

CMSI配信講義B (2016)

第15回

大規模MD並列化の技術2

名古屋大学 工学研究科
附属計算科学連携教育研究センター
特任講師

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

- 並列化技術 2 MPI
- 並列化技術 3 OpenMP, SIMD

第一回

第二回

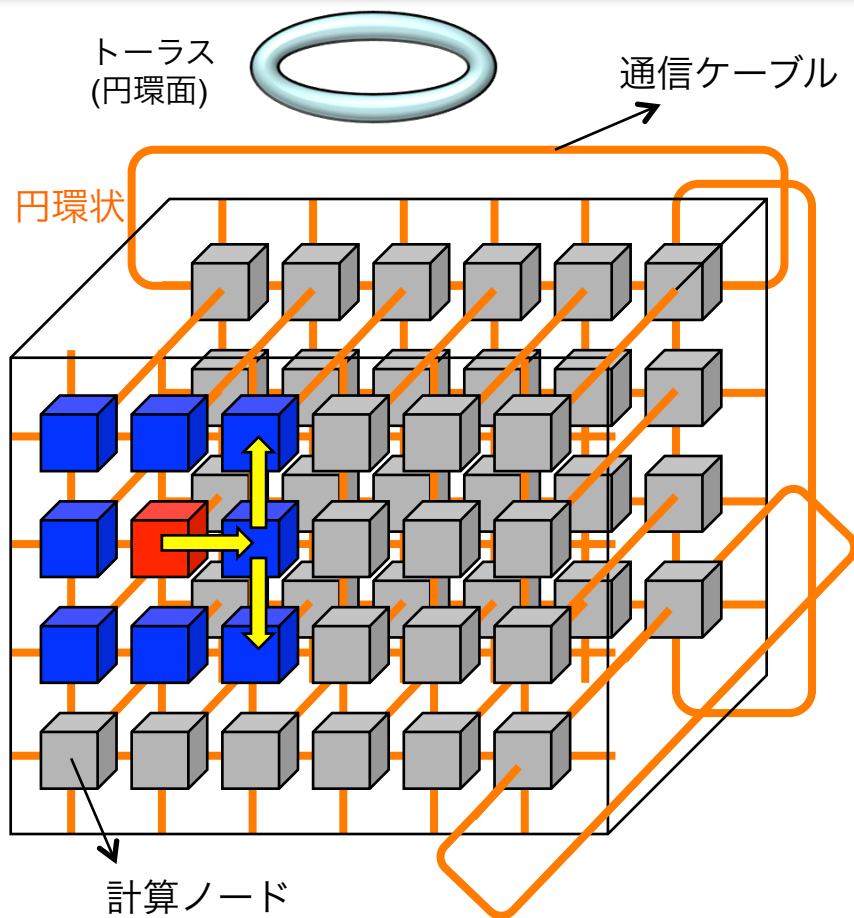
並列化技術 2 MPI

MPI 並列化技術の要点 (特に 3 次元トーラスネットワークに最適化)

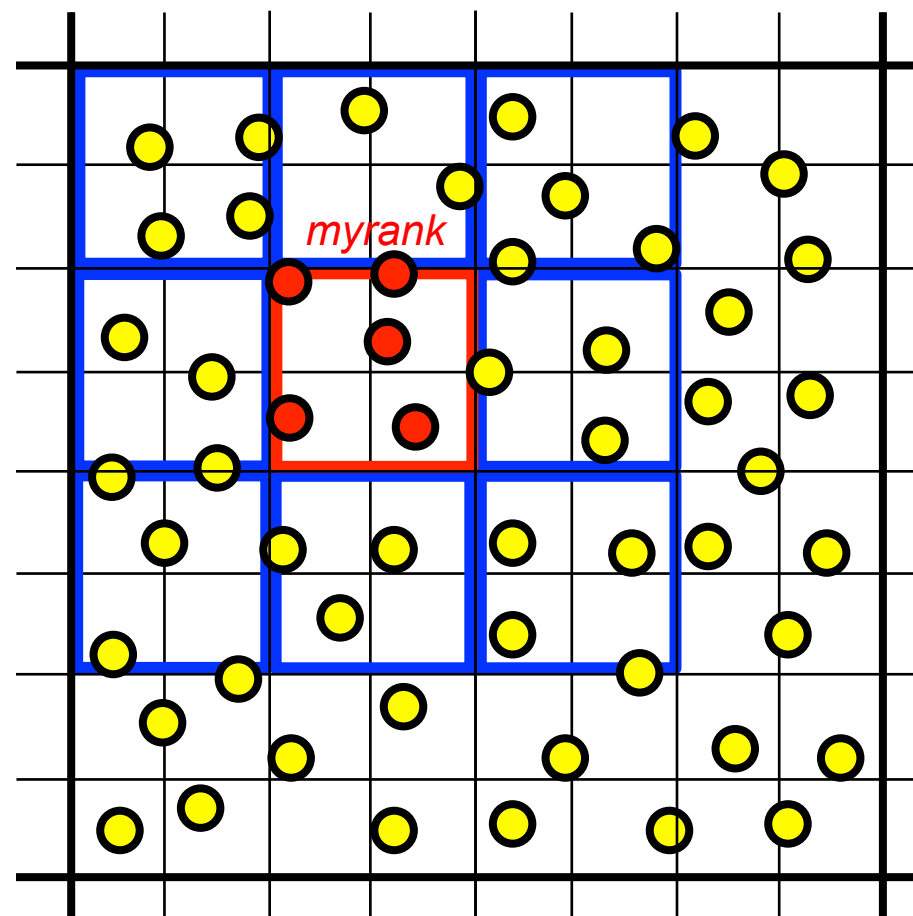
- ① 通信衝突の回避
- ② 通信前後での配列間コピーの消去
- ③ 通信の演算による代用

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 : 空間ドメイン分割によるMPI並列化との相性が良い.

短所 : 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
プロセス間通信の際に衝突が生じやすい.

MPI (message passing interface)

- ・分散メモリー型並列化のための規格
- ・MPICH, OpenMPI などのライブラリーをインストールすることで使用可
- ・「**プロセス**」と呼ばれる処理単位に仕事(タスク)が割り振られる
- ・各種の MPI 関数をコードに挿入
- ・自動並列化という概念はない

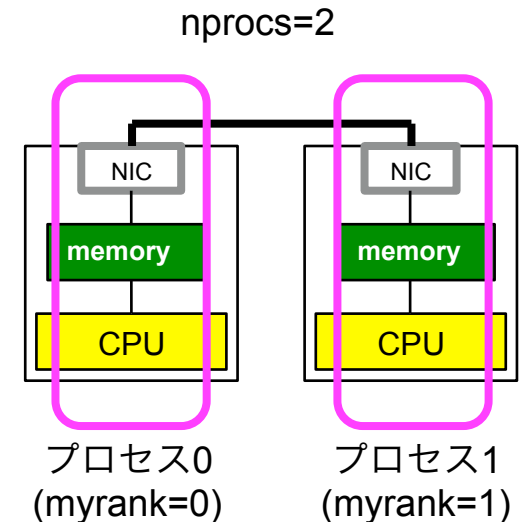


表1 代表的なMPI関数

関数名	意味
mpi_init	MPI を起動. MPI 並列化範囲の先頭を書く.
mpi_finalize	MPI を終了. MPI 並列化範囲の最後を書く.
mpi_comm_size	コミュニケータ内のプロセス数 nprocs の取得.
mpi_comm_rank	コミュニケータ内での自プロセス番号 myrank の取得.
mpi_send, mpi_recv	一対一の同期通信. _send は送信, _recv は受信.
mpi_sendrecv	一対一の同期通信. 送信受信を一度におこなう.
mpi_isend, mpi_irecv	一対一の非同期通信. _send は送信, _recv は受信.
mpi_wait	非同期通信での送受信の完了を待つ.
mpi_bcast	あるプロセスの持つ情報を他のプロセスへ配布する.
mpi_reduce, mpi_allreduce	全てのプロセス間でのリダクション処理 (合計, 最大値計算など)
mpi_gather, mpi_allgather	全てのプロセス間で情報を集約する.
mpi_wtime	時間計測用の関数

MPI (message passing interface)

サンプル hello_mpi.f:

```
include 'mpif.h'
```

MPI 変数,関数の読み込み

```
CALL MPI_INIT(ierr)          MPI 起動
```

```
CALL MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)  プロセス数 (nprocs)取得
```

```
CALL MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)  自プロセス番号 (myrank)取得  
*0 始まり
```

```
WRITE(*,*) 'Hello', myrank
```

```
CALL MPI_FINALIZE(ierr)     MPI 終了
```

```
STOP  
END
```

```
コンパイル  
> mpif90 hello_mpi.f
```

実行

```
> mpirun -np 4 ./a.out
```

```
Hello 0
```

```
Hello 1
```

```
Hello 2
```

```
Hello 3
```

mpi_sendrecv 関数

3Dトーラスネットワークでの通信に適する MPI 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

styp e : 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

rtyp e : 受信データの型

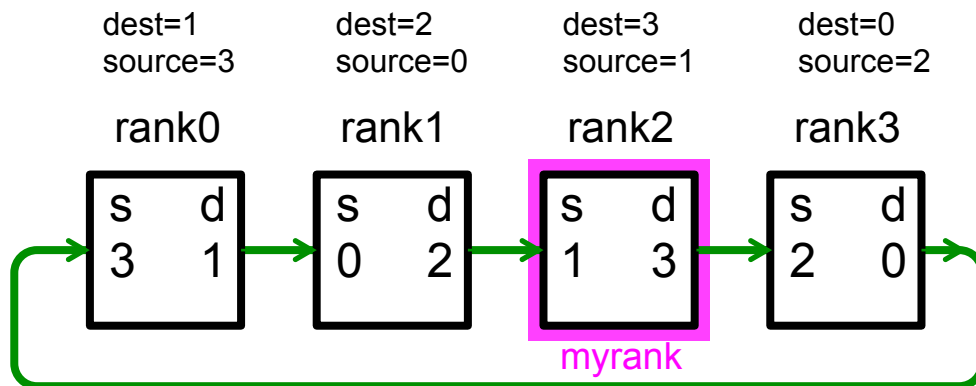
source : 受信元プロセス番号

rtag : 受信タグ

comm : コミュニケータ

status : 状態

ierr : 完了コード



→ データの流れ

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**.

