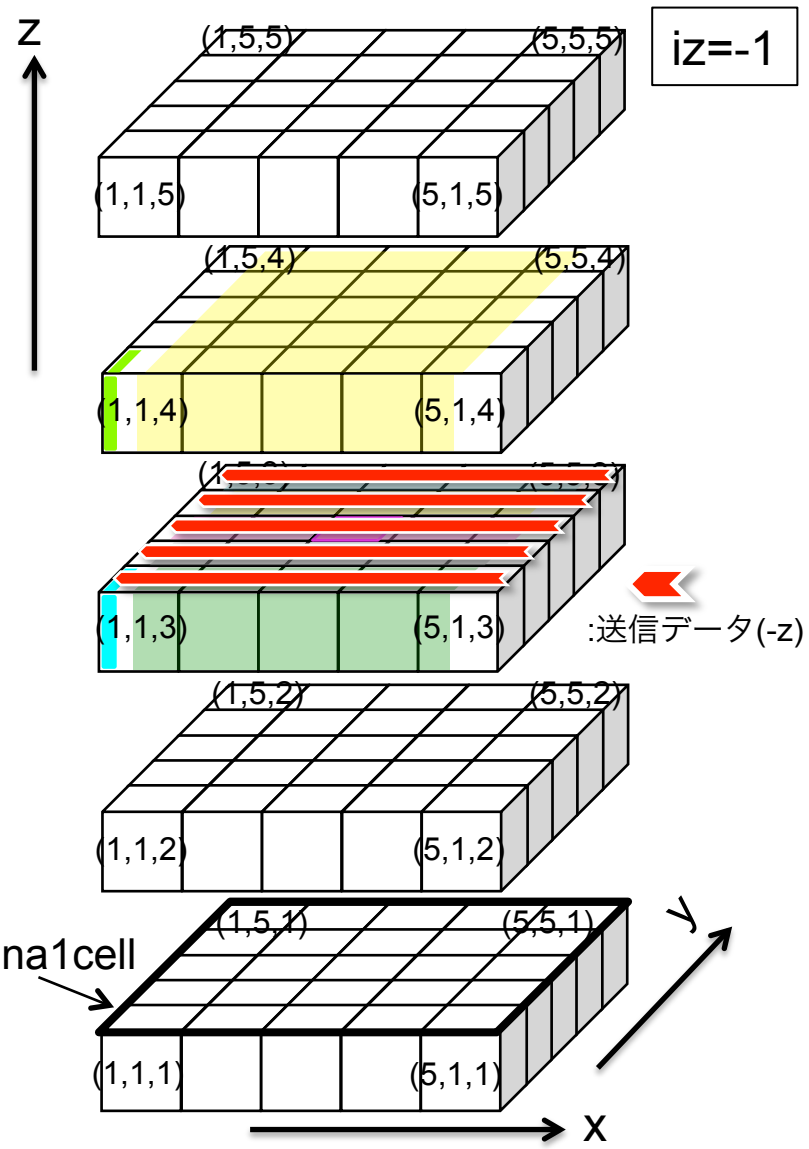
(1,1,3)セルの先頭
アドレスを指定

```
call mpi_sendrecv(meta_x(),narea,..,ipz_mdest,..,
                  meta_x(),narea,...,ipz_msrc,..)
```

(1,1,4)セルの先頭
アドレスを指定

enddo

注) myrank からみると -z 方向通信により
+z 方向の座標情報を受信



MPI 並列化技術:通信前後での配列間コピーの消去

コーディングイメージ

-z 軸方向通信

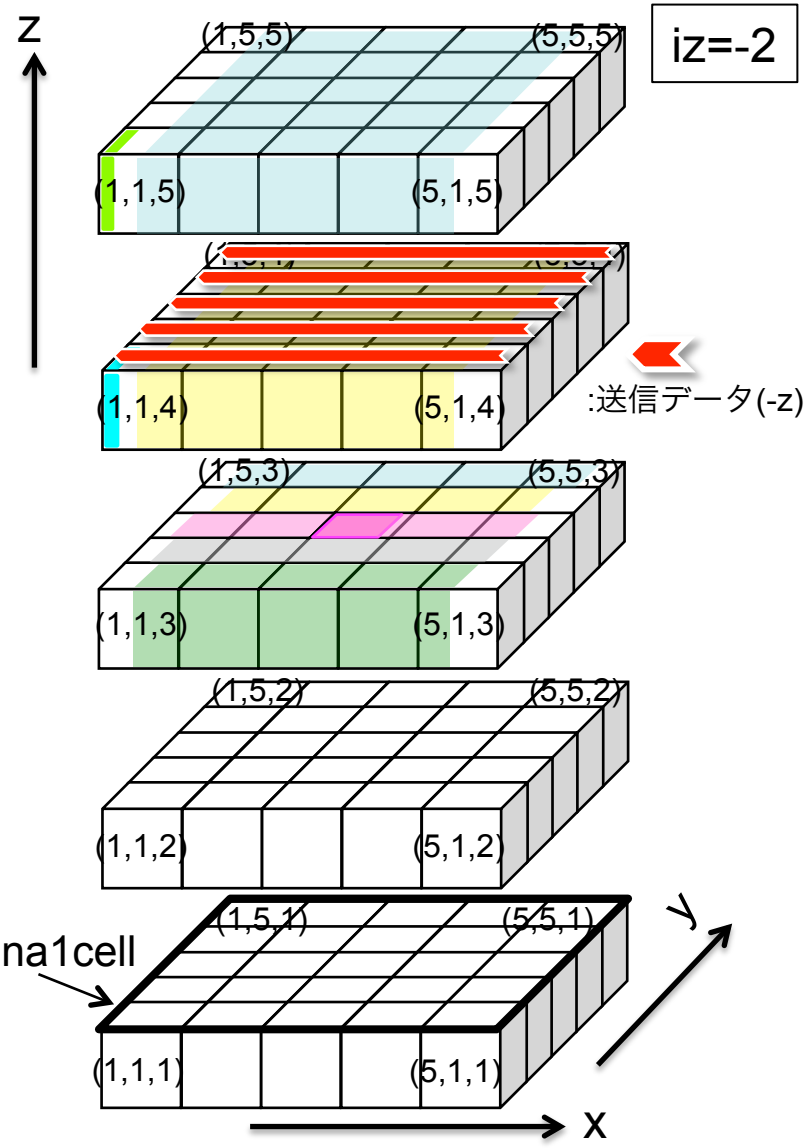
[サイズ一定の面状データを送受信]

```
ipz_mdest  =送信先プロセス番号(-z)
ipz_msrc=受信元プロセス番号(-z)
do iz=-1,-2,-1
c 原子数情報の送受信

   call mpi_sendrecv(na_per_cell(1,1,4),25,...,ipz_mdest, ...,
                     na_per_cell(1,1,5),25,..., ipz_msrc, ...)
   受信データのtag(1:5,1:5,5)を作成
c 座標情報の送受信

   call mpi_sendrecv(meta_x(),narea,..,ipz_mdest,..,
                     meta_x(),narea,...,ipz_msrc,..)

enddo
```

面状データを構成するサブセル数を指定

(1,1,4)セルの先頭アドレスを指定

(1,1,5)セルの先頭アドレスを指定



注) myrank からみると -z 方向通信により
+z 方向の座標情報を受信

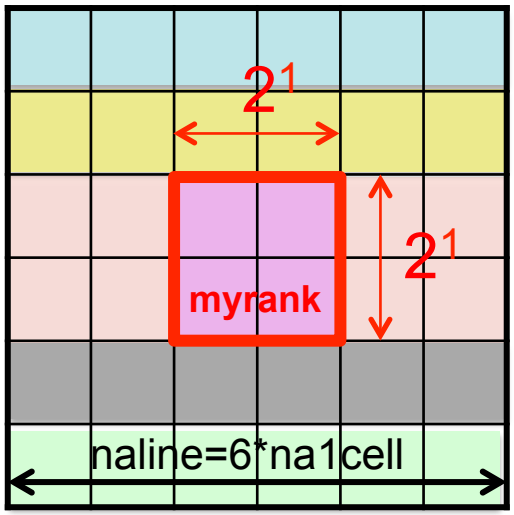
最後の+z 軸方向通信も同様
→ すべての袖部のデータが受信される

MPI 並列化技術:通信前後での配列間コピーの消去

汎用性のため実際のコードでは以下の点にも対応.

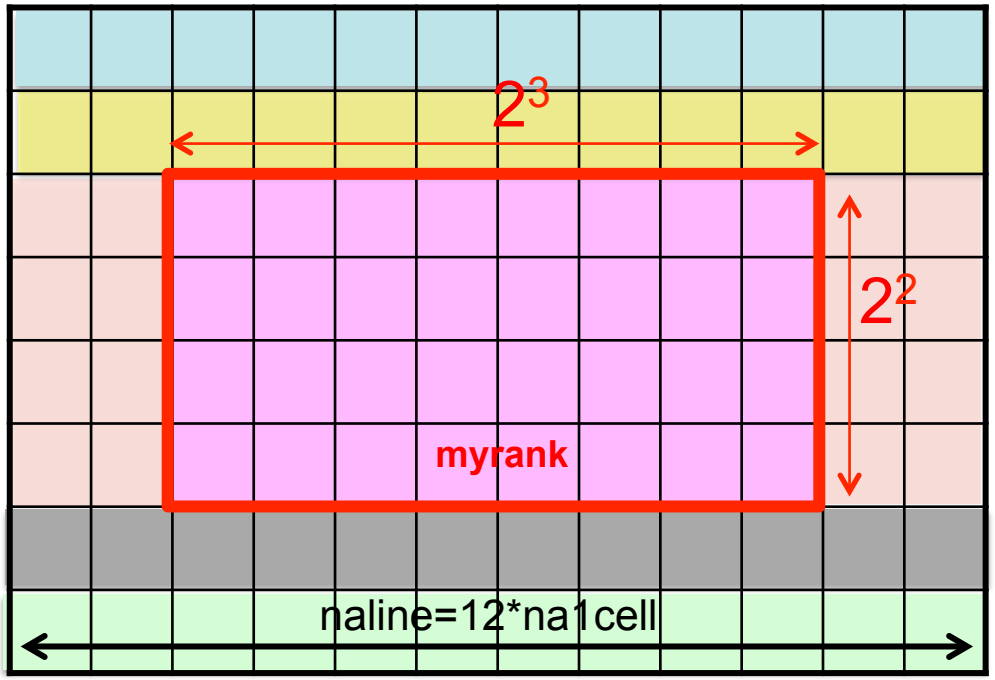
- 任意のサブセルブロック厚さ2^n (n≥0)
- 直方体形状のサブセルブロック

narea=36*na1cell




```
allocate( tag(1:6,1:6) )
allocate( na_per_cell(1:6,1:6) )
```

narea=96*na1cell




```
allocate( tag(1:12,1:8) )
allocate( na_per_cell(1:12,1:8) )
```

注) 開発経緯の都合上, 実際のコードでは z → y → x の順に通信しています.
comm_3.f, comm_fmm.f

MPI 並列化技術: 通信の演算による代用

代用の目的 (1): プログラム全体の計算時間の短縮

冗長な多重演算時間 < 単一演算結果の通信時間

代用

現在のスパコン構成では, しばしば生じる関係.

代用の目的 (2): デバッグのしやすいコードの作成

- **多重演算だが, 分かりやすいコード**
- **単一演算だが, 複雑なコード**

代用

上のコードが完成した上で, 下のコードを派生させる.
最初から下のコードを作ることは難易度高. (腕に覚えある人はどうぞ)

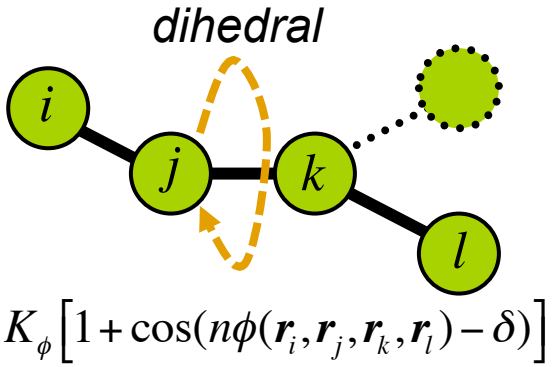
MPI 並列化技術: 通信の演算による代用

冗長な多重演算の例

- 分子内相互作用計算 [目的(2)]
- 分子間近距離相互作用計算 [目的(1),(2)]
- FMM 上位階層の M2M/L2L [目的(1)]

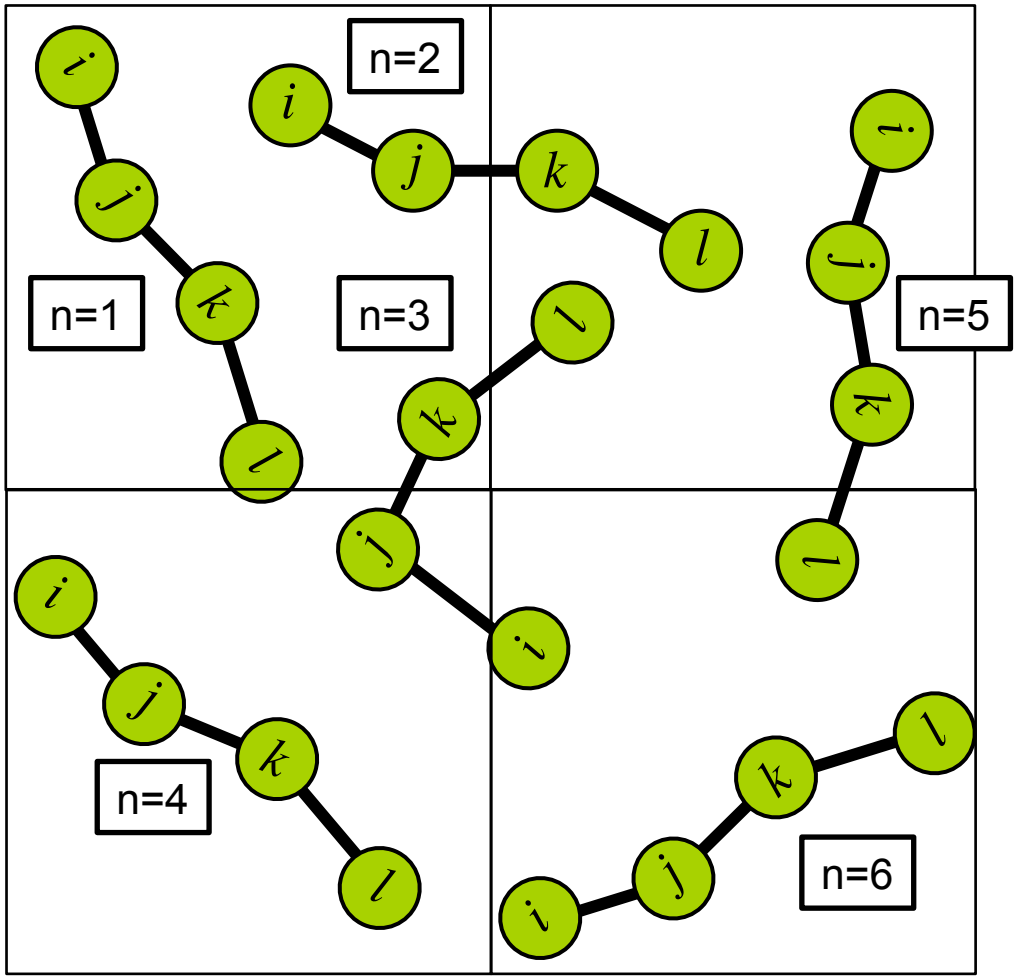
MPI 並列化技術: 通信の演算による代用

冗長な多重演算: 分子内相互作用計算



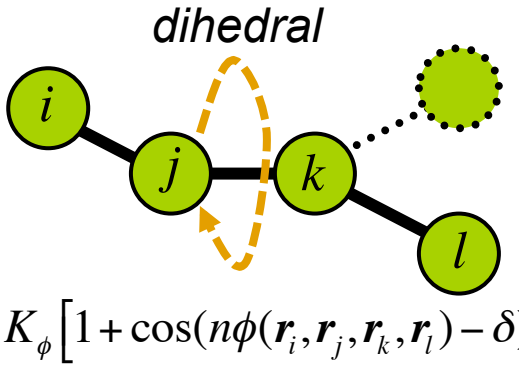
オリジナルコード

```
do n=1,ndihedrals
  phi=phi(ri, rj, rk, rl)
  ポテンシャルの計算
  Fi, Fj, Fk, Fl の計算
  f(i)=f(i)+Fi
  f(j)=f(j)+Fj
  f(k)=f(k)+Fk
  f(l)=f(l)+Fl
enddo
```



MPI 並列化技術: 通信の演算による代用

冗長な多重演算: 分子内相互作用計算

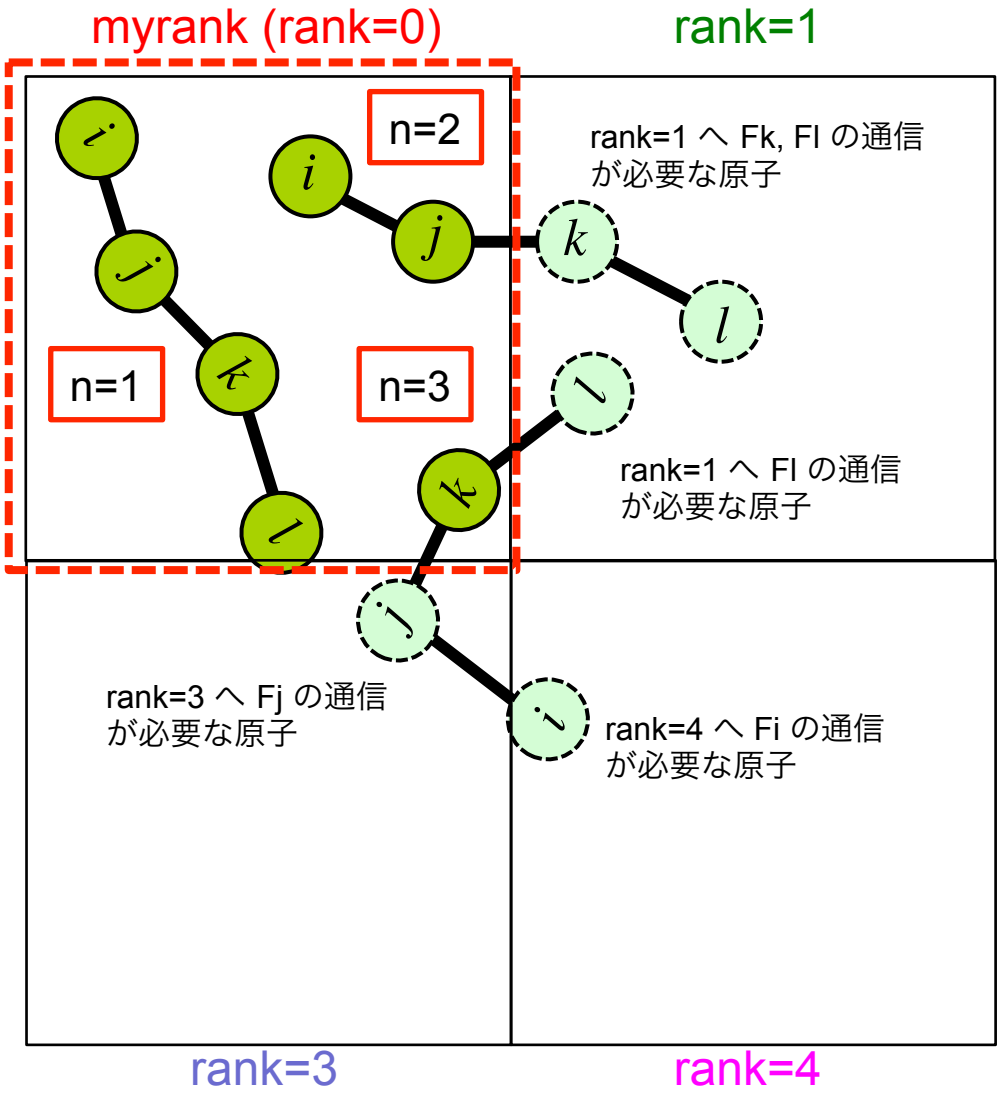


$$K_\phi\left[1+\cos(n\phi(\boldsymbol{r}_i,\boldsymbol{r}_j,\boldsymbol{r}_k,\boldsymbol{r}_l)-\delta)\right]$$

MPI並列化コード

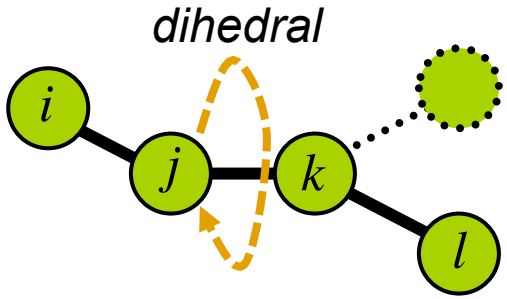
```
do n=1,ndihedrals(myrank)
  phi=phi(ri, rj, rk, rl)
  ポテンシャルの計算
  Fi, Fj, Fk, Fl の計算
  IF(ri in myrank)    f(i)=f(i)+Fi,  ELSE Fiを通信
  IF(rj in myrank)    f(j)=f(j)+Fj,  ELSE Fjを通信
  IF(rk in myrank) f(k)=f(k)+Fk, ELSE Fkを通信
  IF(rl  in myrank)   f(l)=f(l)+Fl,  ELSE Flを通信
enddo
```

＊ELSEの場合, 実際には合力をまとめて
該当プロセスへ通信します



MPI 並列化技術: 通信の演算による代用

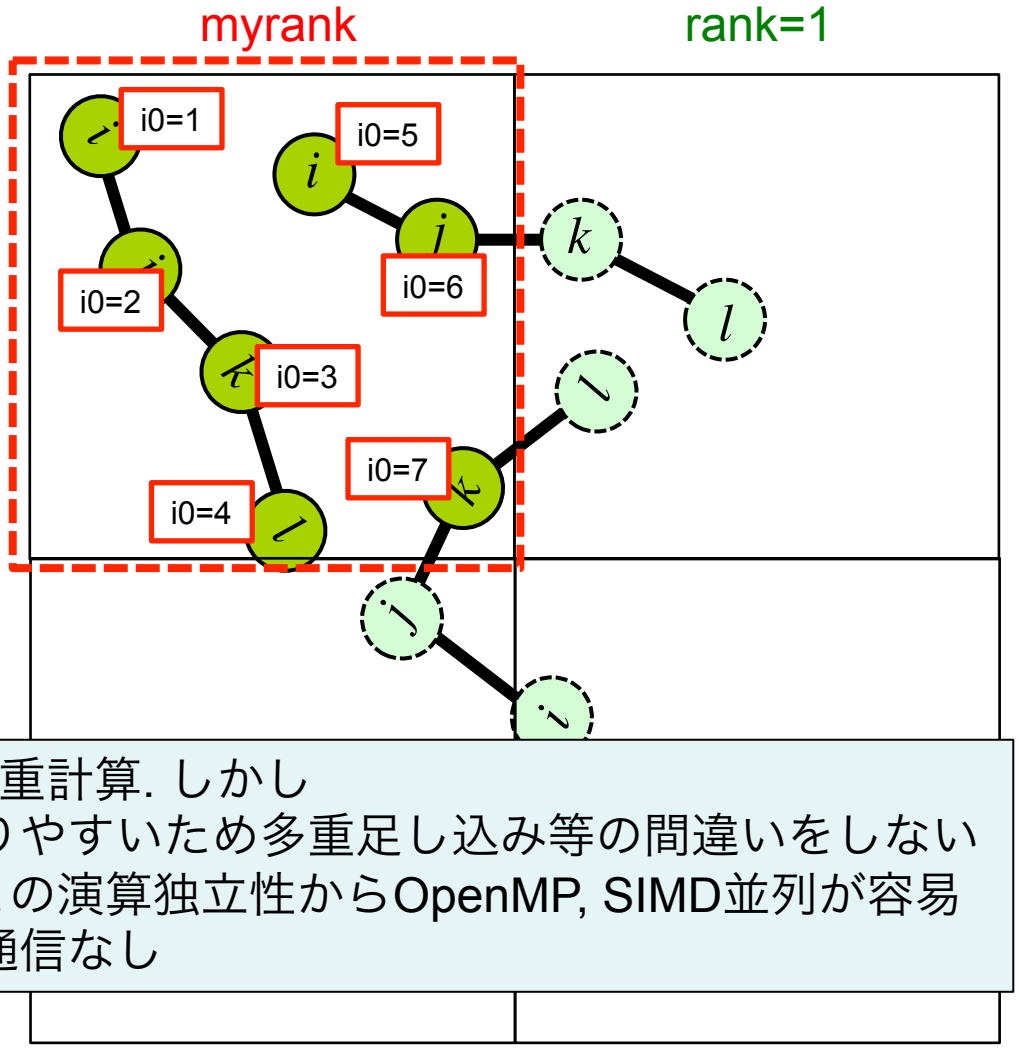
冗長な多重演算: 分子内相互作用計算



$$K_\phi\left[1+\cos(n\phi(\boldsymbol{r}_i,\boldsymbol{r}_j,\boldsymbol{r}_k,\boldsymbol{r}_l)-\delta)\right]$$

演算冗長MPI並列化コード

```
do i0=1,natom(myrank)
  dihedral番号を i0 より逆引き
  phi=phi(ri, rj, rk, rl)
  ポテンシャルの計算
  Fi の計算
  f(i0)=f(i0)+Fi
enddo
```



4重の多重計算. しかし
・分かりやすいため多重足し込み等の間違いをしない
・i0ごとの演算独立性からOpenMP, SIMD並列が容易
・力の通信なし

md_charmm_f90.f

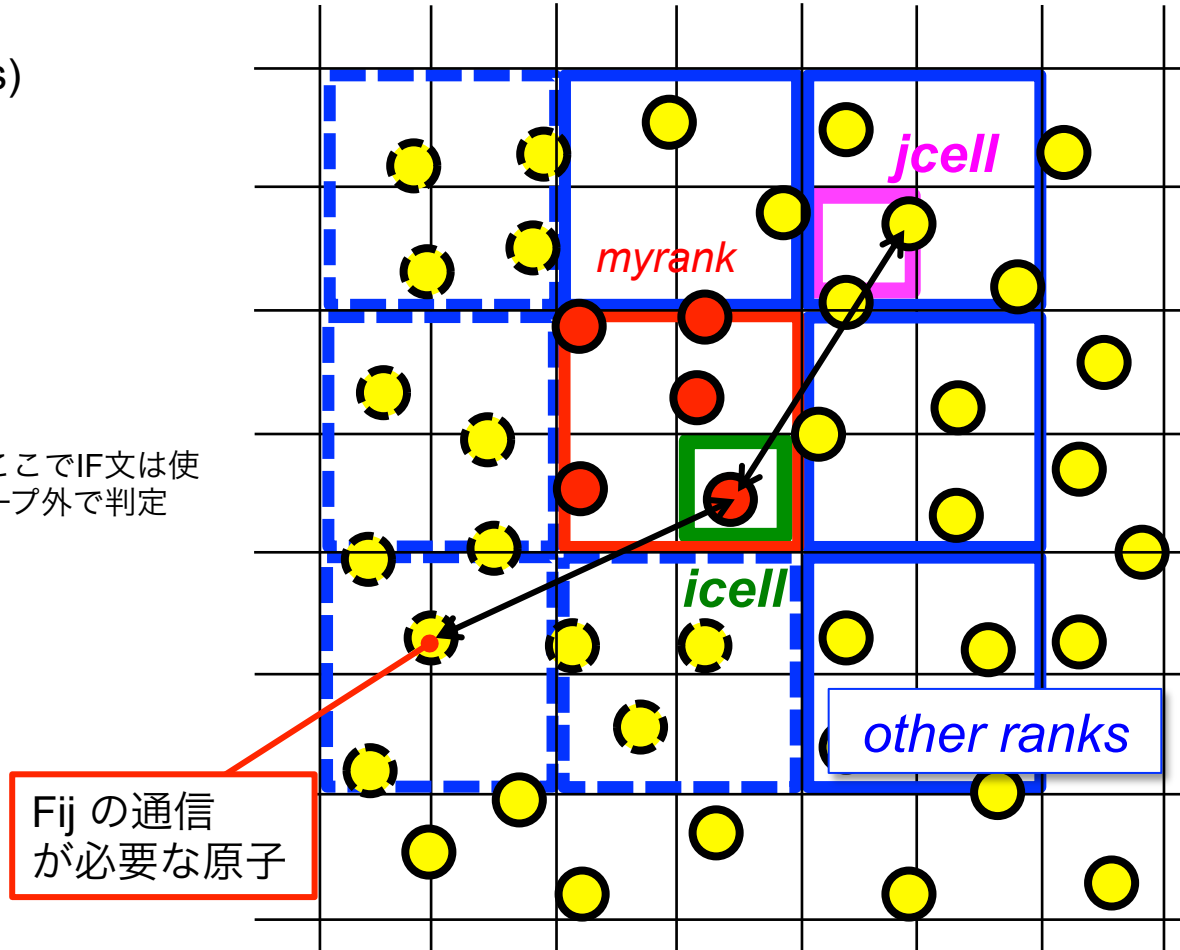
MPI 並列化技術: 通信の演算による代用

冗長な多重演算: 分子間近距離相互作用計算

二体力　$\boldsymbol{F}_{ij} = -\boldsymbol{F}_{ji}$　　例) Lennard-Jones, Coulombの粒子対計算

```
do icell(myrank)
do jcell_list(myrank or otherranks)
do i=na_per_cell(icell)
do j=na_per_cell(jcell)
  rij=rij(ri, rj)
  カットオフ判定
  ポテンシャルの計算
  Fij の計算
  f(i)=f(i)+Fij
  IF(rj in myrank) f(j)=f(j)-Fij
  ELSE storeF(j)=storeF(j)-Fij
enddo ! j
enddo ! i
enddo ! jcell
enddo ! icell
storeF(j)を対象プロセスへ通信
```

＊実際にはここでIF文は使わず, DOループ外で判定

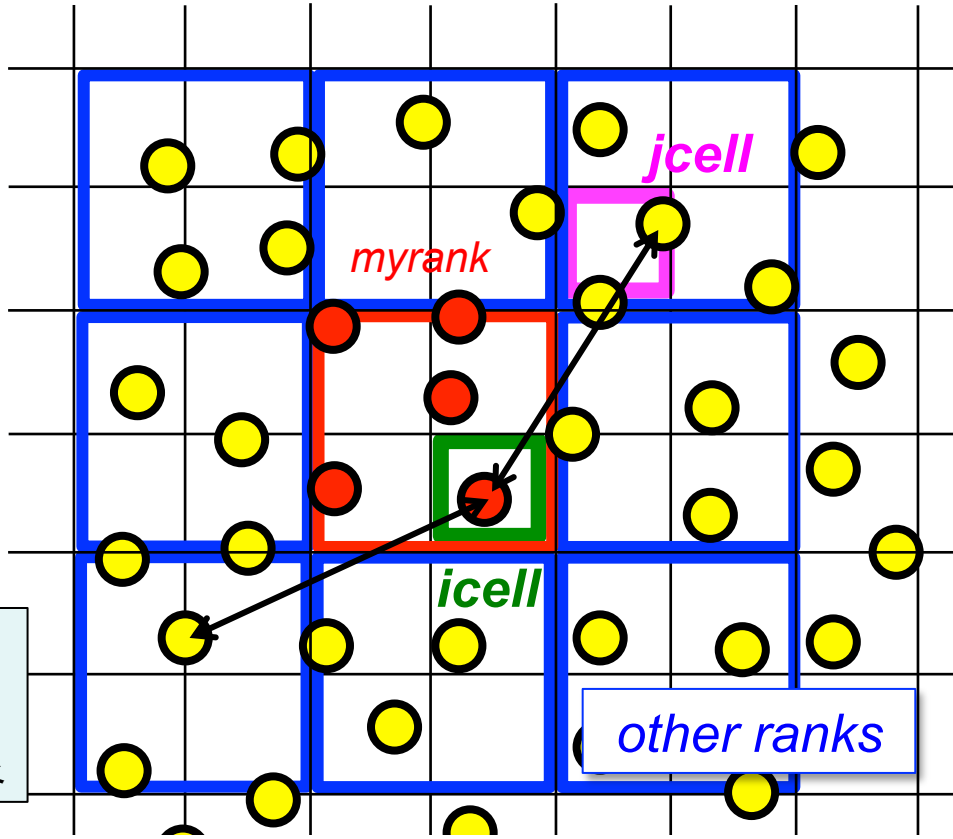


MPI 並列化技術: 通信の演算による代用

冗長な多重演算: 分子間近距離相互作用計算

二体力 $\boldsymbol{F}_{ij} = -\boldsymbol{F}_{ji}$ 例) Lennard-Jones, Coulombの粒子対計算

```
do icell(myrank)
do jcell_list(myrank or otherranks)
do i=na_per_cell(icell)
do j=na_per_cell(jcell)
  rij=rij(ri, rj)
  カットオフ判定
  ポテンシャルの計算
  Fij の計算
  f(i)=f(i)+Fij        ! i 原子のみ足し込み
enddo ; enddo ; enddo ; enddo
```