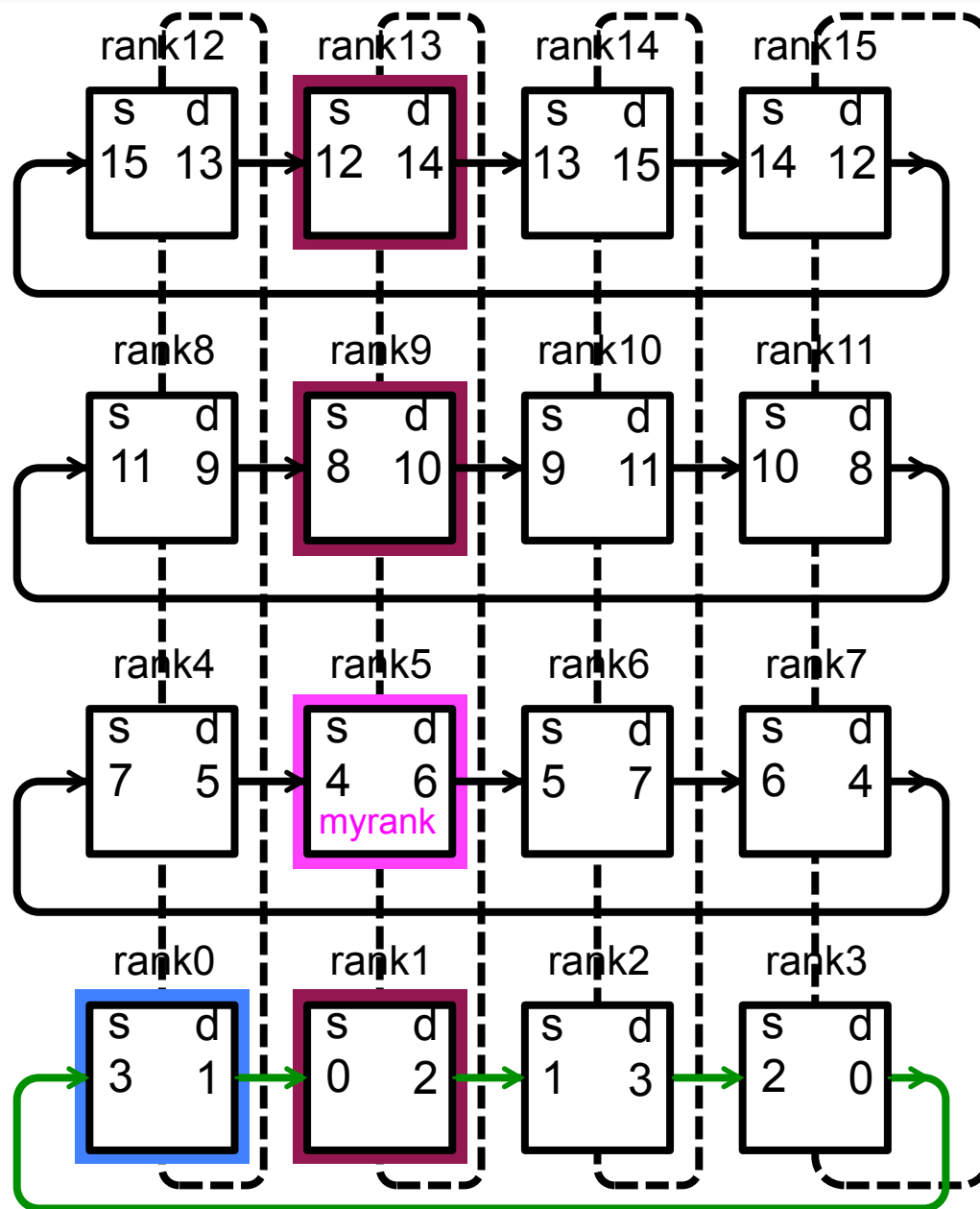
destination : 送信先
source : 送信元

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

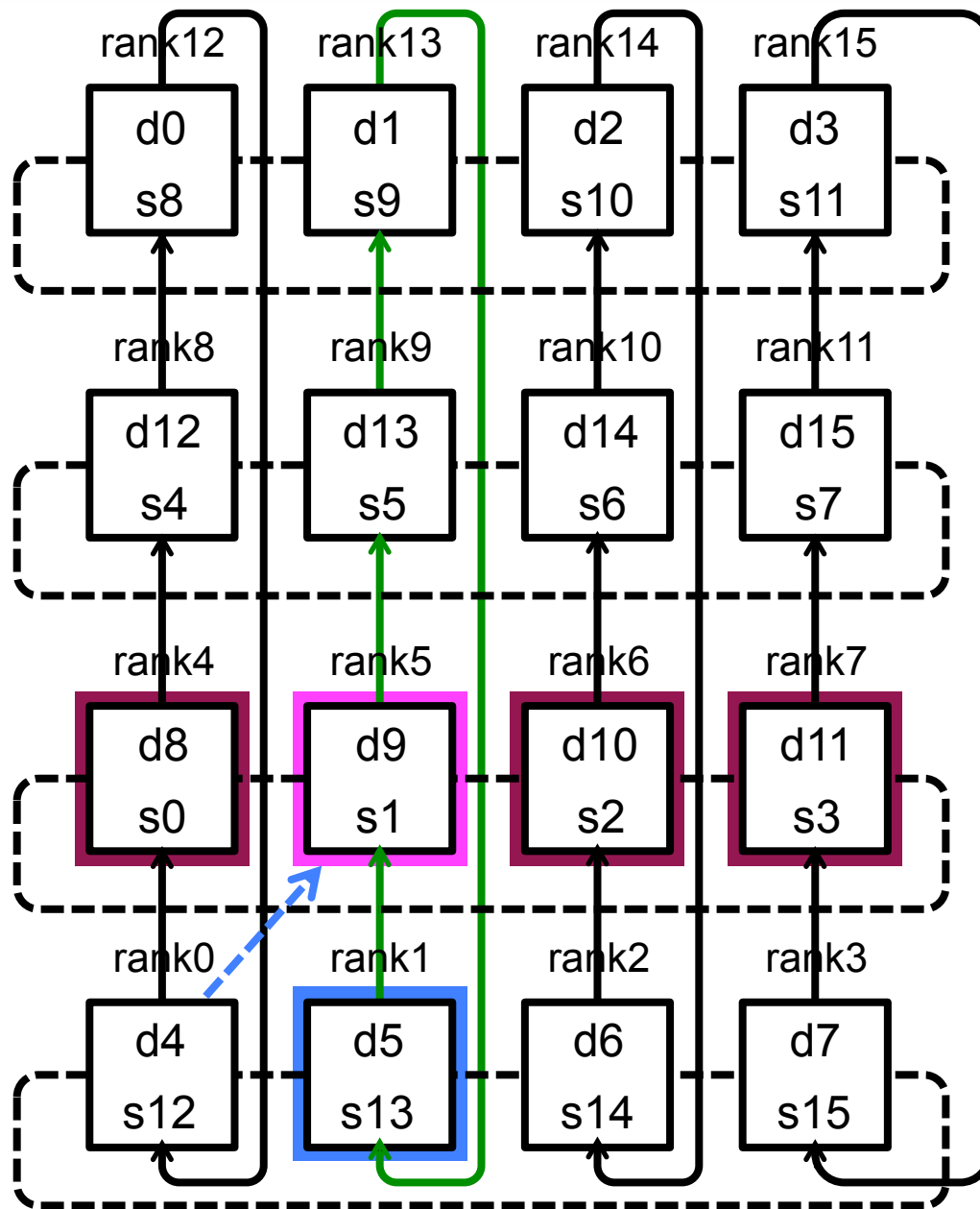
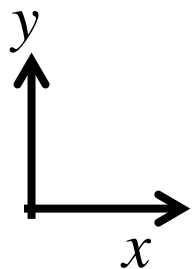
+x方向