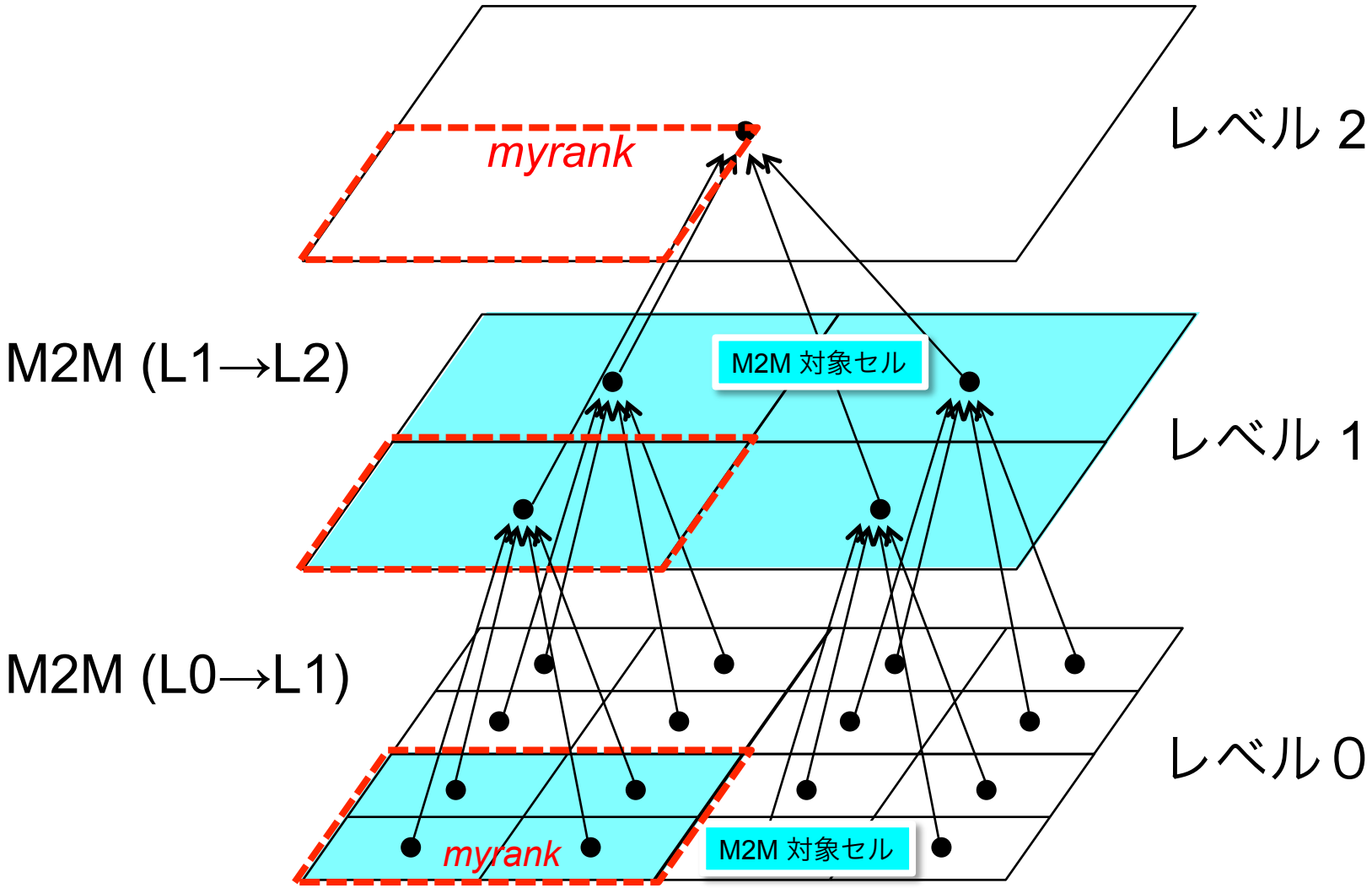


2重の多重計算. しかし
・力の通信なし
・OpenMP, SIMD並列が容易 = 演算効率良

実演算時間 < 実通信時間の範囲で有効 (例えばノード数が多く演算時間が短いとき)

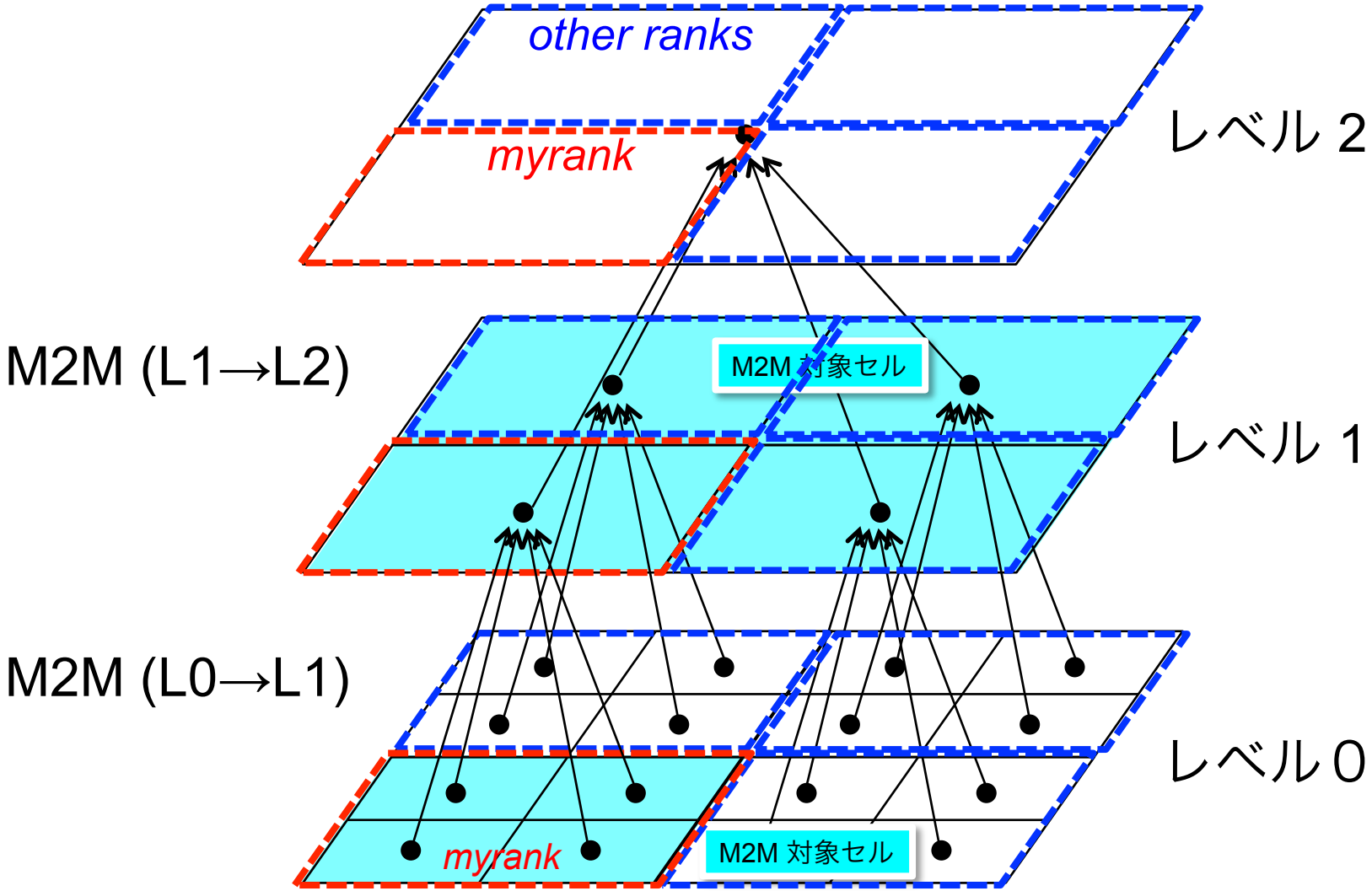
MPI 並列化技術: 通信の演算による代用

冗長な多重演算: FMM 上位階層の M2M



MPI 並列化技術: 通信の演算による代用

冗長な多重演算: FMM 上位階層の M2M

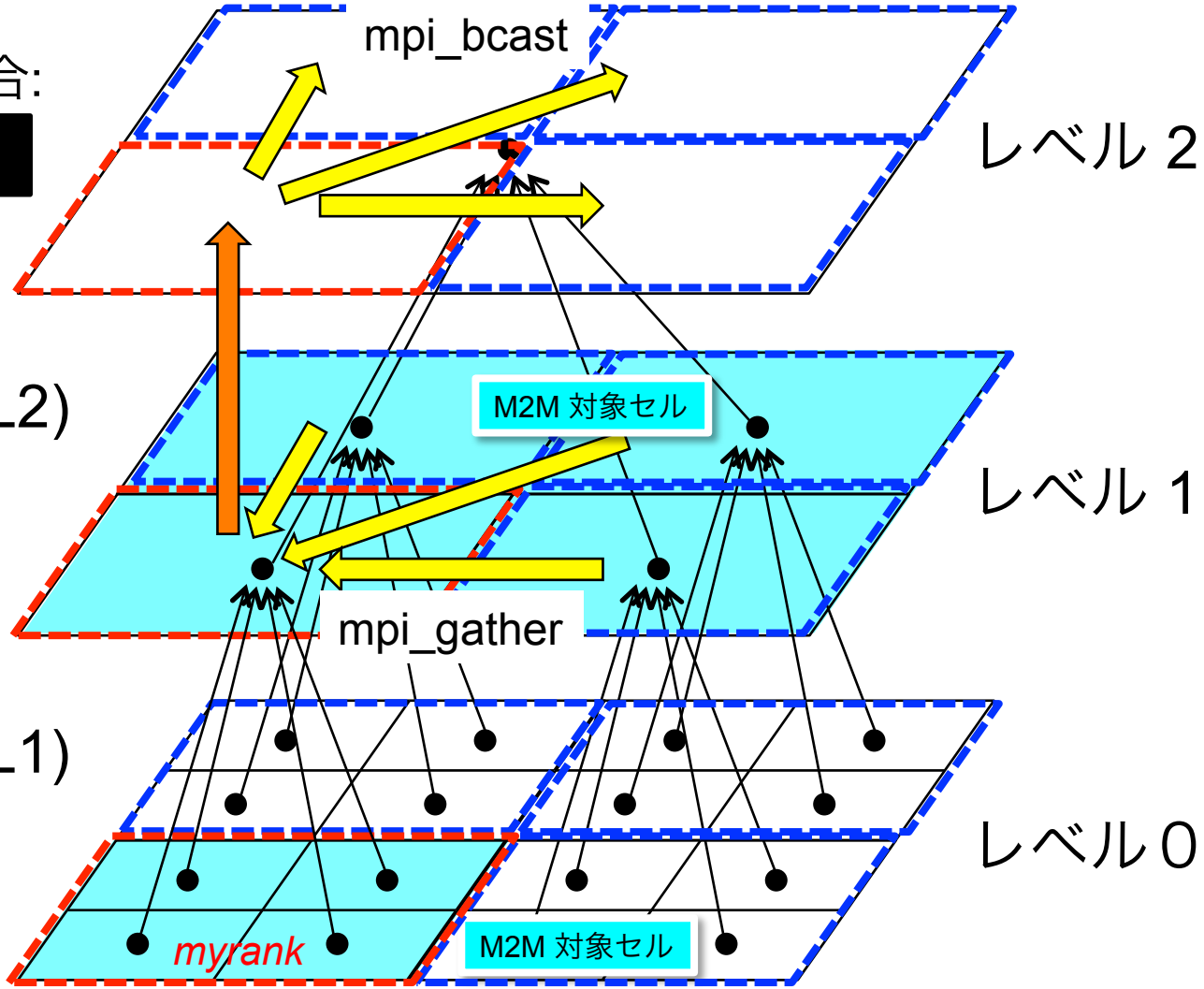


MPI 並列化技術: 通信の演算による代用

冗長な多重演算: FMM 上位階層の M2M

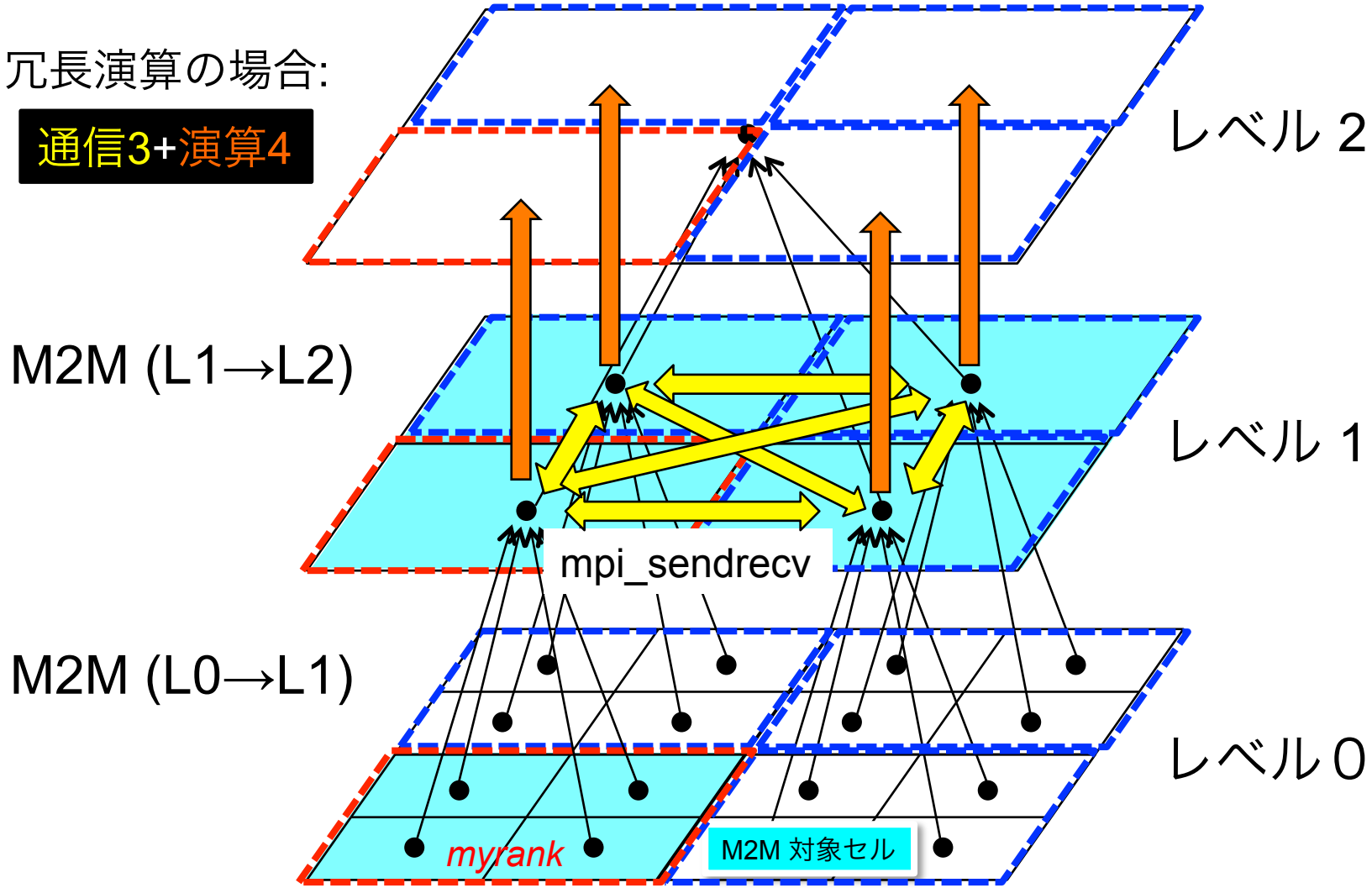
単一演算の場合:

通信6+演算1



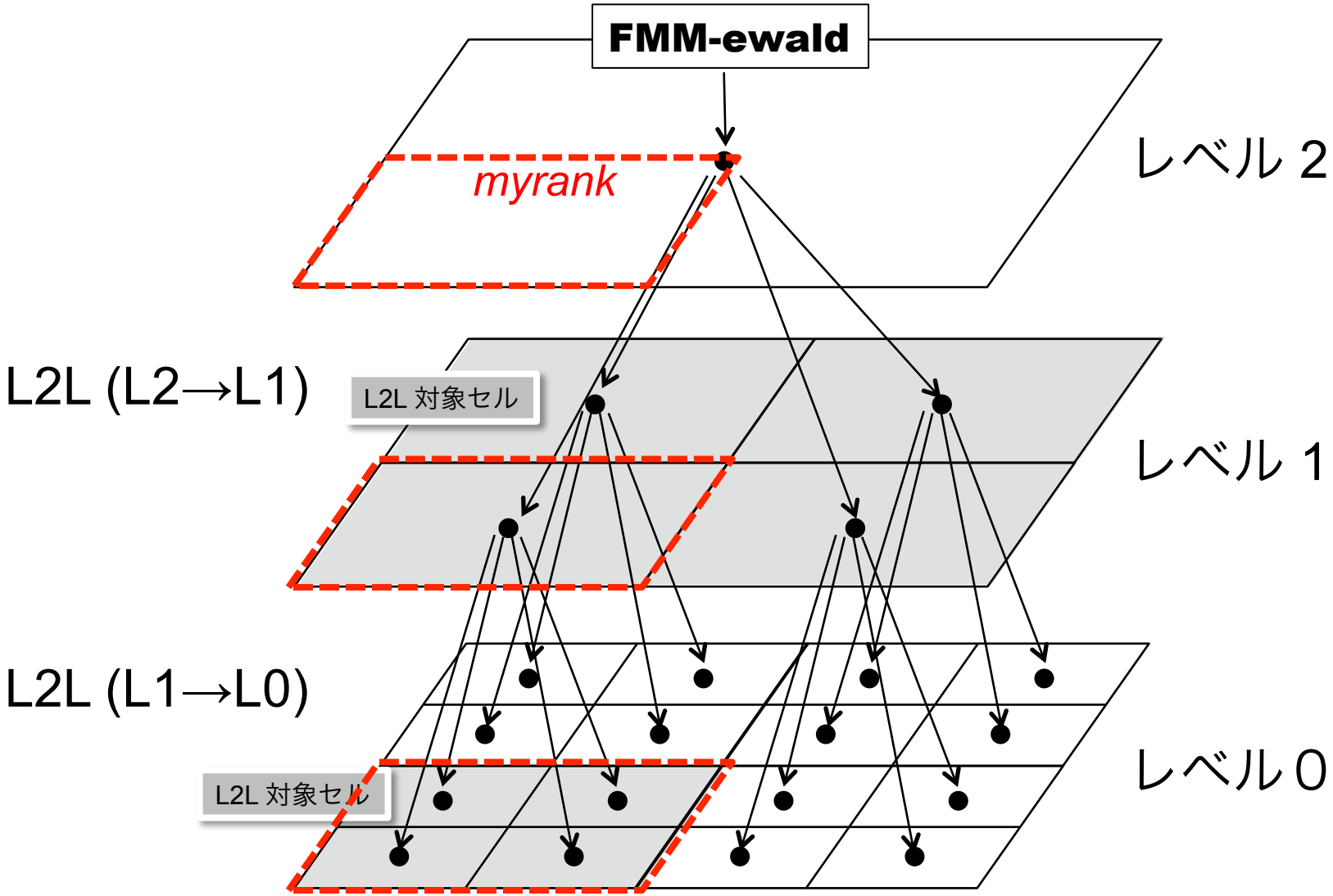
MPI 並列化技術: 通信の演算による代用

冗長な多重演算: FMM 上位階層の M2M



MPI 並列化技術: 通信の演算による代用

冗長な多重演算: FMM 上位階層の L2L

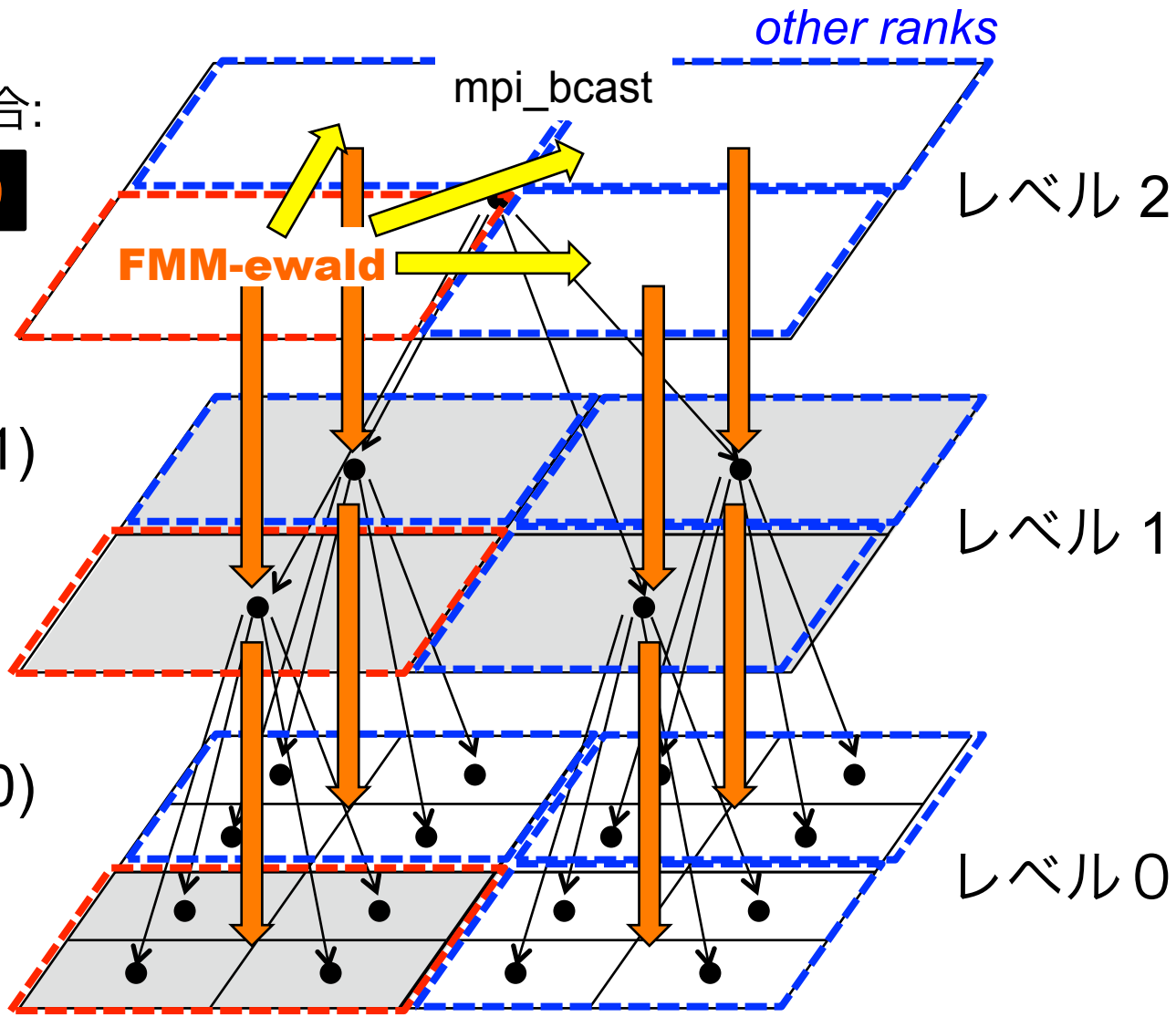


MPI 並列化技術: 通信の演算による代用

冗長な多重演算: FMM 上位階層の L2L

単一演算の場合:

通信3+演算9



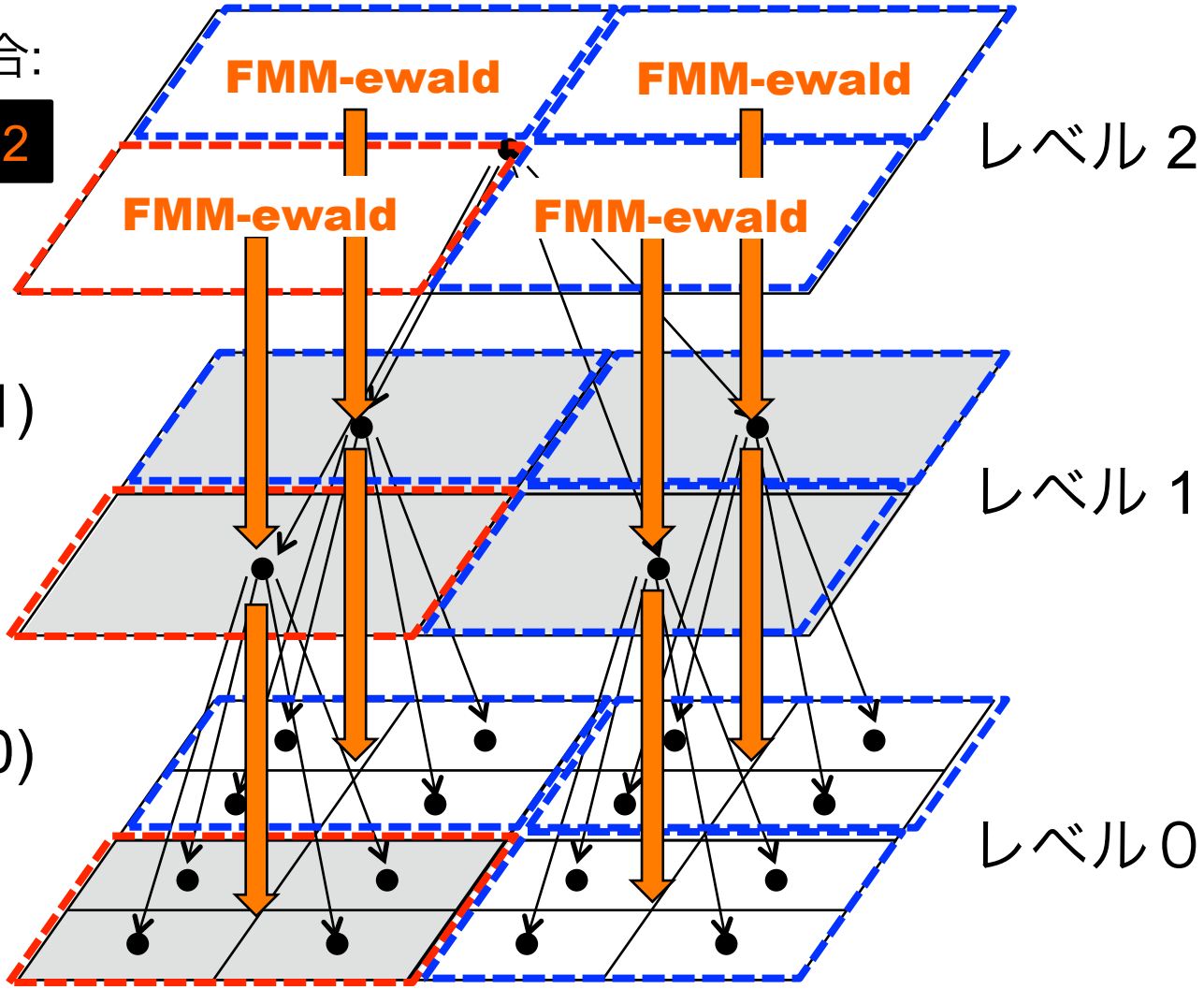
MPI 並列化技術: 通信の演算による代用

冗長な多重演算: FMM 上位階層の L2L

注) 最上位層の wm は M2Mの過程でプロセス間で冗長に保持

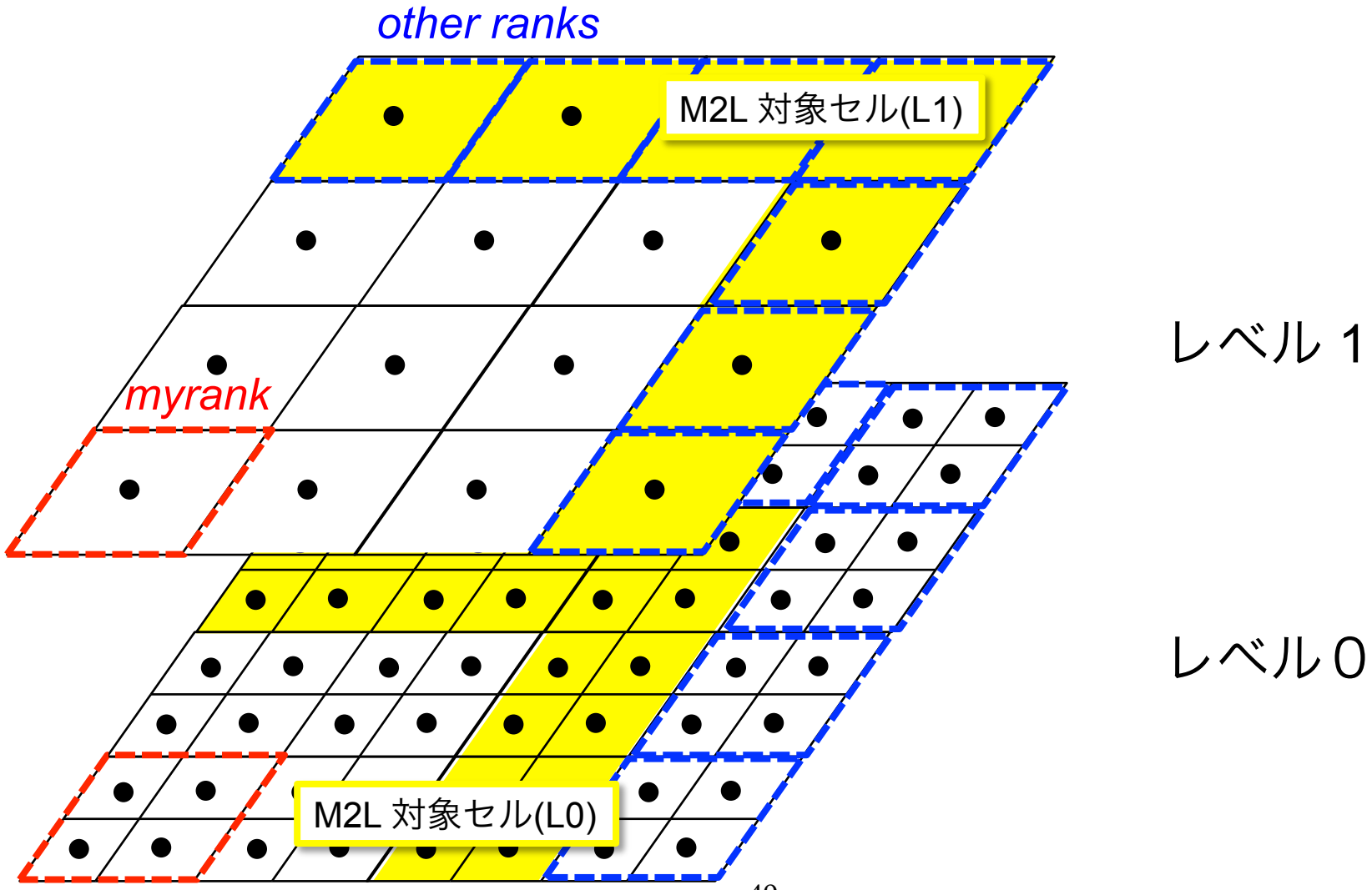
冗長演算の場合:

通信0+演算12



MPI 並列化技術: 通信の演算による代用

M2L は?

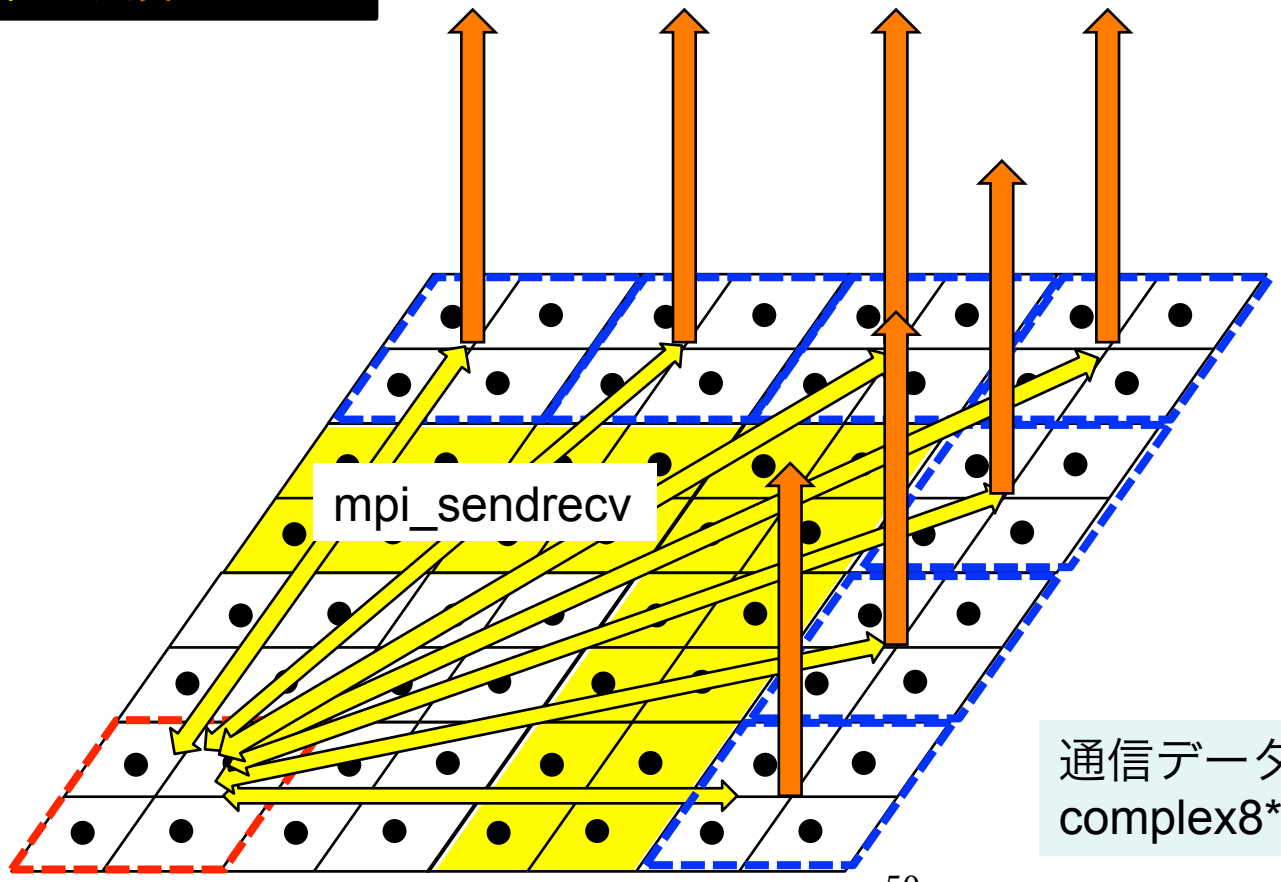


MPI 並列化技術: 通信の演算による代用

M2L は?

冗長演算の場合:

通信7+演算7+M2L



レベル 1

レベル 0

通信データ量:
complex8*(nmax+1)^2*4サブセル

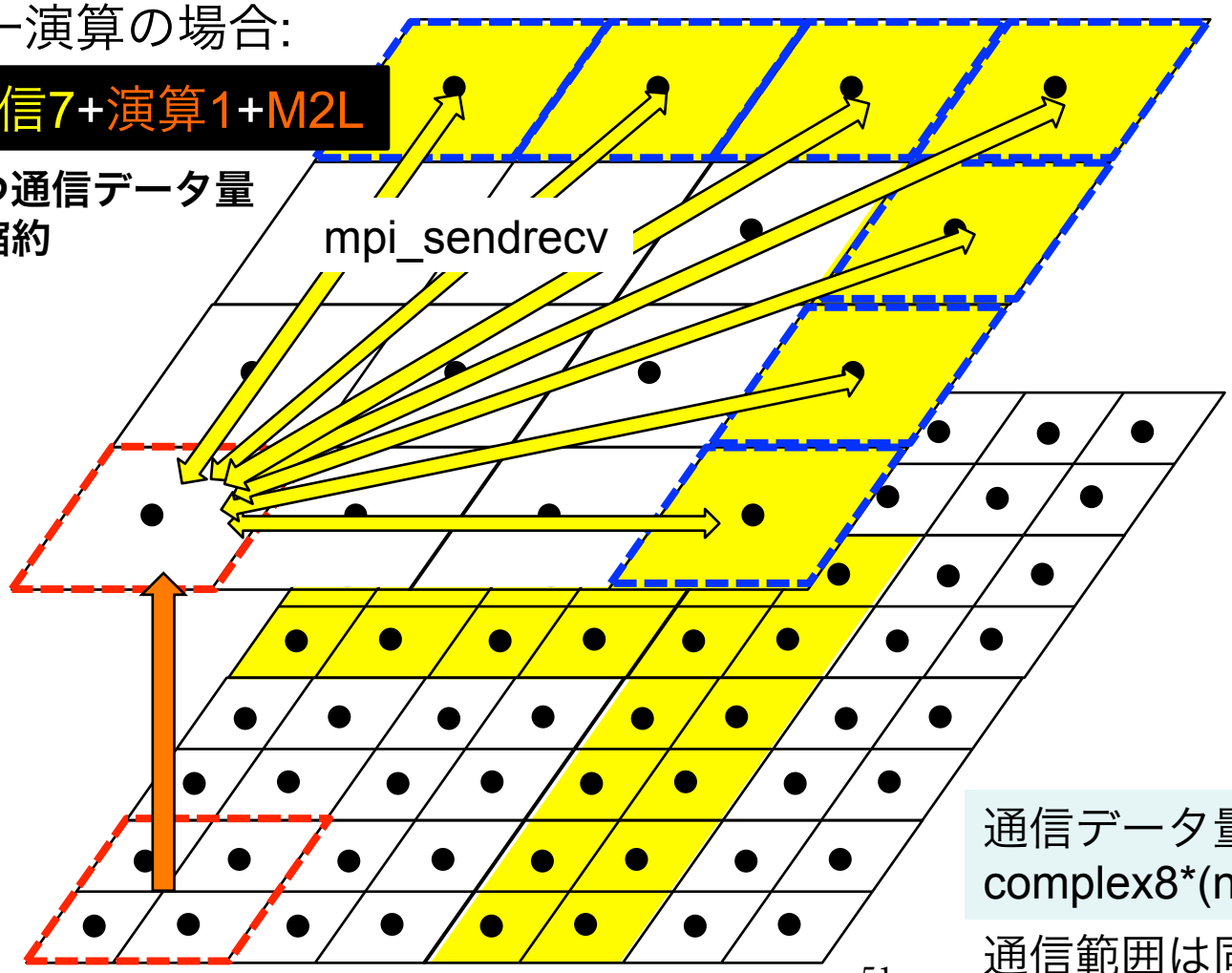
MPI 並列化技術: 通信の演算による代用

M2L は? →下層での通信は効率的でない

単一演算の場合:

通信7+演算1+M2L

かつ通信データ量を縮約



レベル 1

レベル 0

通信データ量:
complex8*(nmax+1)^2*1スーパセル

通信範囲は同じ

並列化技術 3

演算効率化の前提

・データの連続化

・ブロック化によるキャッシュの有効利用

OpenMP 並列化技術

・スレッド間のロードバランス調整 (M2L)

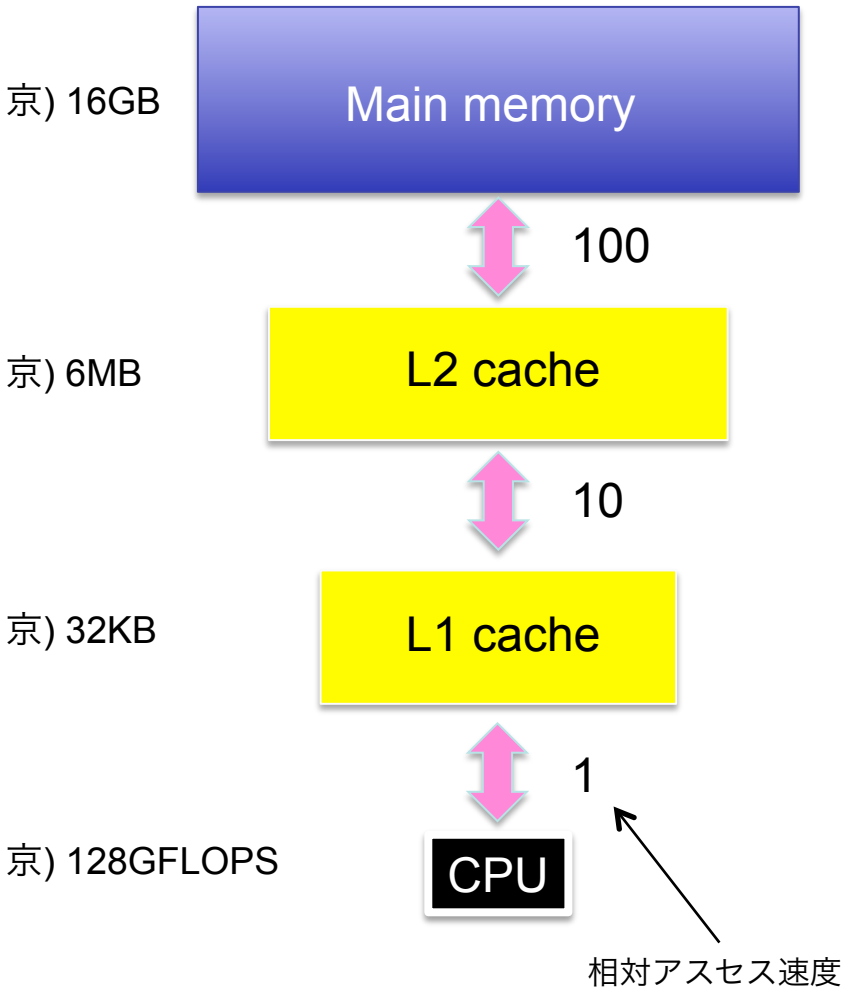
・スレッド並列前後処理の削減

SIMD 並列化技術

・IF文の削除

・ベクトル長の確保

演算効率化のために



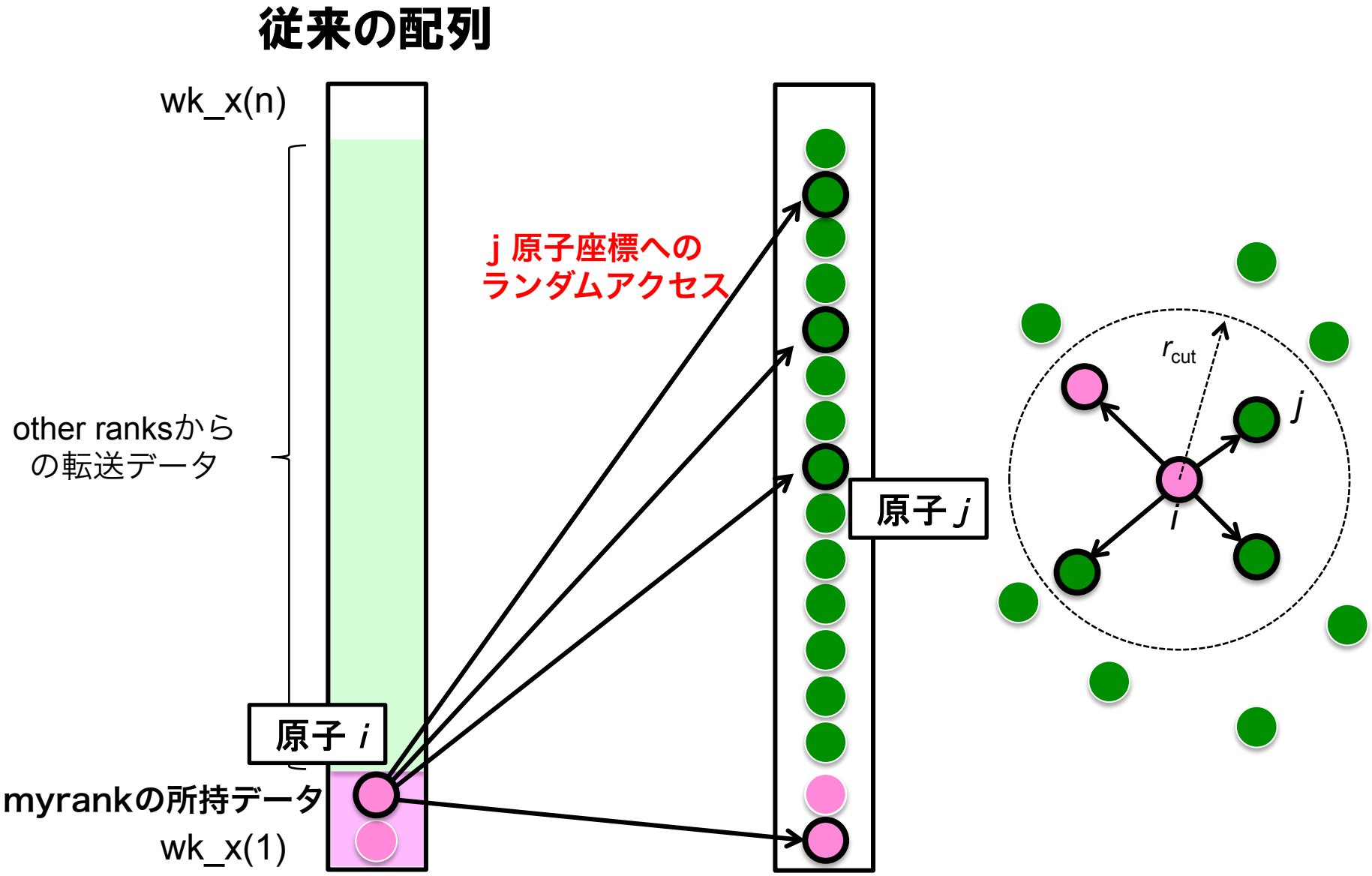
CPUを効率よく動作させるためには
L1 キャッシュの有効利用が必要.

MD計算では, 例えば
・L1上相手 j 原子座標の再利用促進
・L1上 wm, M2L 変換行列の再利用促進

前提

(1) メモリ上の演算対象データの連続化
(2) ブロック化

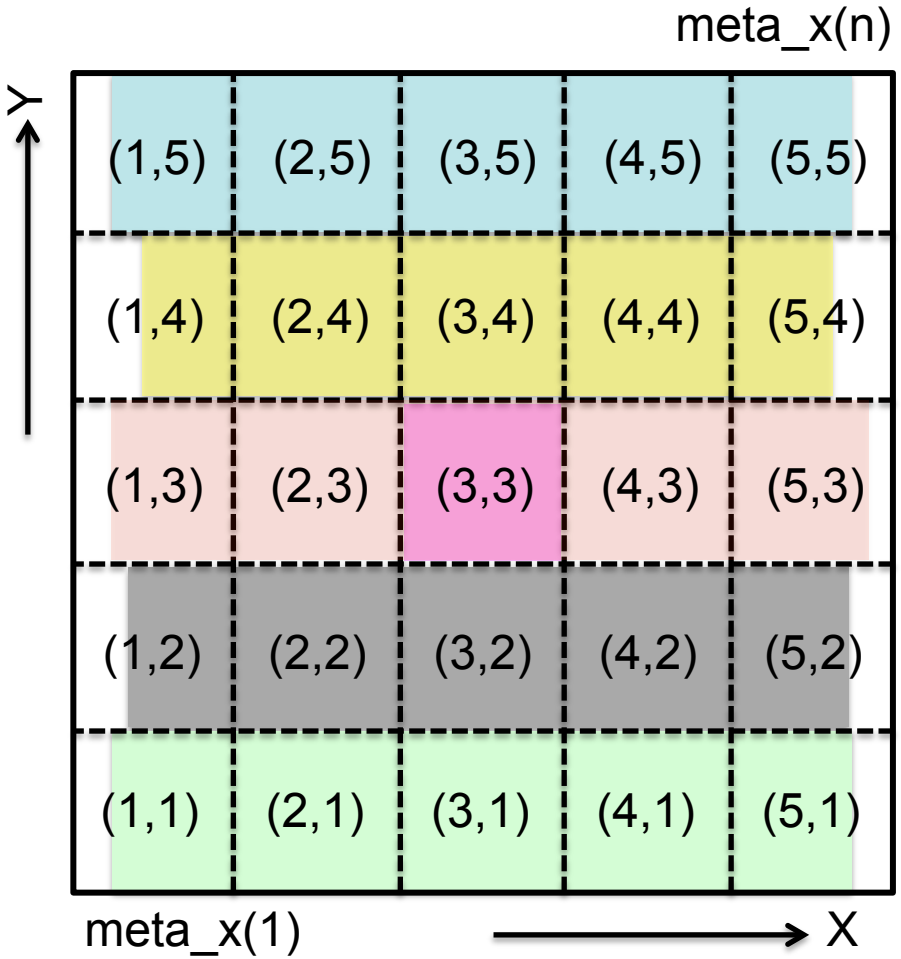
データの連続化　[1] 座標

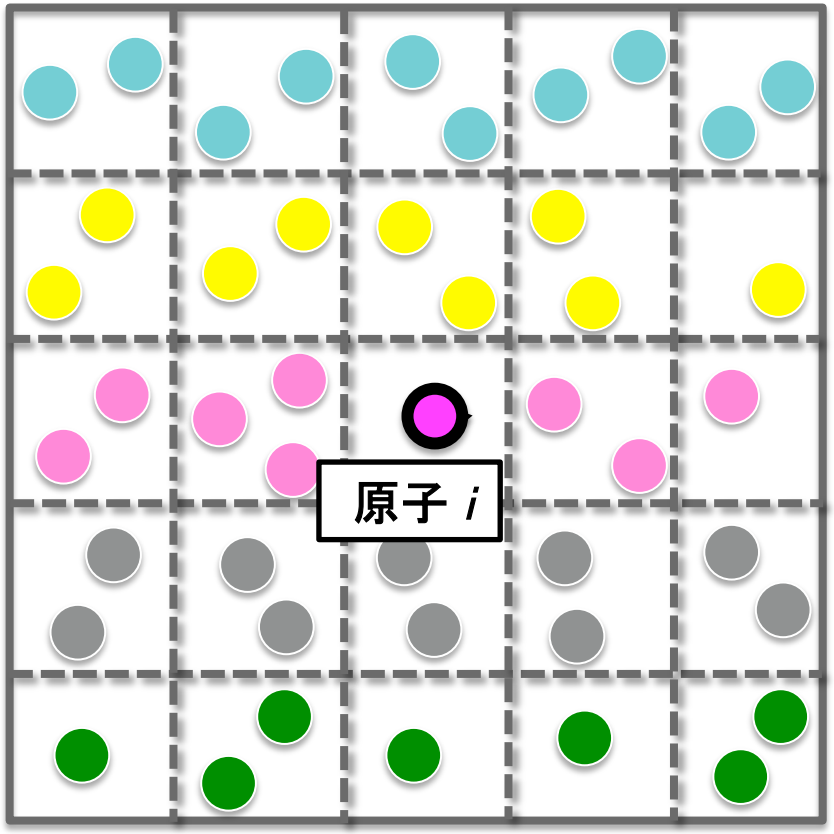

従来の配列
wk_x(n)
other ranksから
の転送データ
原子 i
myrankの所持データ
wk_x(1)
j 原子座標へのランダムアクセス
原子 j
r_{cut}
j
i