mpi_sendrecv 関数

2次元

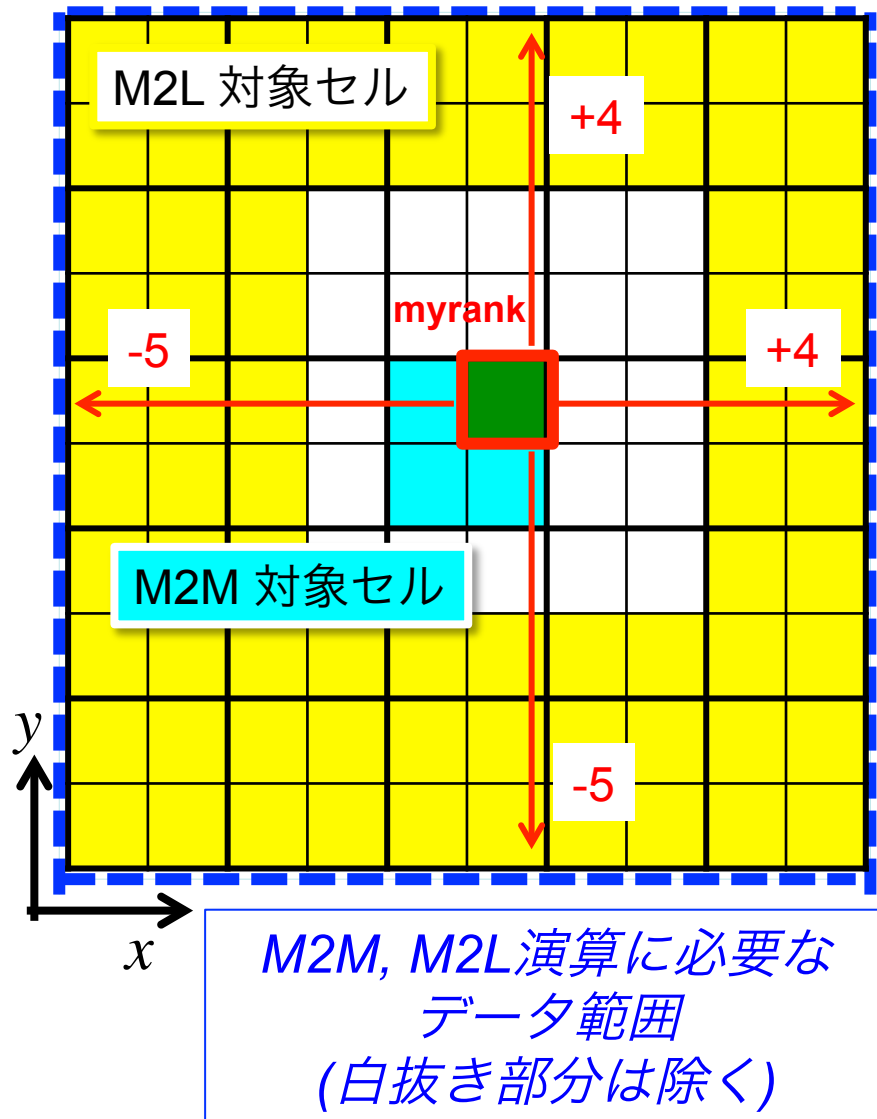
+y方向



→ データの流れ

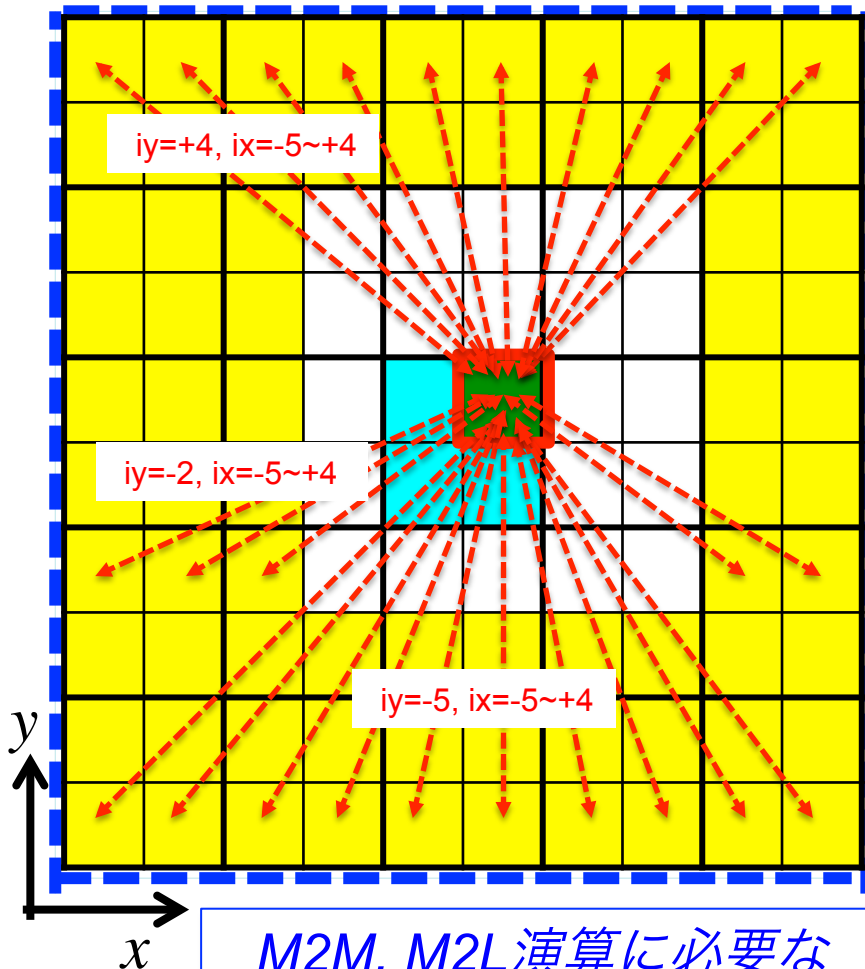
MPI 並列化技術①: 通信衝突の回避

例) 多極子の通信 (1スーパーセル/プロセス)



MPI 並列化技術①: 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)

古い通信コード

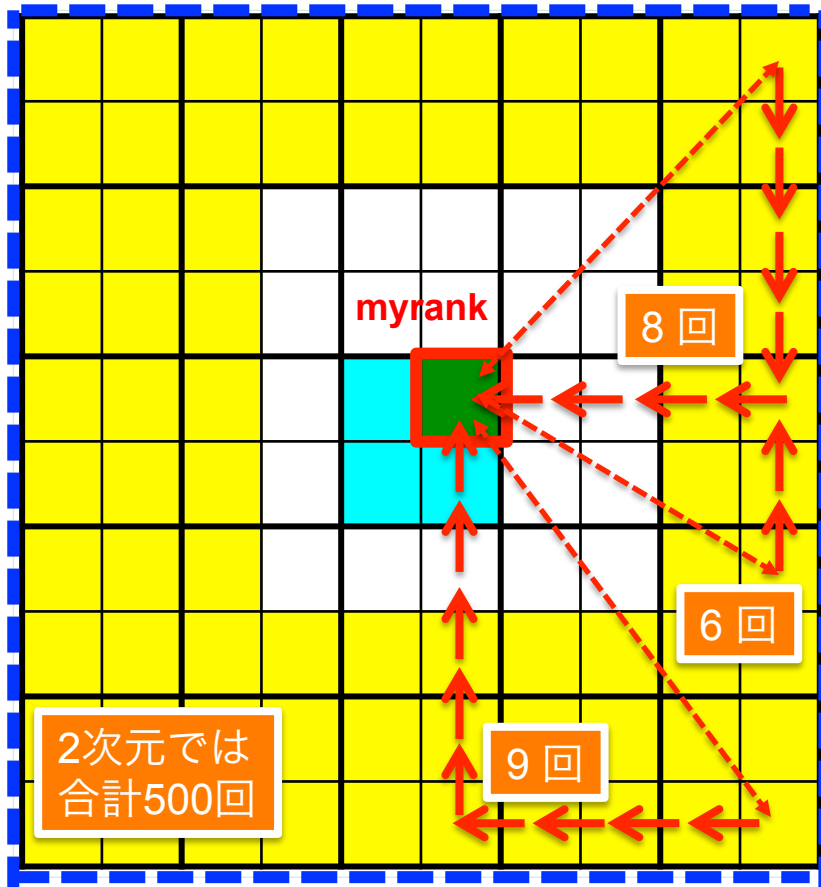
コーディングイメージ

```
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①: 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

古い通信コード

コーディングイメージ

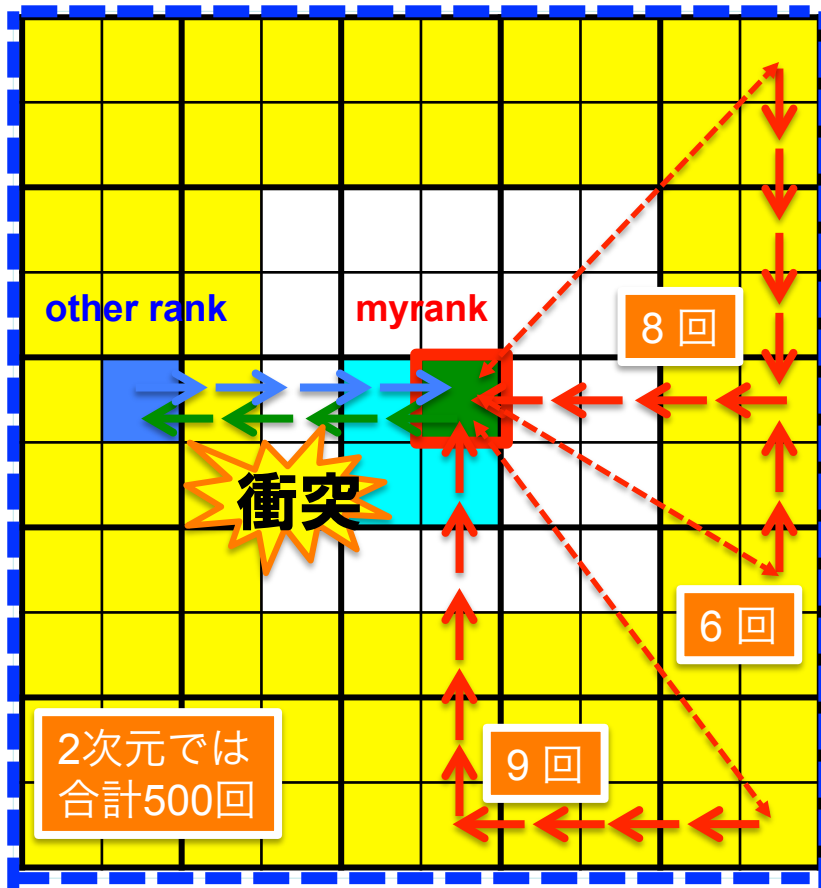
```
do iy=-5,+4
do ix=-5,+4
  ip_dest =送信先プロセス番号
  ip_src  =受信元プロセス番号
  IF( (ix, iy) が白抜き部分 ) cycle
  call mpi_sendrecv(..,ip_dest,..,ip_src,..)
enddo
enddo
```

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]**

さらに、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①: 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

古い通信コード

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]**

さらに、トーラスネットワークに特有の問題:

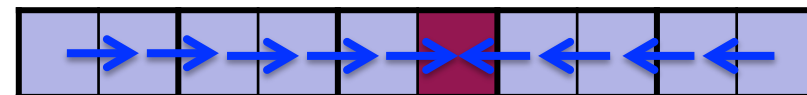
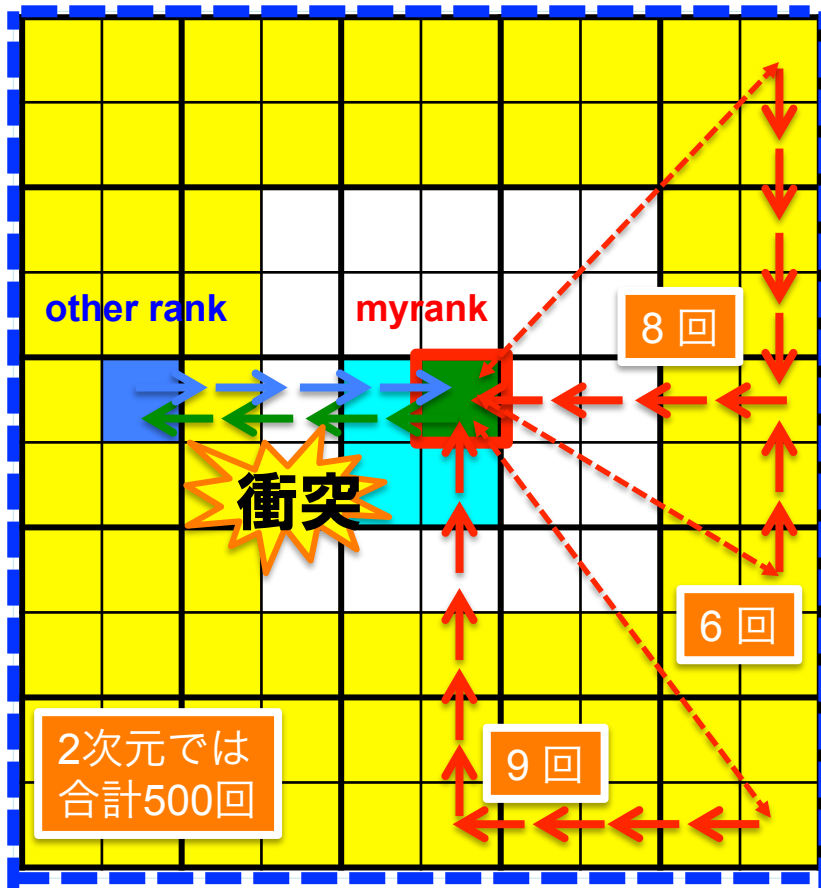
- (1) 882個の通信先ごと多数のホップ回数
- (2) 通信の衝突が至る所で発生

➡ 通信時間の増加 = 並列性能の低下

MPI 並列化技術①: 通信衝突の回避

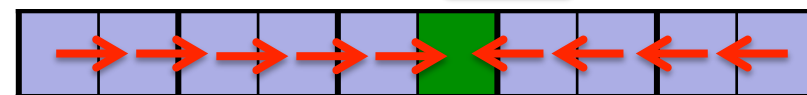
例) 多極子の通信 (1スーパーセル/プロセス)

新しい通信コード



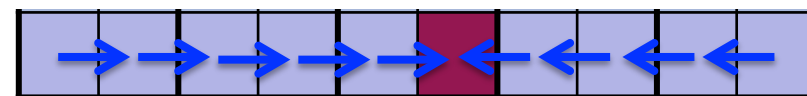
1) $\pm x$ 軸方向通信

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

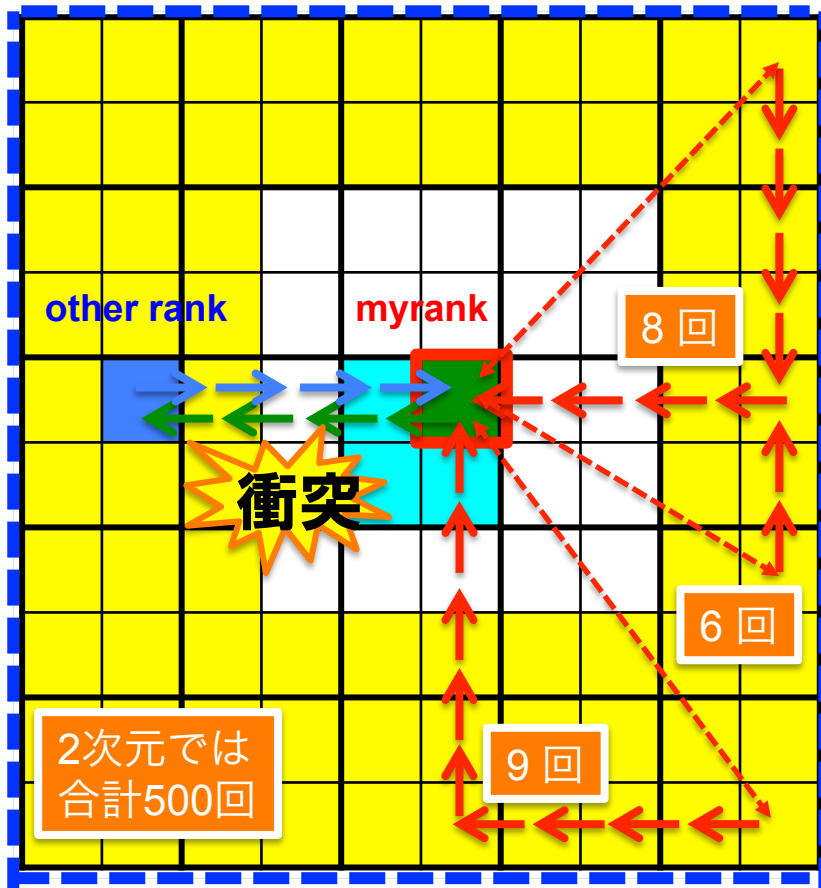


トーラスネットワークに特有の問題:

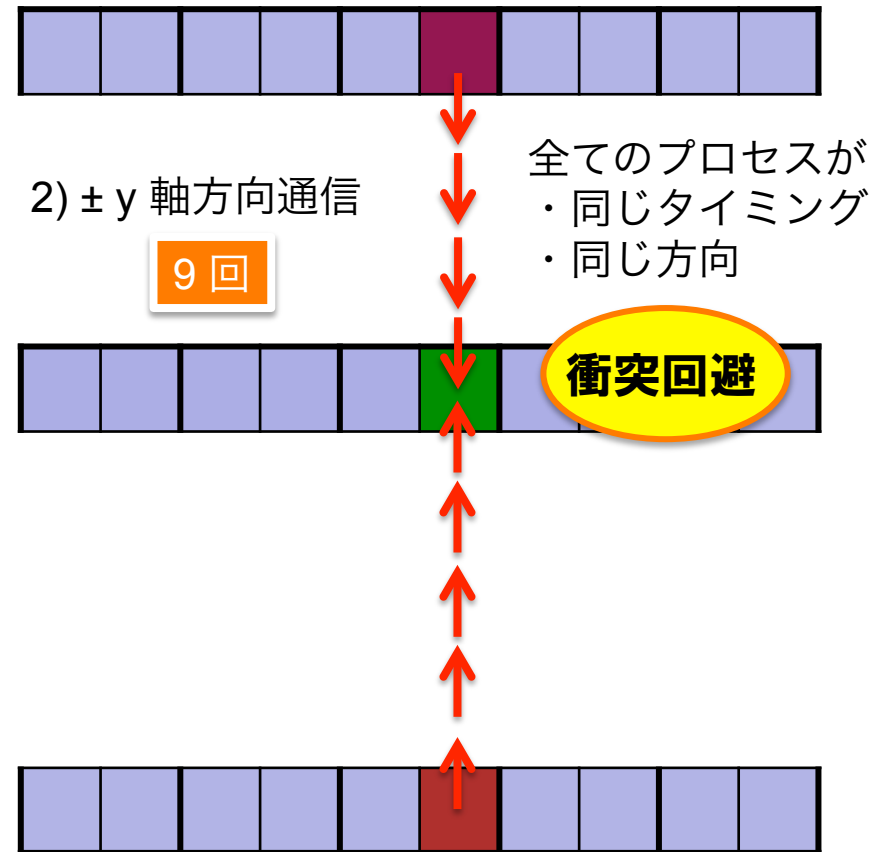
- (1) 882個の通信先ごと多数のホップ回数
- (2) 通信の衝突が至る所で発生

MPI 並列化技術①: 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



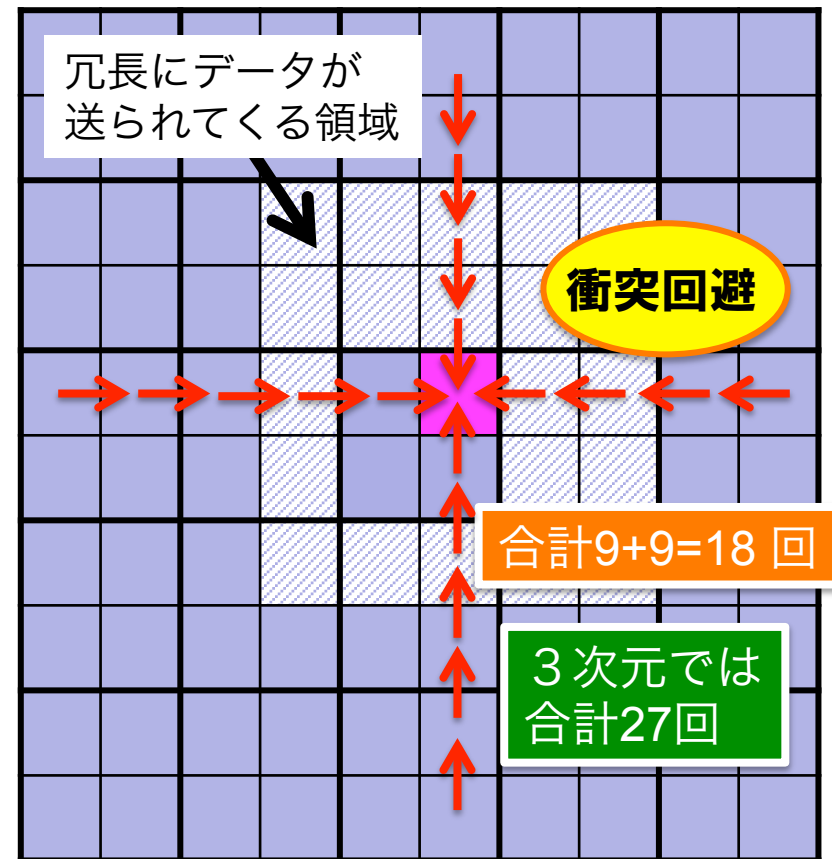
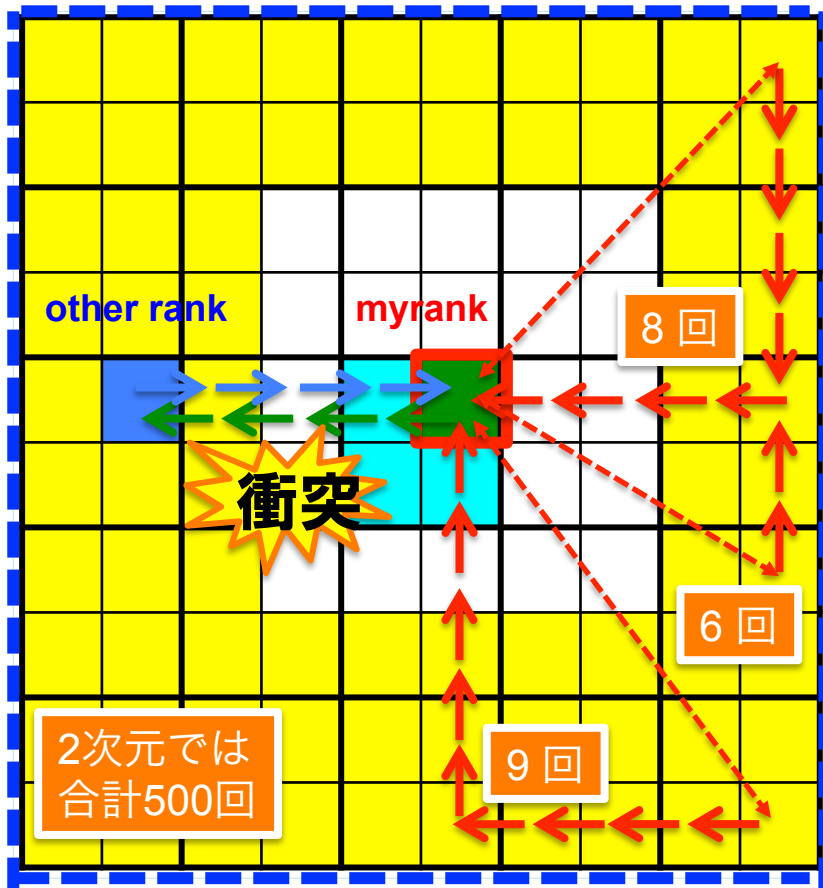
トーラスネットワークに特有の問題:

- (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①: 通信衝突の回避

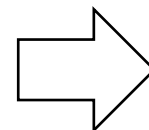
例) 多極子の通信 (1スーパーセル/プロセス)

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抑える
- (2) 通信の衝突を回避

MPI 並列化技術①: 通信衝突の回避

x方向通信コーディングイメージ

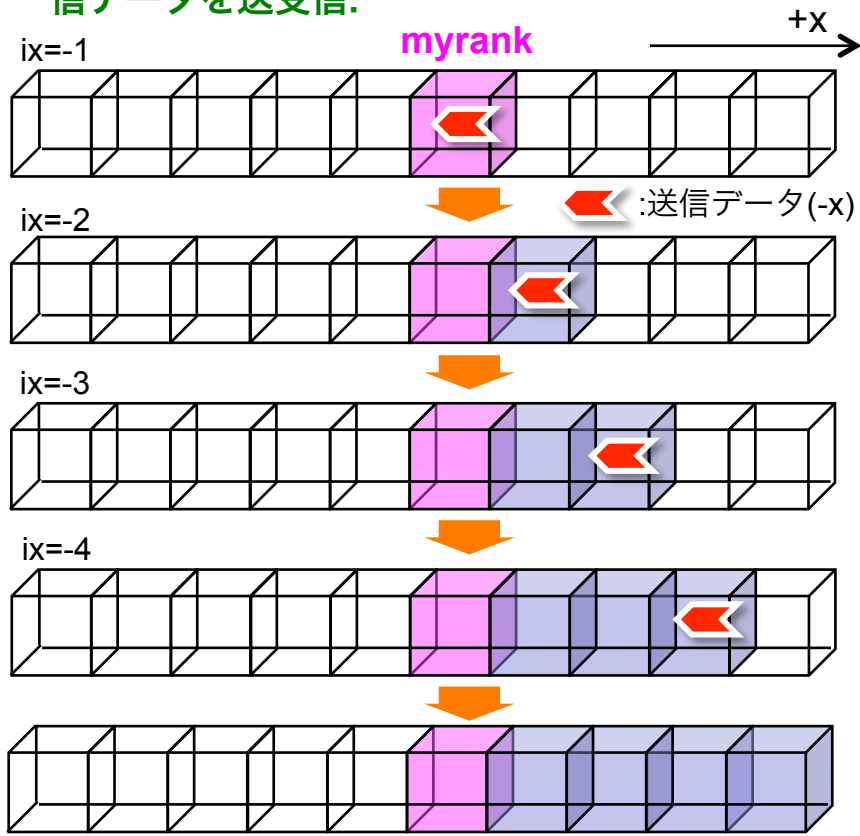
```