データの連続化　[1] 座標

メタデータ配列

meta_x: X-Y 平面上での原子の相対位置関係 (= **メタデータ**) を保持





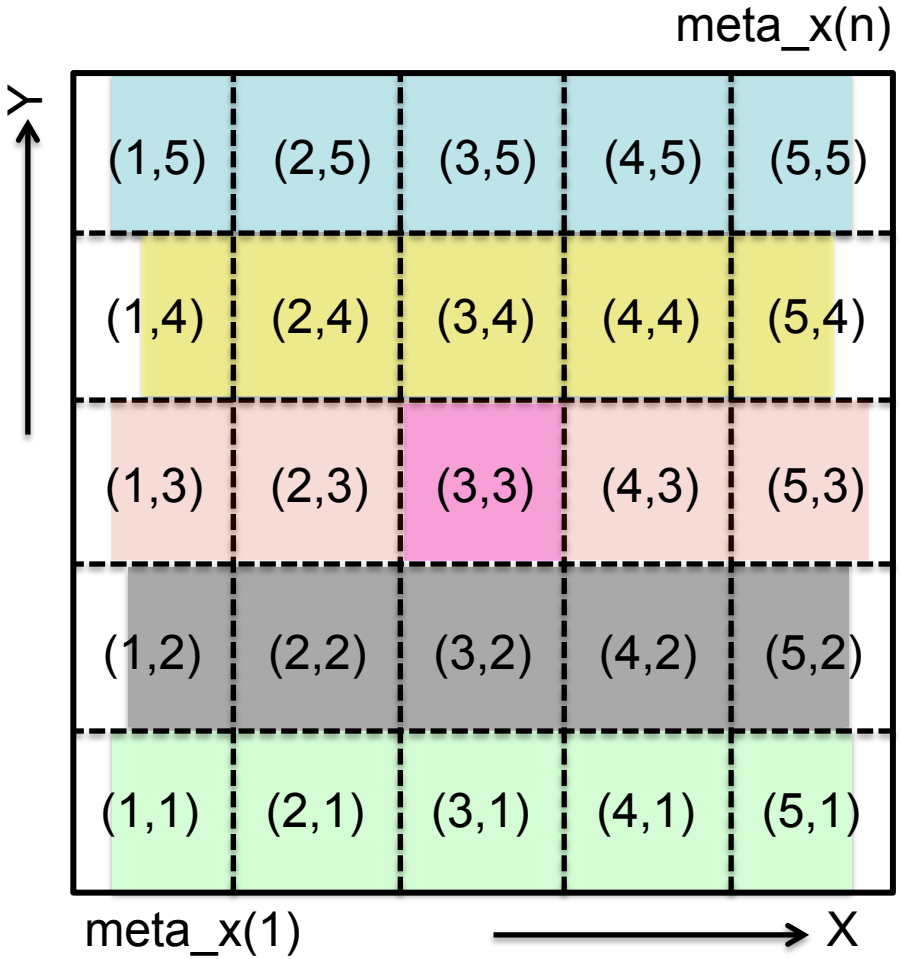
1) 自プロセスを中心にデータを局所化

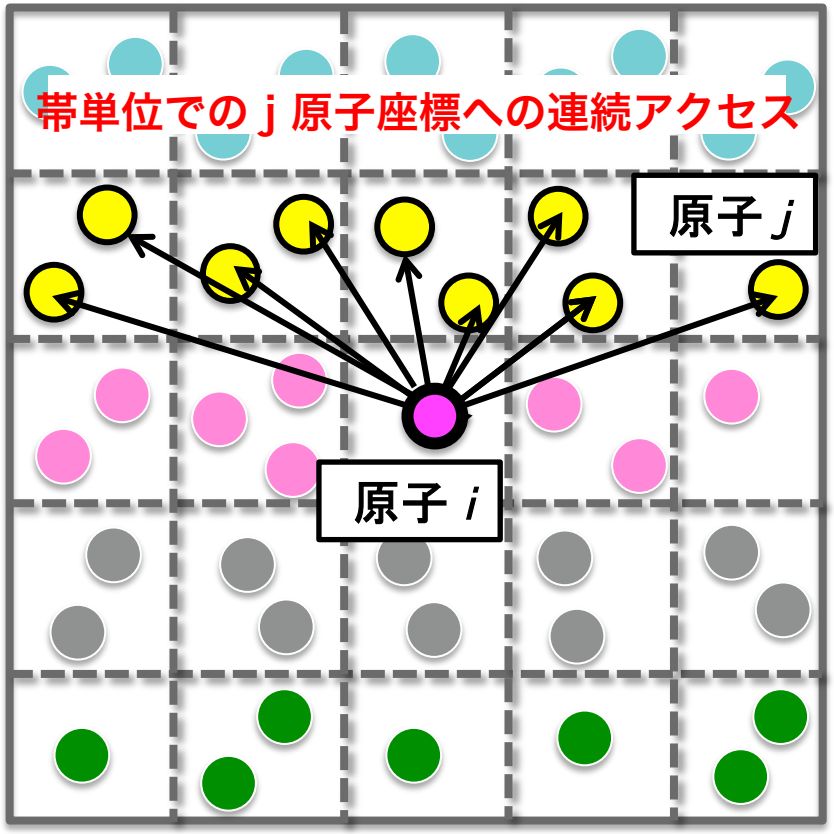
2) 袖部に直にデータ装填

データの連続化　[1] 座標

メタデータ配列

meta_x: X-Y 平面上での原子の相対位置関係 (= **メタデータ**) を保持



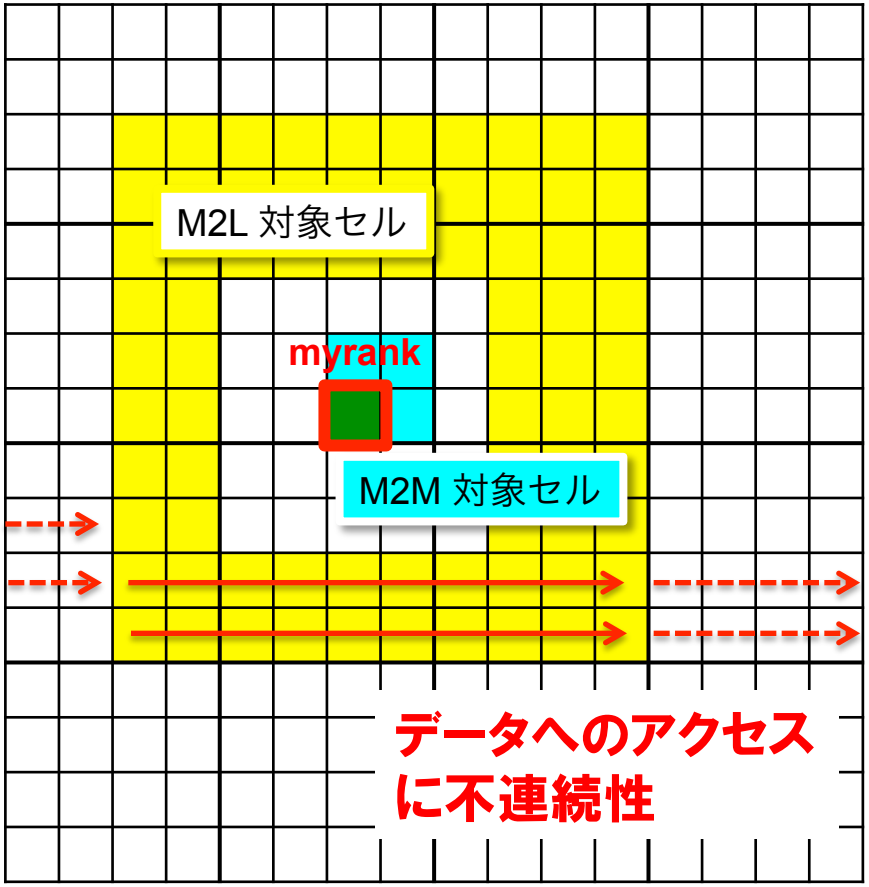


1) 自プロセスを中心にデータを局所化

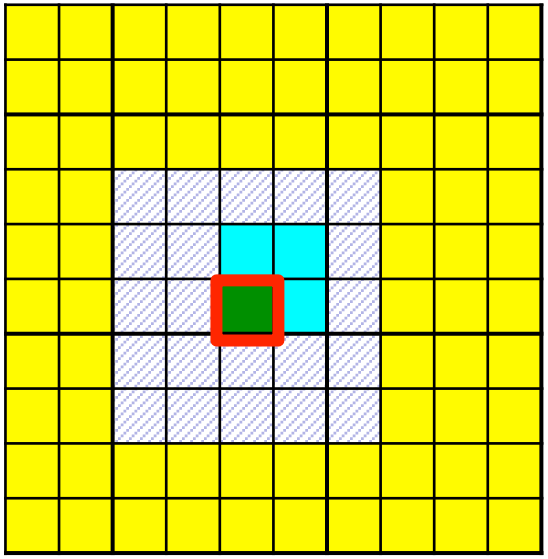
2) 袖部に直にデータ装填

データの連続化　[2] 多極子

従来の配列



袖部付き局所化配列

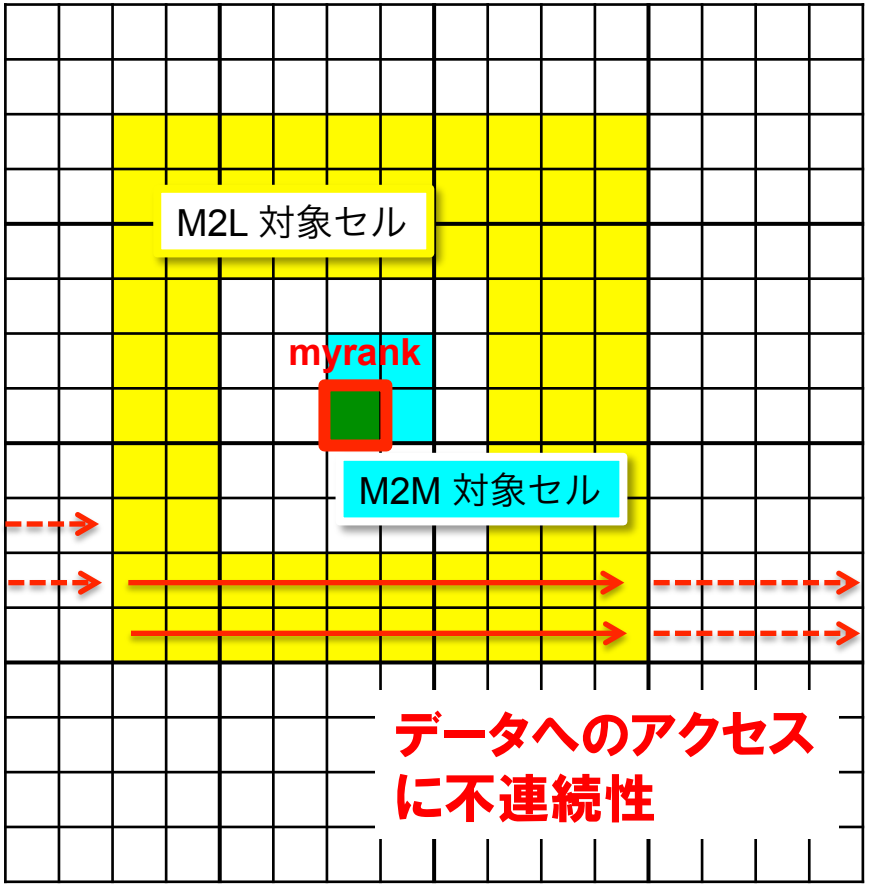


1) 自プロセスを中心にデータを局所化

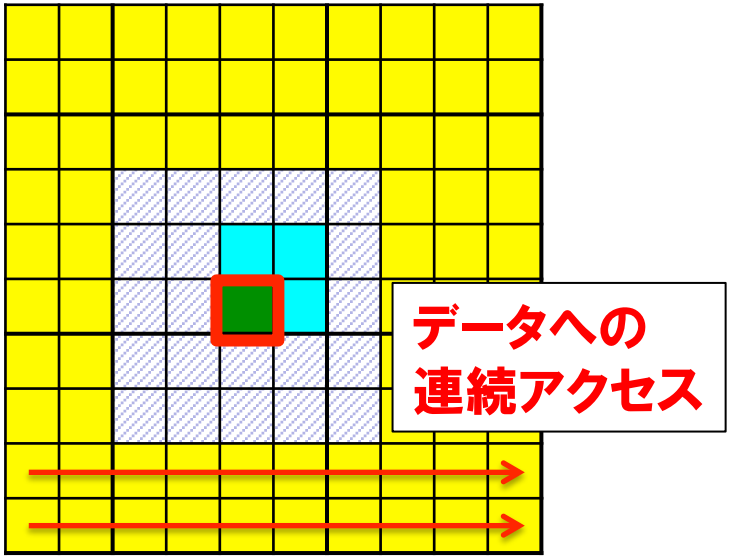
2) 袖部に直にデータ装填

データの連続化　[2] 多極子

従来の配列



袖部付き局所化配列



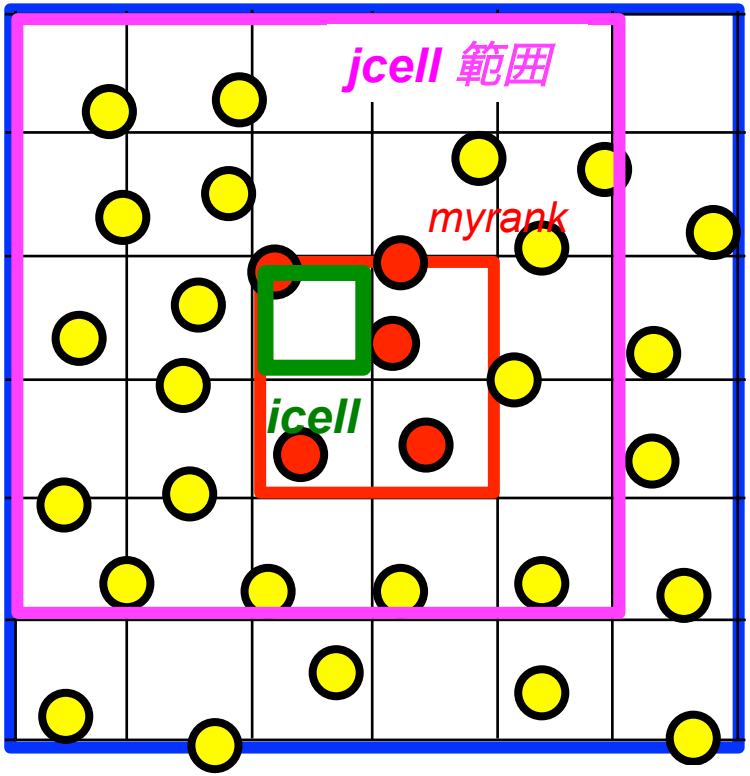
1) 自プロセスを中心にデータを局所化

2) 袖部に直にデータ装填

ブロック化によるキャッシュの有効利用

ブロック化前のループ構造:

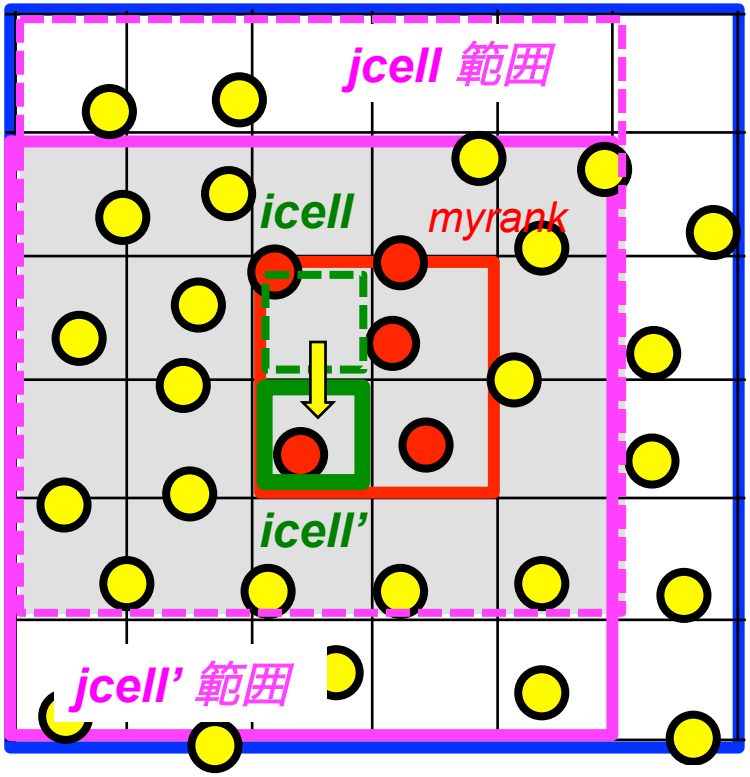
```
do icell(myrank)
do jcell
  do iatm=tag(icell),
             tag(icell)+na_per_cell(icell)-1
  do jatm=tag(jcell),
             tag(jcell)+na_per_cell(jcell)-1
    ポテンシャルの計算
    力の計算
  enddo
  enddo
enddo
enddo
```



ブロック化によるキャッシュの有効利用

ブロック化前のループ構造:

```
do icell(myrank)
do jcell
  do iatm=tag(icell),
             tag(icell)+na_per_cell(icell)-1
  do jatm=tag(jcell),
             tag(jcell)+na_per_cell(jcell)-1
    ポテンシャルの計算
    力の計算
  enddo
  enddo
enddo
enddo
```



座標データ転送済範囲

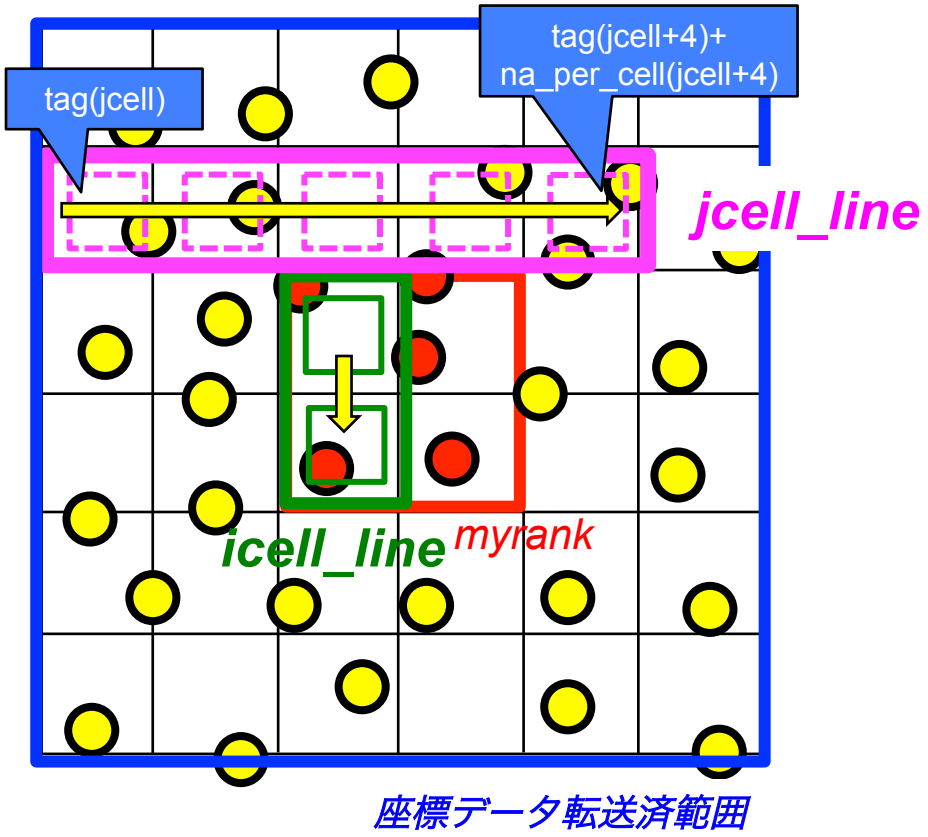
問題点:

(1) 右図灰色部分のデータは icell ⇨ icell' と変わった結果メモリから再ロードされる
よって、いったんキャッシュに乗ったデータを使い切っていない
(2) jcell 内原子数が少なく最内ベクトル長が稼げない．よって, SIMDの性能が出ない

ブロック化によるキャッシュの有効利用

ブロック化後のループ構造:

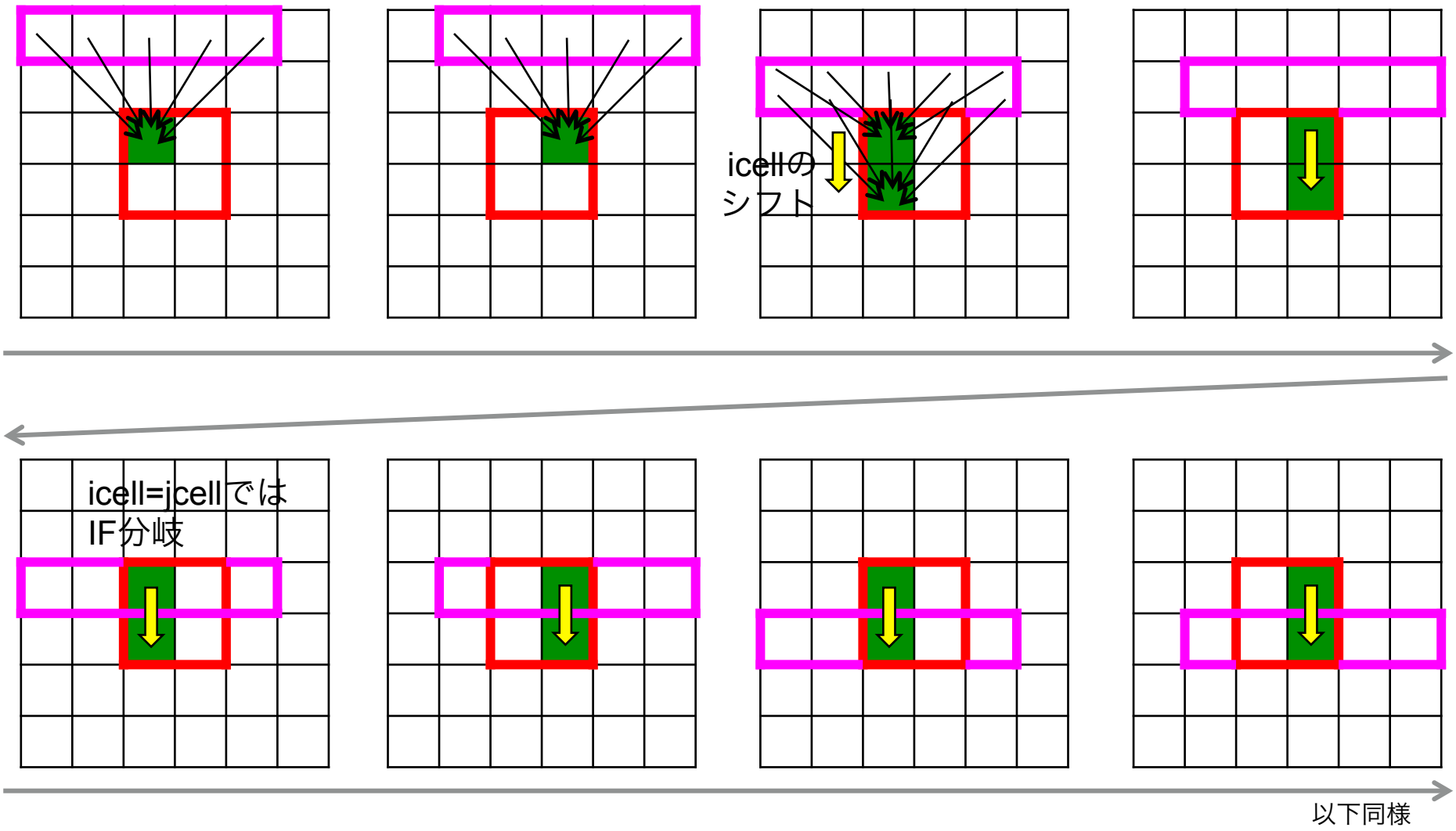
```
do jcell_line
do icell(myrank)              [along icell_line]
  do iatom=tag(icell),
                tag(icell)+na_per_cell(icell)-1
  do jatom=tag(jcell),
          tag(jcell+4)+na_per_cell(jcell+4)-1
    ポテンシャルの計算
    力の計算
  enddo
  enddo
enddo
enddo
```



(1) jcell_lineの原子座標データは 1 回のみロードされる.
　すなわち、いったんキャッシュに乗ったデータを使い切る.
(2) jcell_line 内原子数は 5 倍のベクトル長, かつ連続.　よってSIMDの性能アップ.

ブロック化によるキャッシュの有効利用

最外DOループによる jcell_line シフトの概念図

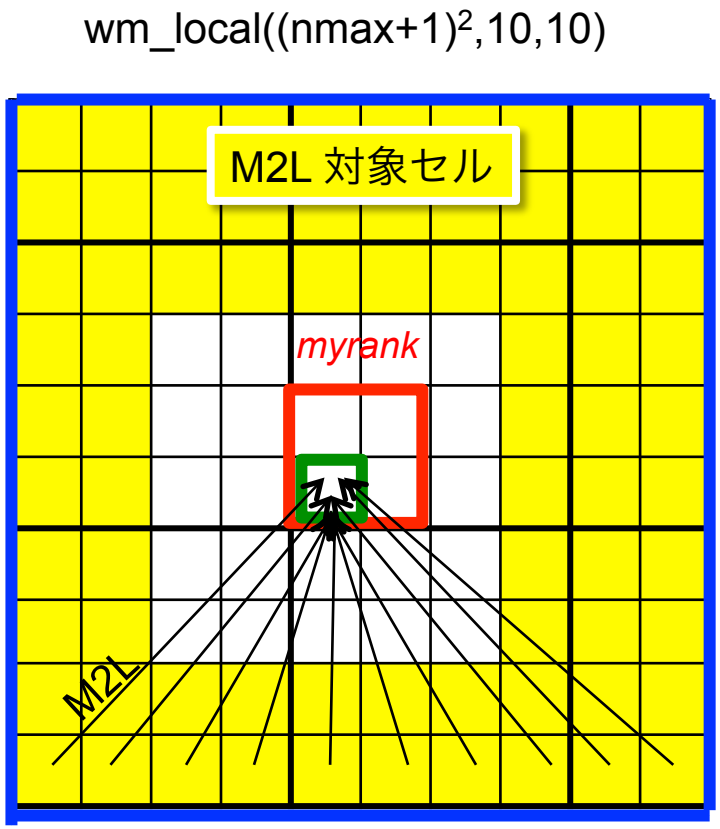


実際のコードは3次元のシフトに対応　**md_direct_f90.f**

ブロック化によるキャッシュの有効利用

ブロック化前のループ構造 (M2L):

```
do icell(myrank)
do icy=1,10
do icx=1,10
if(icx,icyが近接2セル以遠の領域にあれば)
do m1=1,(nmax+1)^2
do m2=1,(nmax+1)^2
   wl_local=wl_local+m2l*wm_local
enddo
enddo
endif
enddo
enddo
enddo
```

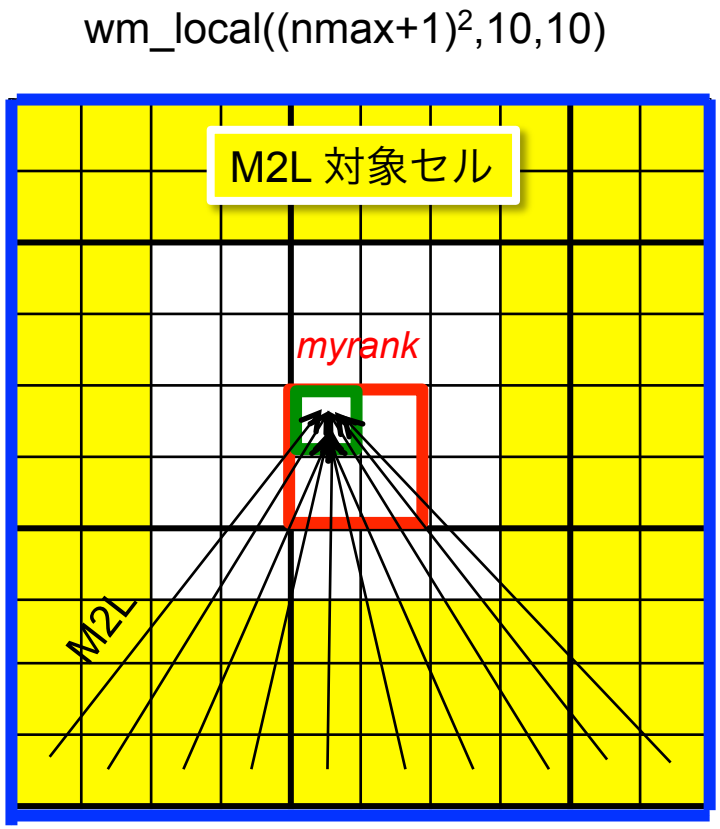


多極子データ転送済範囲

ブロック化によるキャッシュの有効利用

ブロック化前のループ構造 (M2L):

```
do icell(myrank)
do icy=1,10
do icx=1,10
if(icx,icyが近接2セル以遠の領域にあれば)
do m1=1,(nmax+1)^2
do m2=1,(nmax+1)^2
   wl_local=wl_local+m2l*wm_local
enddo
enddo
endif
enddo
enddo
enddo
```

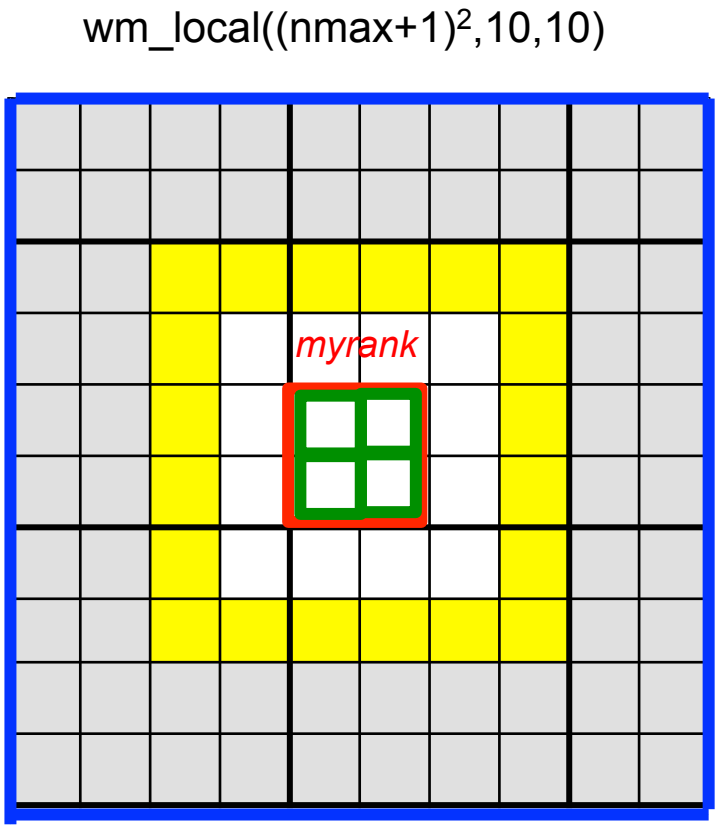


多極子データ転送済範囲

ブロック化によるキャッシュの有効利用

ブロック化前のループ構造 (M2L):

```
do icell(myrank)
do icy=1,10
do icx=1,10
if(icx,icyが近接2セル以遠の領域にあれば)
do m1=1,(nmax+1)^2
do m2=1,(nmax+1)^2
   wl_local=wl_local+m2l*wm_local
enddo
enddo
endif
enddo
enddo
enddo
```



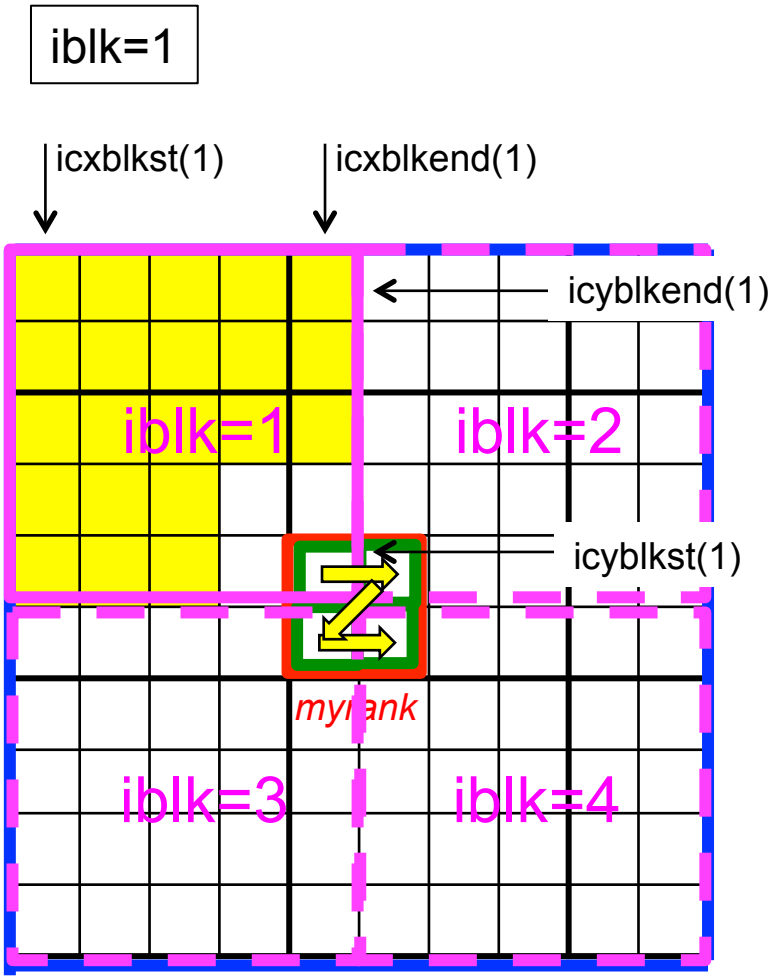
問題点

・右図灰色の領域の**wm_local**, 変換行列**m2l**は icell が変わるごとメモリから再ロード.
　よって、いったんキャッシュに乗ったデータを使い切っていない.