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①: 通信衝突の回避

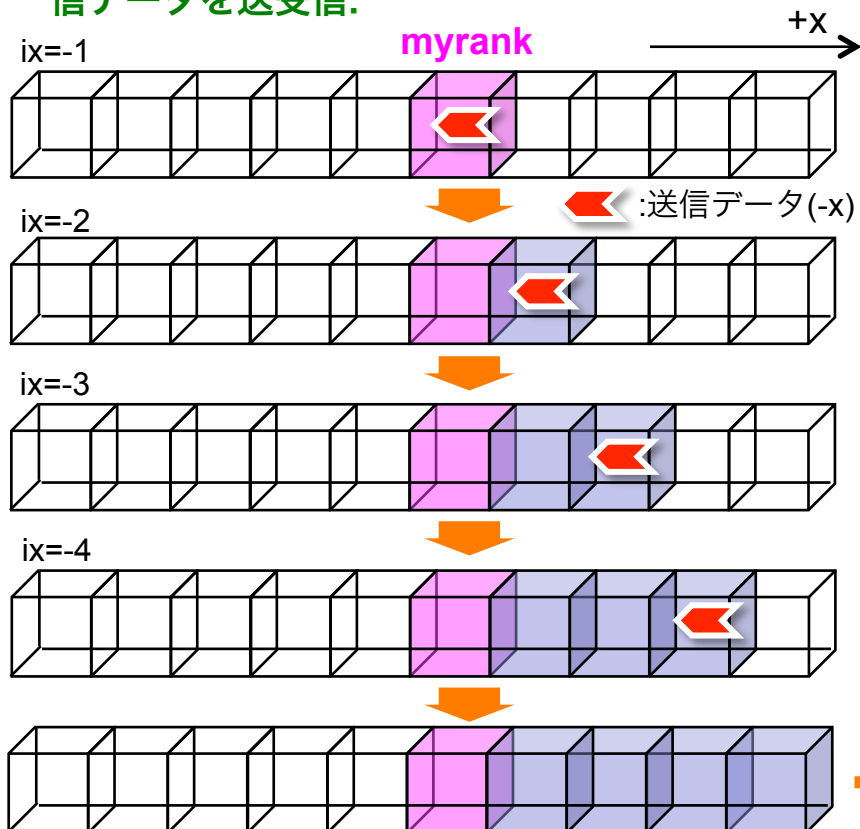
x方向通信コーディングイメージ

```

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

```

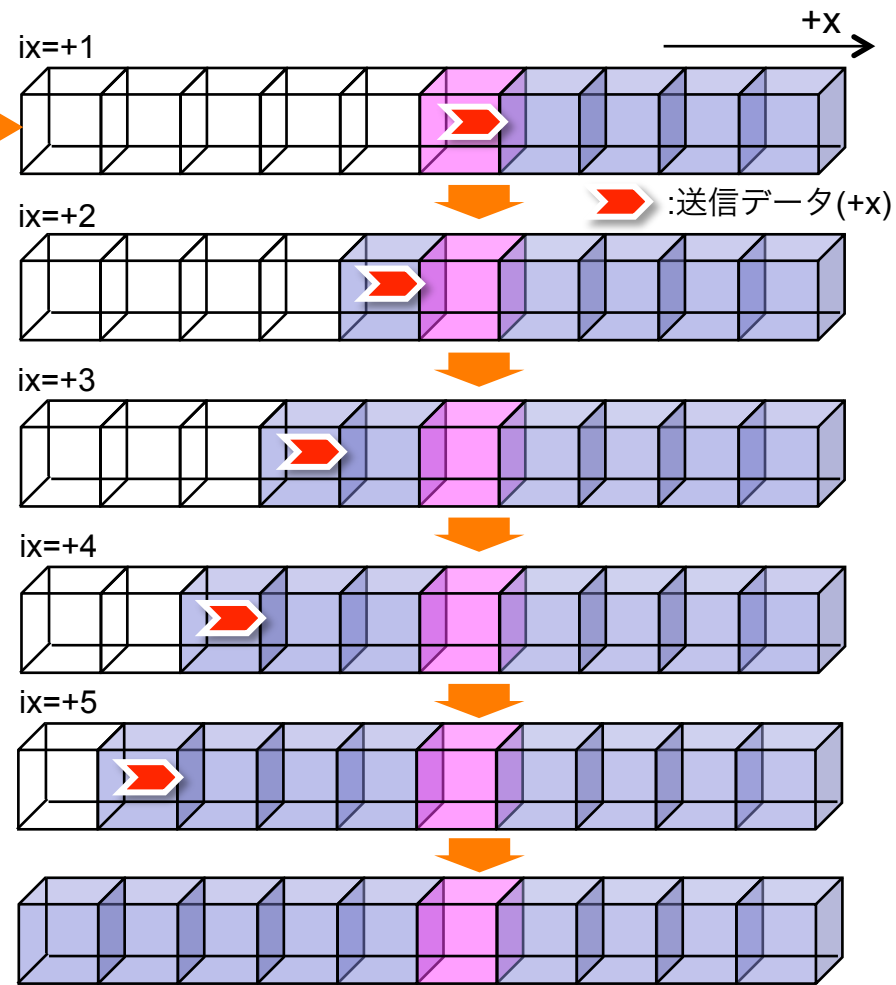
袖部つき局所化されたデータ構造の所定位置に受信データを格納。次回通信では前回の受信データを送受信。



```

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①: 通信衝突の回避

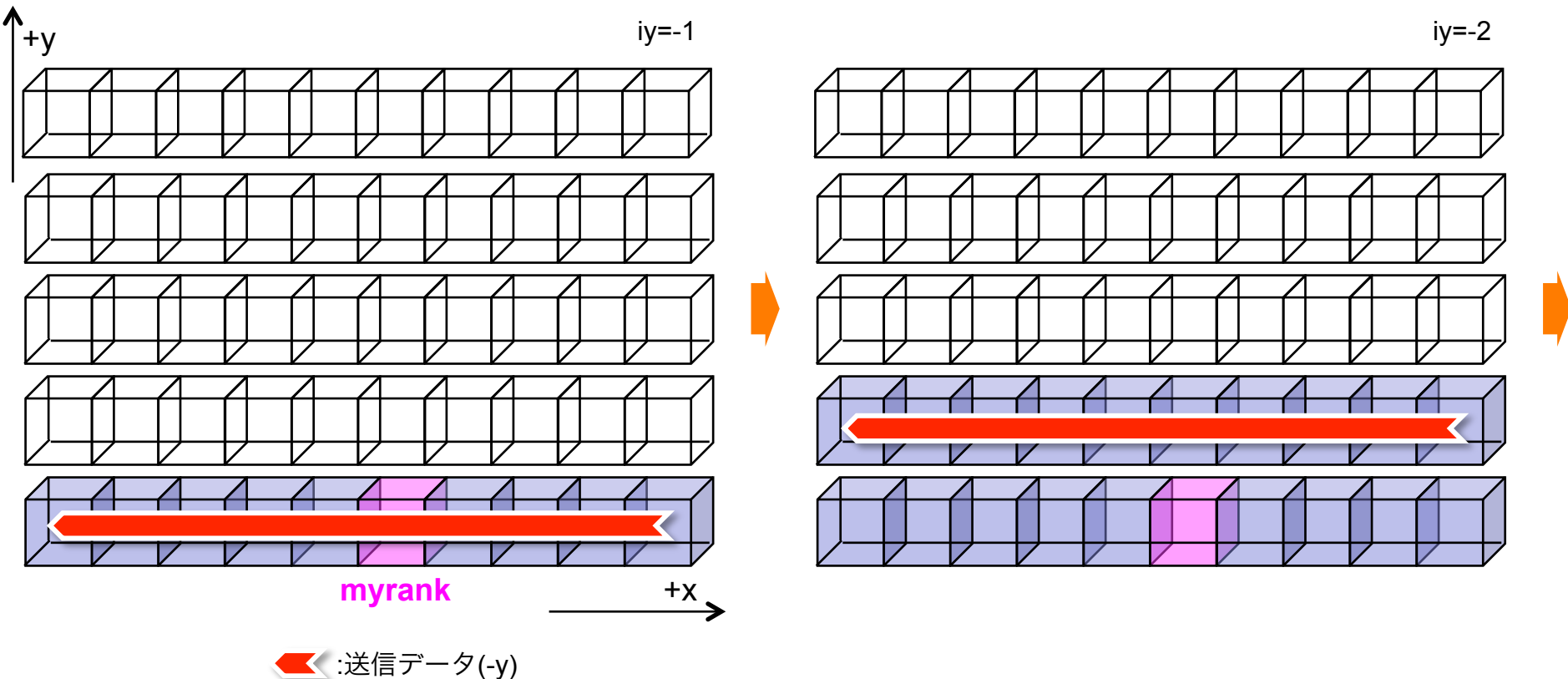
y方向通信コーディングイメージ

```

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①: 通信衝突の回避

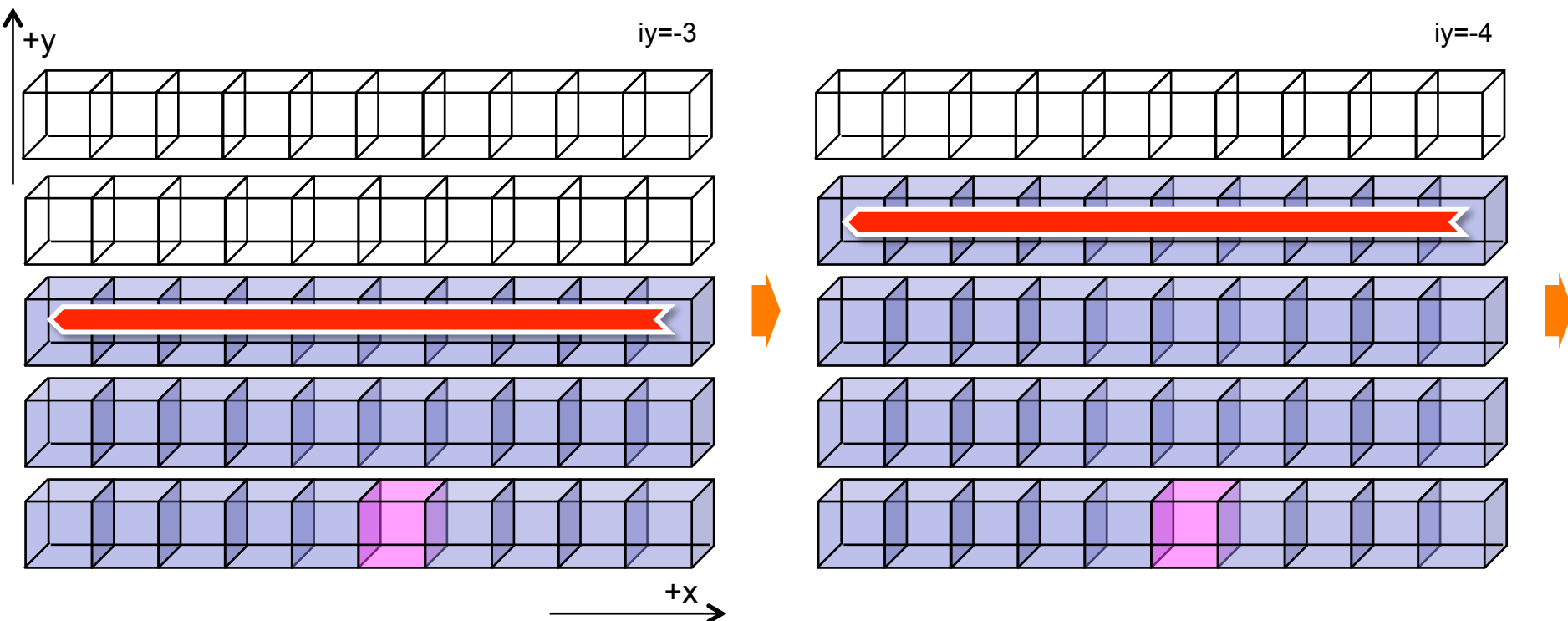
y方向通信コーディングイメージ

```

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①: 通信衝突の回避

y方向通信コーディングイメージ

```

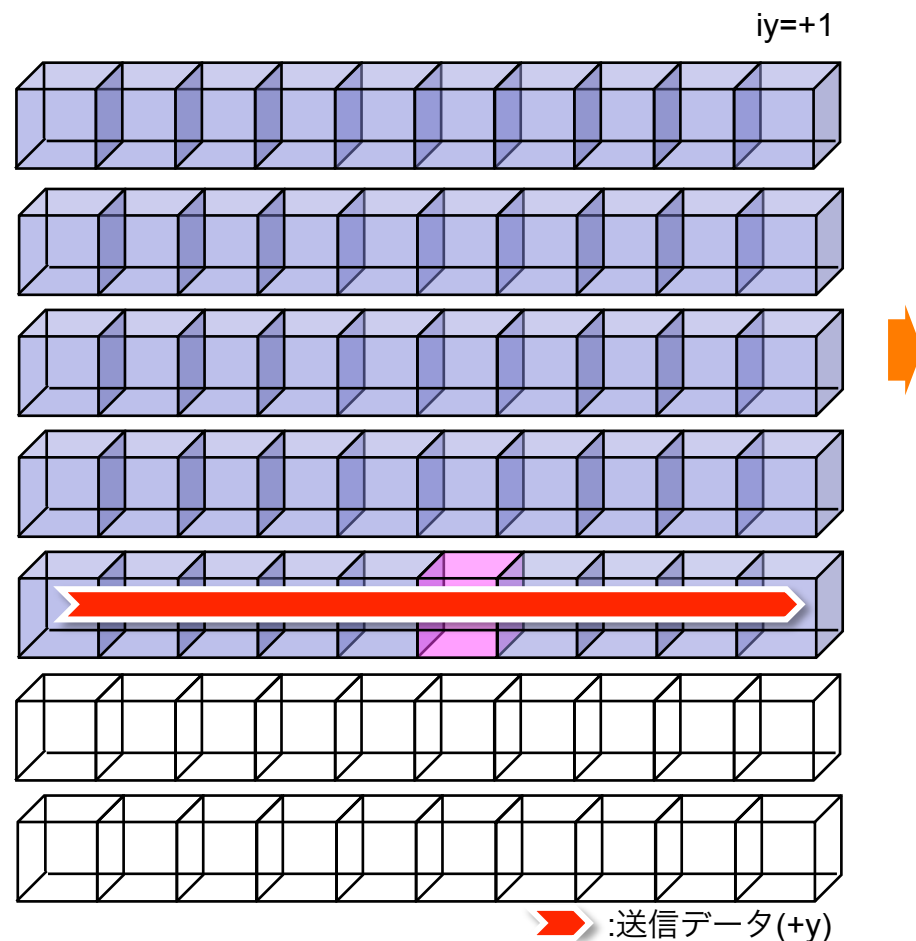
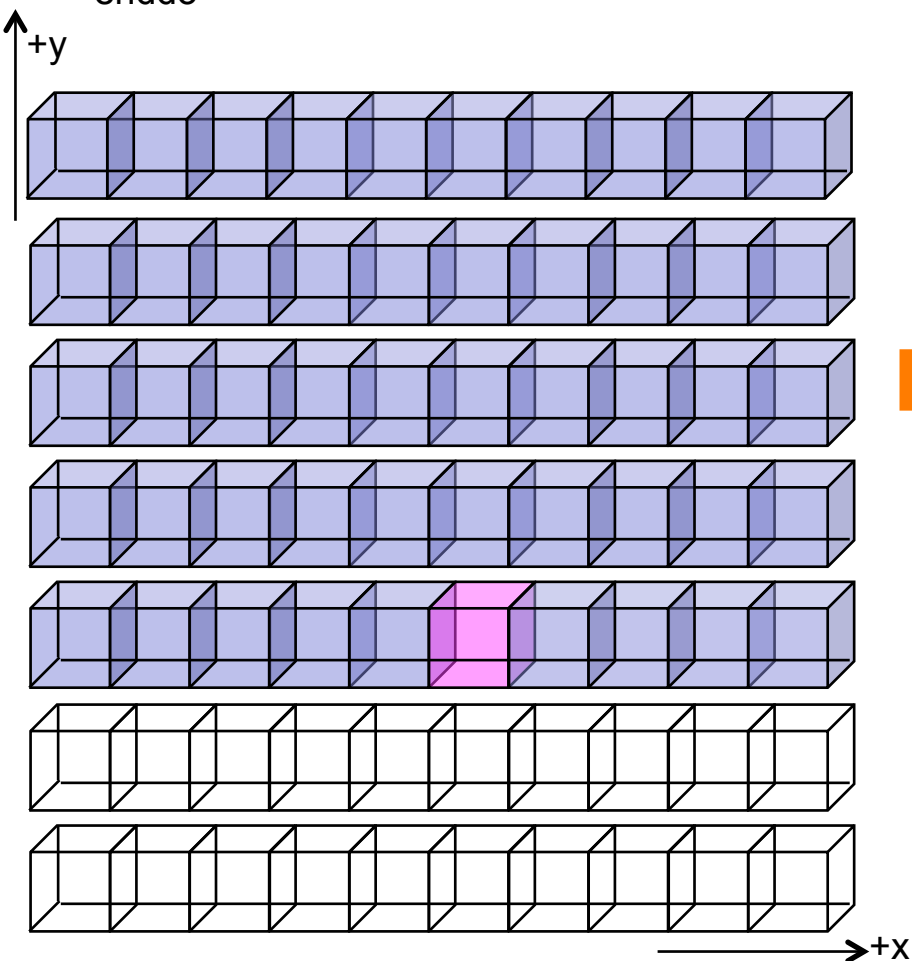
ipy_dest =送信先プロセス番号(-y)
ipy_src  =受信元プロセス番号(-y)
do iy= -1, -4, -1
  call mpi_sendrecv(...,ipy_dest,...,ipy_src,...)
enddo

```