ブロック化によるキャッシュの有効利用

ブロック化後のループ構造 (M2L):

```
do iblk=1,nblock
do icy=icyblkst(iblk),icyblkend(iblk)
do icx=icxblkst(iblk),icxblkend(iblk)
do icell(myrank)
if(icx,icyが近接2セル以遠の領域にあれば)
do m1=1,(nmax+1)^2
do m2=1,(nmax+1)^2
   wl_local=wl_local+m2l*wm_local
enddo
enddo
endif
enddo
enddo
enddo
enddo
```

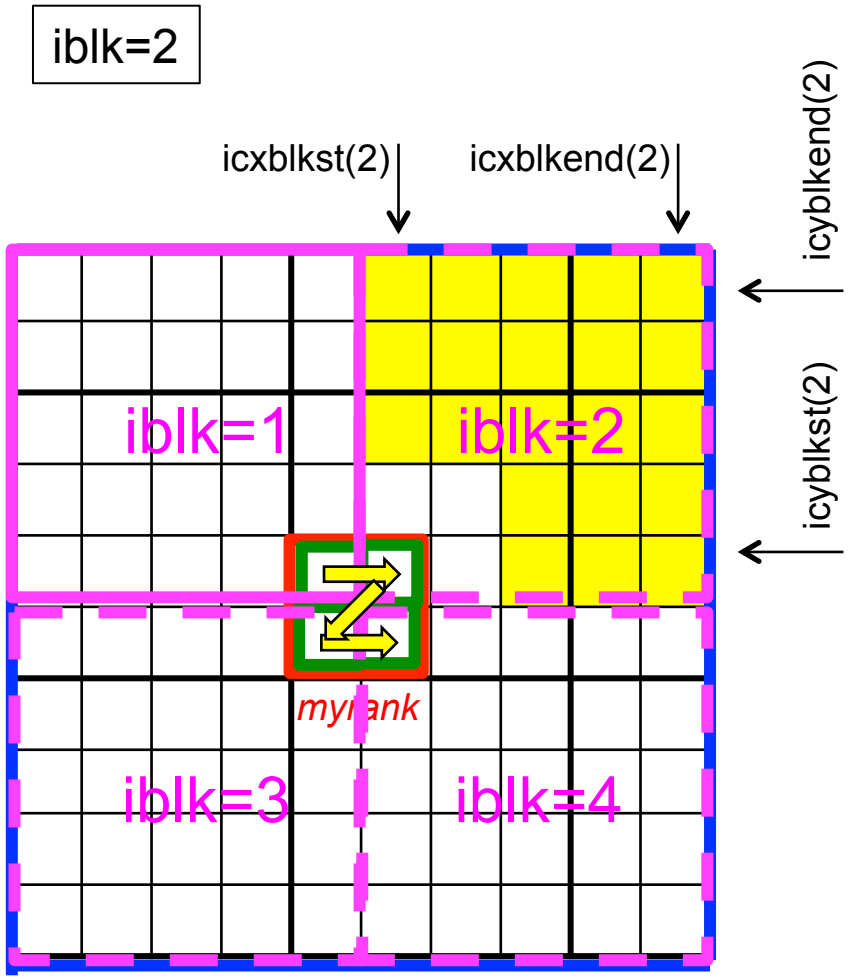


・iblk の**wm_local**, 変換行列**m2l**は 各 icell に対し 1 回のみロードされる.
すなわち、いったんキャッシュに乗ったデータを使い切る.

ブロック化によるキャッシュの有効利用

ブロック化後のループ構造 (M2L):

```
do iblk=1,nblock
do icy=icyblkst(iblk),icyblkend(iblk)
do icx=icxblkst(iblk),icxblkend(iblk)
do icell(myrank)
if(icx,icyが近接2セル以遠の領域にあれば)
do m1=1,(nmax+1)^2
do m2=1,(nmax+1)^2
   wl_local=wl_local+m2l*wm_local
enddo
enddo
endif
enddo
enddo
enddo
enddo
```



・iblk の**wm_local**, 変換行列**m2l**は 各 icell に対し 1 回のみロードされる.
すなわち、いったんキャッシュに乗ったデータを使い切る.

OpenMP 並列化技術

ブロック化+ロードインバランス調整後のループ構造 (M2L):

```
!$omp parallel
!$omp do
do iblk=1,nblock
do icy=icyblkst(iblk),icyblkend(iblk)
do icx=icxblkst(iblk),icxblkend(iblk)
do icell(myrank)
if(icx,icyが近接2セル以遠の領域にあれば)
do m1=1,(nmax+1)^2
do m2=1,(nmax+1)^2
   wl_local=wl_local+m2l*wm_local
enddo
enddo
endif
enddo
enddo
enddo
enddo
!$omp end do
!$omp end parallel
```

あらかじめM2L対象セルをスレッド数にあわせ均等にブロック化しておく.

任意のスレッド数に対応できないという問題

例) x,y,z方向に2分割, 計8ブロック分割.

→8スレッドまでの2ベキスレッド数には対応. 8より大きなスレッド数には未対応.

OpenMP 並列化技術

ブロック化+ロードインバランス調整後のループ構造 (M2L):

```
!$omp parallel
!$omp do schedule(static,nchunk)
do load=1,nload
  icx=lddir(1,load)
  icy=lddir(2,load)
do icell(myrank)
if(icx,icyが近接2セル以遠の領域にあれば)
do m1=1,(nmax+1)^2
do m2=1,(nmax+1)^2
    wl_local=wl_local+m2l*wm_local
enddo
enddo
endif
enddo
enddo
!$omp end do
!$omp end parallel
```

あらかじめM2L対象セル番地をlddirに登録しておく. nchunk個ごとスレッドに割り当てる.

- **任意のスレッド数に対応**
- **異方的セル分割(均等2ベキ以外)にも対応可能**

md_fmm_f90.f

OpenMP 並列化技術

- スレッド並列前後処理の削減

```
do jcell_line
do icell            [along icell_line]
!$omp parallel
!$omp do
do iatom=tag(icell),
                tag(icell)+na_per_cell(icell)-1
  do jatom=tag(jcell),
          tag(jcell+4)+na_per_cell(jcell+4)-1
    ポテンシャルの計算
    力の計算
  enddo
enddo
!$omp enddo
!$omp end parallel
enddo
enddo
```

OpenMP 並列化技術

- スレッド並列前後処理の削減

$!omp parallelを大外に出す

jcell_line数*icell数回の
parallel領域open/closeの
オーバヘッドが削減

```
do jcell_line
do icell          [along icell_line]
!$omp parallel
!$omp do
do iatom=tag(icell),
            tag(icell)+na_per_cell(icell)-1
  do jatom=tag(jcell),
        tag(jcell+4)+na_per_cell(jcell+4)-1
    ポテンシャルの計算
    力の計算
  enddo
enddo
!$omp enddo
!$omp end parallel
enddo
enddo
```

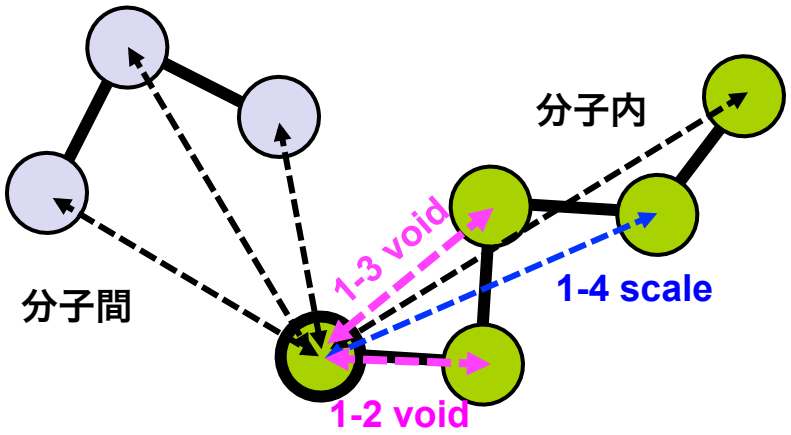
```
!$omp parallel
do jcell_line
do icell          [along icell_line]
!$omp do
do iatom=tag(icell),
            tag(icell)+na_per_cell(icell)-1
  do jatom=tag(jcell),
        tag(jcell+4)+na_per_cell(jcell+4)-1
    ポテンシャルの計算
    力の計算
  enddo
enddo
!$omp enddo
enddo
enddo
!$omp end parallel
```

SIMD 並列化技術

・IF文の削除, ベクトル長の確保

分子のループではOpenMP並列の粒度,
および最内ベクトル長が確保できない

話をだいぶ遡ぼれば・・・



分子間

```
do imol=1,nmol-1
do jmol=imol+1,nmol
do i=1,natom(imol)
do j=1,natom(jmol)
  rij=rij(ri, rj)
  LJカットオフ判定
  φnonbond=φnonbond+φij
  f(i)=f(i)+Fi
  f(j)=f(j)+Fj
enddo
enddo
enddo
enddo
```

分子内

```
do imol=1,nmol
do i=1,natom(imol)-1
do j=i+1,natom(imol)
  rij=rij(ri, rj)
  LJカットオフ判定
  x = { 0  if 1-2,-3 void
      { s  if 1-4 scale
      { 1  else
  φnonbond=φnonbond+x*φij
  f(i)=f(i)+x*Fi
  f(j)=f(j)+x*Fj
enddo
enddo
enddo
```

分子内 *nonbond* 項計算の注意点:

・1番目および2番目の隣接原子とは相互作用しない (1-2, 1-3 void)

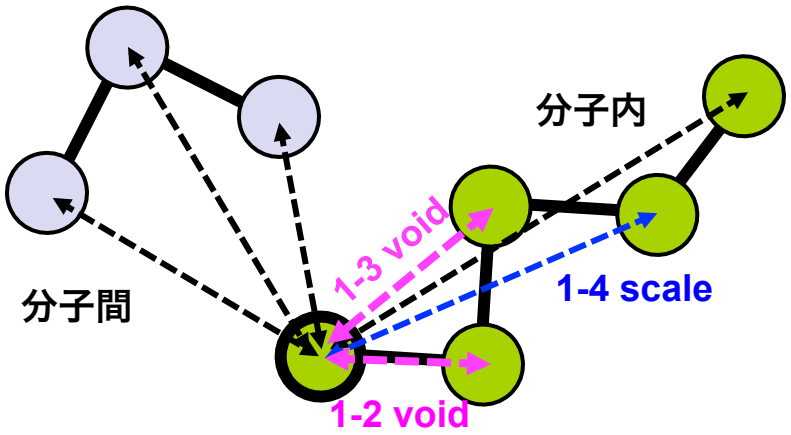
・3番目の隣接原子との相互作用は因子 *s* でスケールする (1-4 scale) [*s* は LJ, Coulomb べつ]

・4番目以降の隣接原子とは通常の相互作用

SIMD 並列化技術

- **IF文の削除, ベクトル長の確保**

話をだいぶ遡ぼれば・・・



分子間と分子内の統合

原子ループでOpenMP並列の粒度を確保

```
do jcell_line
do icell  [along icell_line]
do iatom=tag(icell),
          tag(icell)+na_per_cell(icell)-1
  do jatom=tag(jcell),
        tag(jcell+4)+na_per_cell(jcell+4)-1
   rij=rij(ri,rj)
   LJカットオフ判定
   x= { 0  if 1-2,-3 void
      { s  if 1-4 scale
      { 1  else
   φnonbond=φnonbond+x*φij
   f(i)=f(i)+x*Fi
   f(j)=f(j)+x*Fj
  enddo
enddo
enddo
enddo
```

jcell_line原子ループでベクトル長を確保

代償として if 文がループ最内側に移動

分子内 *nonbond* 項計算の注意点:

- 1番目および2番目の隣接原子とは相互作用しない (1-2, 1-3 void)
- 3番目の隣接原子との相互作用は因子 *s* でスケールする (1-4 scale), *s* は LJ, Coulomb べつ
- 4番目以降の隣接原子とは通常の相互作用

SIMD 並列化技術

- **IF文の削除, ベクトル長の確保**

```
do jcell_line
do icell  [along icell_line]
do iatom=tag(icell),
            tag(icell)+na_per_cell(icell)-1
do jatom=tag(jcell),
      tag(jcell+4)+na_per_cell(jcell+4)-1
   rij=rij(ri,rj)
   if(rij≥rcut) LJ_epsilon=0d0
   φnonbond=φnonbond+φij
   f(i)=f(i)+Fi
   f(j)=f(j)+Fj
  enddo
enddo
enddo
enddo
```

jcell_line原子ループでベクトル長を確保

マスク処理を利用

ひとまずvoid, scaleの区別無くすべて計算

```
do iatom=tag(icell),
            tag(icell)+na_per_cell(icell)-1
  do jatom=1,voidpair123(iatom)
    rij=rij(ri,rj)
    φnonbond=φnonbond−φij
    f(i)=f(i) – Fi
    f(j)=f(j) – Fj
  enddo
  do jatom=1,scalepair14(iatom)
    rij=rij(ri,rj)
    x=1-s
    φnonbond=φnonbond−x*φij
    f(i)=f(i) – x*Fi
    f(j)=f(j) – x*Fj
  enddo
enddo
```

voidを引き算

scale との差分を引き算

赤字：専用計算機 (MDGRAPE) の分野で使われてきたテクニック

問題点

桁落ちによる精度低下 (倍精度なら実用上問題なし).

SIMD並列化技術

コンパイラメッセージ　　frtpx -Qt md_direct_f90.f

```
<<< Loop-information Start >>>
                          <<<  [OPTIMIZATION]
                          <<<    SIMD
                          <<<    SOFTWARE PIPELINING
                          <<< Loop-information  End >>>
      140     9   p    v                                  do j0=tag(jzb-2,jyb,jxb),
      141     9                  &                           tag(jzb+2,jyb,jxb)
      142     9                  &                           + na_per_cell(jzb+2,jyb,jxb)-1
      143     9   p    v                                     rx=xi-wkxyz(1,j0)
      144     9   p    v                                     ry=yi-wkxyz(2,j0)
      145     9   p    v                                     rz=zi-wkxyz(3,j0)
      146     9   p    v                                     r2=rx*rx+ry*ry+rz*rz
      149     9                                              ! ^^^ spherical cut-off ^^^
      150    10   p    v                                     if(r2<=cutrad2) then
      151    10   p    v                                        eps=epsilon_sqrt_i0
      152    10                  &                                *epsilon_sqrt_table(ic,iam)
      153    10   p    v                                     else
      154    10   p    v                                        eps=0d0
      155    10   p    v                                     endif !cut-off
      167     9   p    v                                     sUlj12=sUlj12+Ulj12
      168     9   p    v                                     sUlj6 =sUlj6 +Ulj6
      184     9   p    v                                     sUcoulomb=sUcoulomb+Ucoulomb
      170     9   p    v                                     stlcx=stlcx+tlx
      171     9   p    v                                     stlcy=stlcy+tly
      172     9   p    v                                     stlcz=stlcz+tlz
      185     9   p    v                                     stlcx=stlcx+tcx
      186     9   p    v                                     stlcy=stlcy+tcy
      187     9   p    v                                     stlcz=stlcz+tcz
      188     9   p    v                                     ic=ic+1
      189     9   p    v                                  enddo !j0
```

SIMD並列化技術

コンパイラメッセージ　　frtpx -Qt md_fmm_f90.f

```
1050     1   p                DO load = 1, nload
1051     1   p                  ic = lddir(1,load)
1052     1   p                  jc = lddir(2,load)
1053     1   p                  kc = lddir(3,load)
1091     4                  **** multipole to local translation
                         <<< Loop-information Start >>>
                         <<<  [OPTIMIZATION]
                         <<<    PREFETCH       : 6
                         <<<      wwl_localx: 6
                         <<< Loop-information  End >>>
1092     5   p                        do m1=1,(nmax+1)*(nmax+1)
                         <<< Loop-information Start >>>
                         <<<  [OPTIMIZATION]
                         <<<    SIMD
                         <<<    SOFTWARE PIPELINING
                         <<< Loop-information  End >>>
1093     6   p   6v                     do m2=1,(nmax+1)*(nmax+1)
1094     6   p   6v                       wwl_localx(m1,icz0,icy0,icx0,iam)
1095     6                          $      = wwl_localx(m1,icz0,icy0,icx0,iam)
1096     6                          $      + wm_localx(m2,icz1,icy1,icx1)*shml(m2,m1,kc,jc,ic,nl)
1097     6   p   6v                     enddo
1098     5   p                        enddo
1105     1
1106     1   p                  ENDDO ! load
```

まとめ

- 3次元トーラスネットワークに最適化したMPI並列化手法について, 通信間衝突の回避,通信前後での配列間コピーの消去,および通信の演算による代用をキーワードに解説した.
- 演算効率化の前提となる,データの連続化およびブロック化によるキャッシュの有効利用について解説した.
- OpenMP並列化技術 (スレッド間のロードバランス調整,スレッド並列前後処理の削減) およびSIMD並列化技術 (IF文の削除,ベクトル長の確保) について解説した.

付録 今回の講義でスキップした項目

- 非同期通信 (mpi_isend, mpi_irecv)
- 反作用力の演算, 通信コード
- FMM上位階層でのデータ構造および通信方式の切り替え
- 原子間距離拘束 (SHAKE/RATTLE), および反復回数削減アルゴリズム (p-SHAKE/p-RATTLE) の並列化
- *NVT*, *NPT* アンサンブルコードの並列化
- マルチタイムステップコードの並列化
- セル分割数およびプロセス分割数 2^n*3^m への対応
- PME法の袖部付き局所化データ形式での実装