```

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①: 通信衝突の回避

y方向通信コーディングイメージ

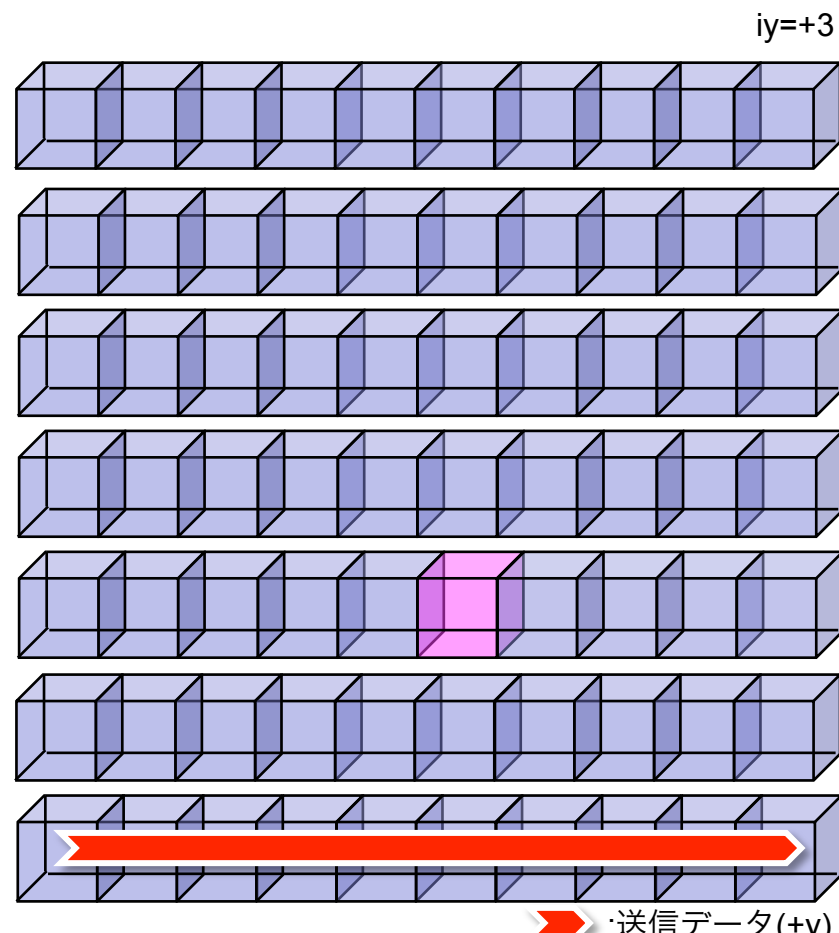
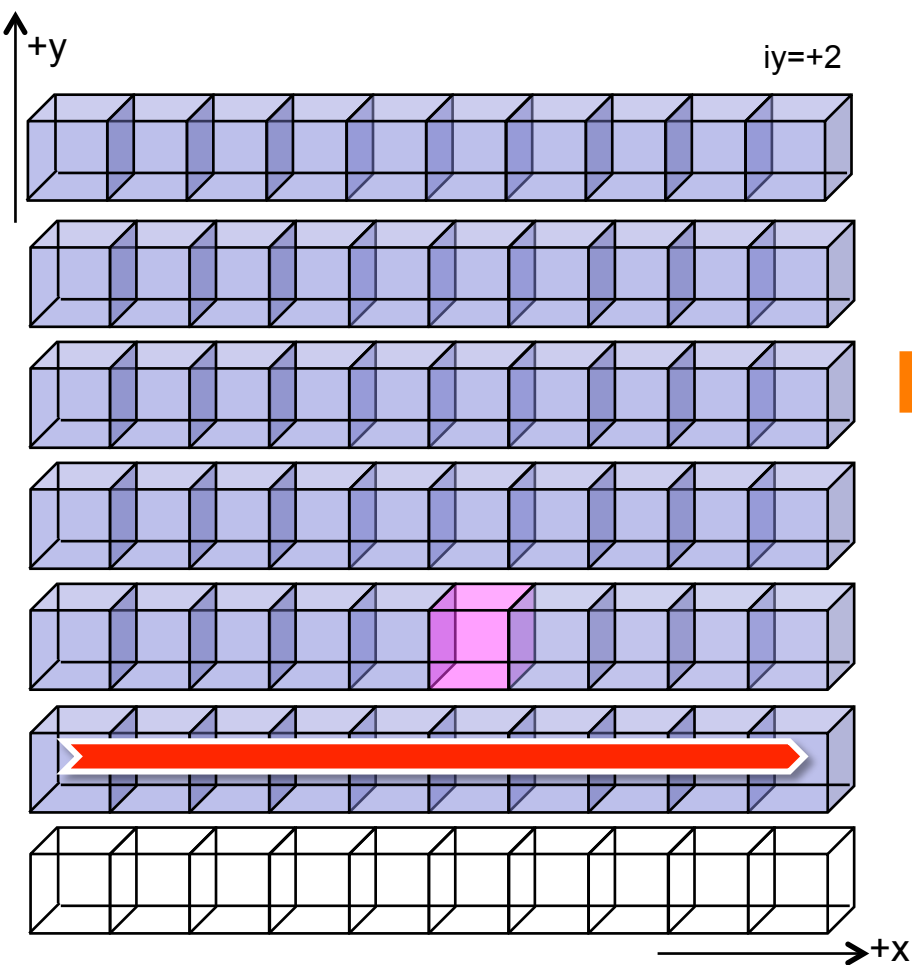
ipy_dest = 送信先プロセス番号(+y)

ipy_src = 受信元プロセス番号(+y)

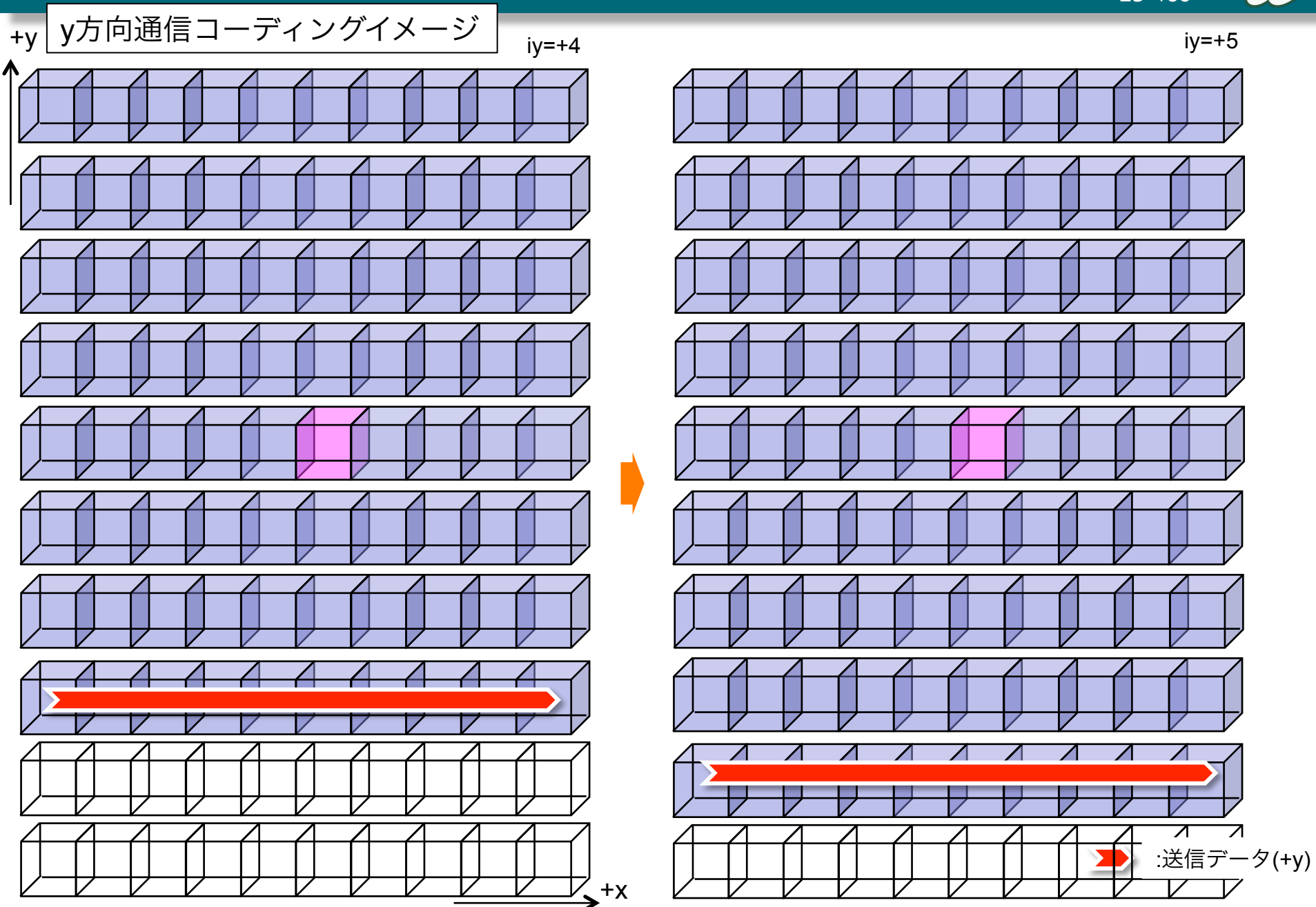
do iy= +1, +5, +1

call mpi_sendrecv(...,ipy_dest,...,ipy_src,...)

enddo



MPI 並列化技術①: 通信衝突の回避



MPI 並列化技術①: 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) } 固定
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

```

-x

```

ipx_dest =送信先プロセス番号(+x) } 固定
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

```

+x

```

ipy_dest =送信先プロセス番号(-y) } 固定
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

```

-y

```

ipy_dest =送信先プロセス番号(+y) } 固定
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

```

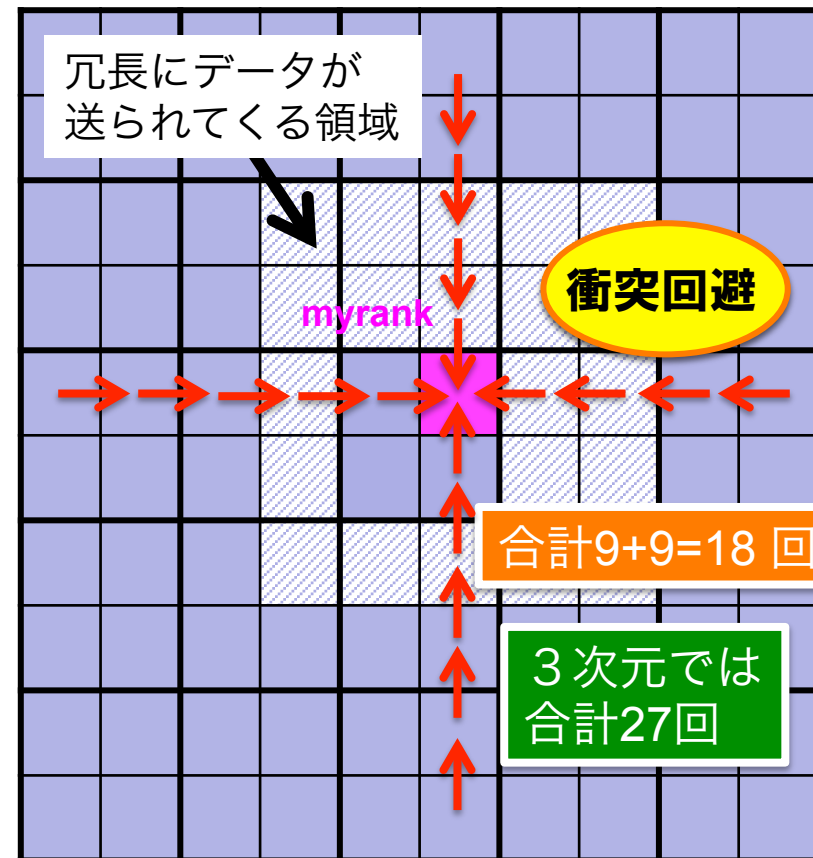
+y

残るz方向も同様

-z

+z

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



旧データ構造

新データ構造 (メタデータ・袖部付き局所化)

演算部

wk_x

meta_x

通信対象を抽出/頭詰め
通信バッファへコピー



最初に通信する1軸方向
のみ最小限のコピー



通信部

trans_x

meta_x

受信バッファへ格納
元の配列末尾へ格納



最初に通信する1軸方向
のみ最小限のコピー



演算部

wk_x

meta_x

MPI 並列化技術②: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

```

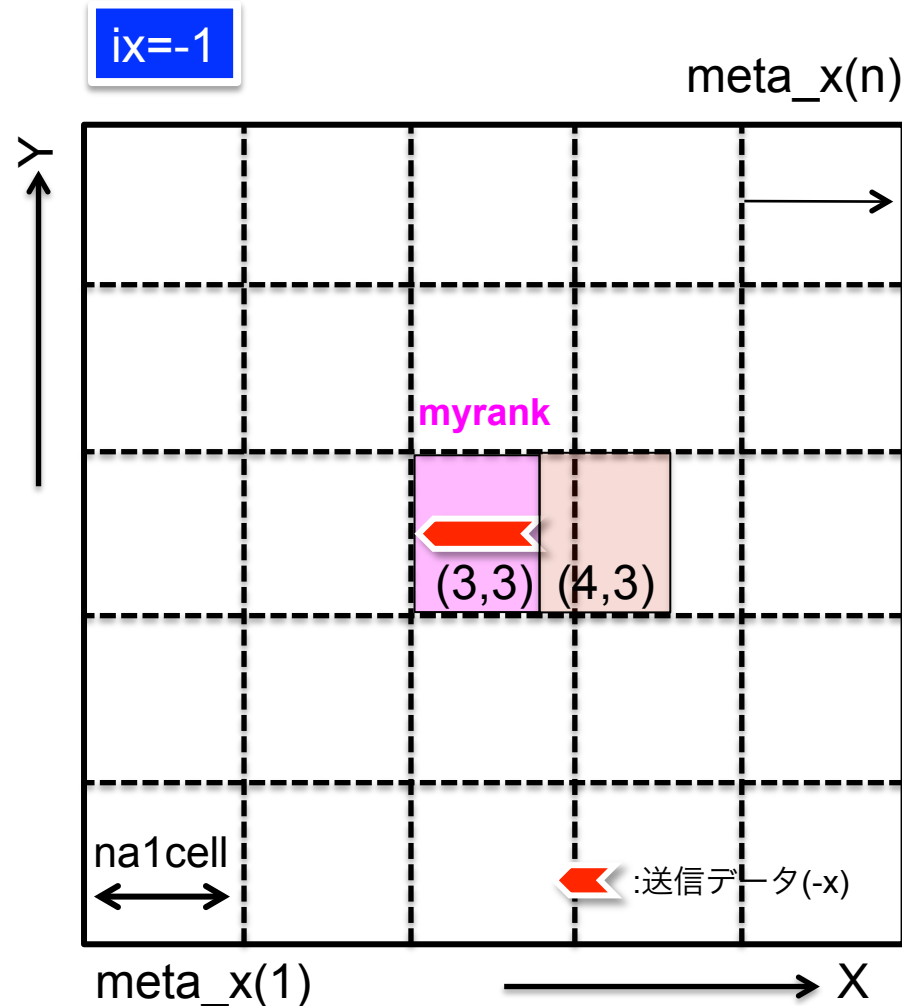
ipx_mdest =送信先プロセス番号(-x)
ipx_msrc=受信元プロセス番号(-x)
do ix=-1,-2,-1
c 原子数情報の送受信
  icbufm( ) = na_per_cell( 3, 3 )
  call mpi_sendrecv(icbufm, ...,ipx_mdest, ...,
                    ircbufm,..., ipx_msrc, ...)
  na_per_cell( 4, 3 )=ircbufm( )
  受信データのtag(4,3)を作成
c 座標情報の送受信
  buffm( ) = meta_x( ) ← 送信バッファへコピー
  call mpi_sendrecv(buffm,...,ipx_mdest,...,
                    rbuffm, ...,ipx_msrc,..)
  meta_x( ) = rbuffm( ) ← データを左づめにしつつ元配列にコピー
enddo

```

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

-x 軸方向通信

[サイズ不定のセル単位データを送受信]





MPI 並列化技術②: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

```

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
    
```

送信バッファへコピー

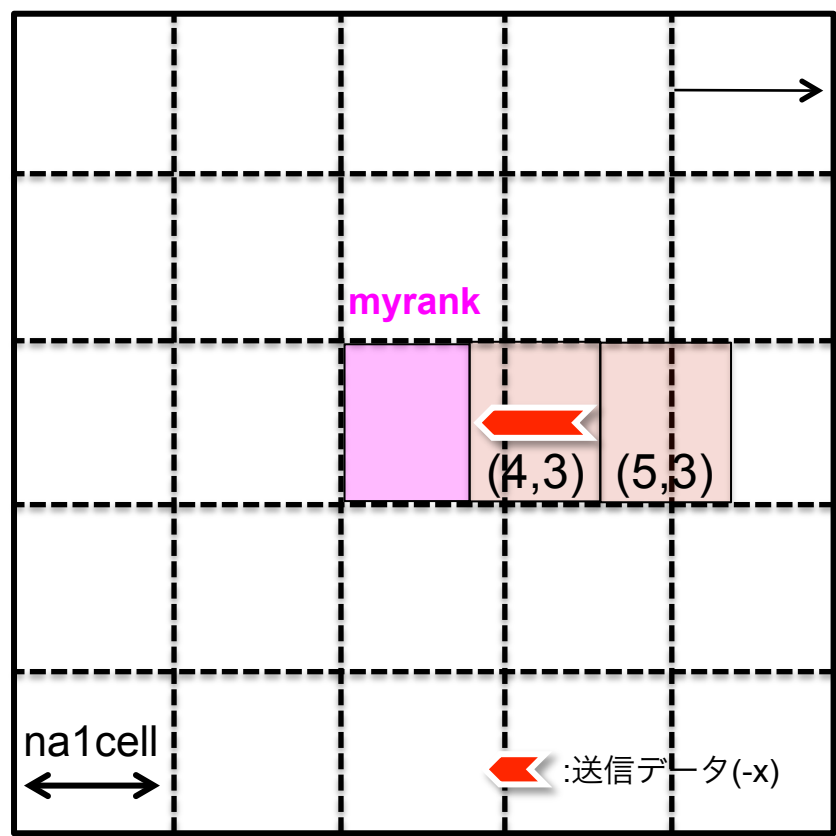
データを左づめにしつつ元配列にコピー

-x 軸方向通信

[サイズ不定のセル単位データを送受信]

ix=-2

↑ Y



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②: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, ...,  
irbufp, ..., ipx_psrc, ...)
```

```
na_per_cell( 2, 3 )=irbufp( )
```

```
受信データのtag(2,3)を作成
```

```
c 座標情報の送受信
```

```
bufp( ) = meta_x( )
```

```
call mpi_sendrecv(bufp, ...,ipx_pdest, ...,  
rbufp, ...,ipx_psrc, ...)
```

```
meta_x( ) = rbufp( )
```

```
enddo
```

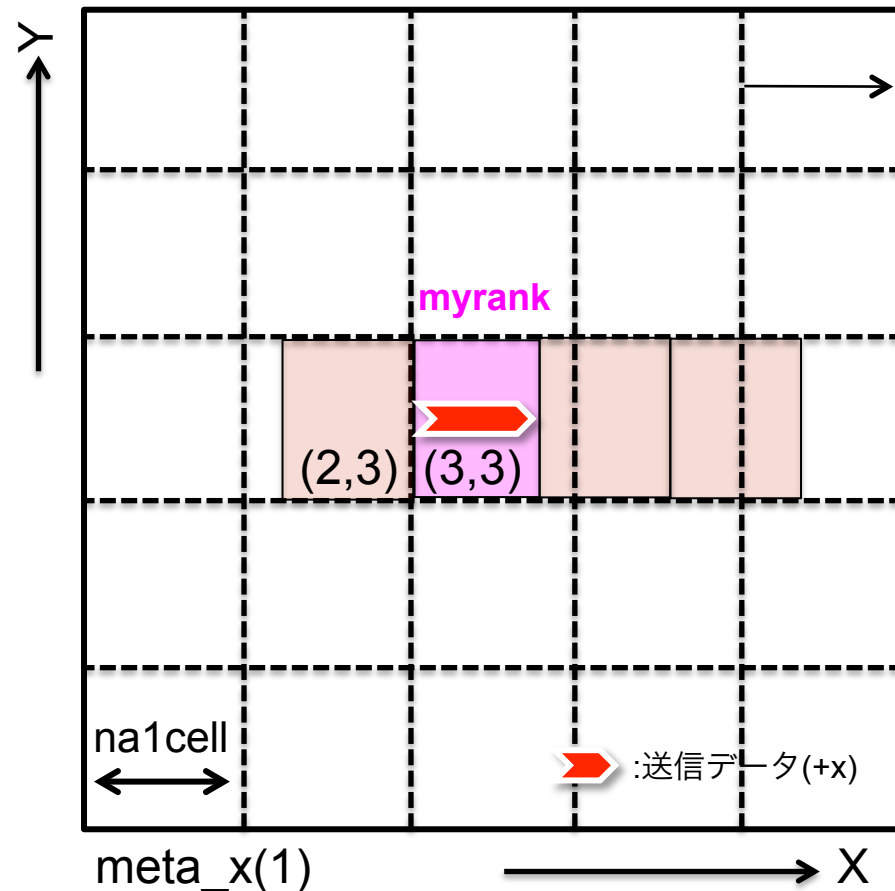
送信バッファへ
コピー

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

ix=+1

meta_x(n)



MPI 並列化技術②: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, ...,  
irbufp, ..., ipx_psrc, ...)
```

```
na_per_cell( 1, 3 )=irbufp( )
```

```
受信データのtag(1,3)を作成
```

```
c 座標情報の送受信
```

```
bufp( ) = meta_x( )
```

```
call mpi_sendrecv(bufp, ...,ipx_pdest, ...,  
rbufp, ...,ipx_psrc, ...)
```

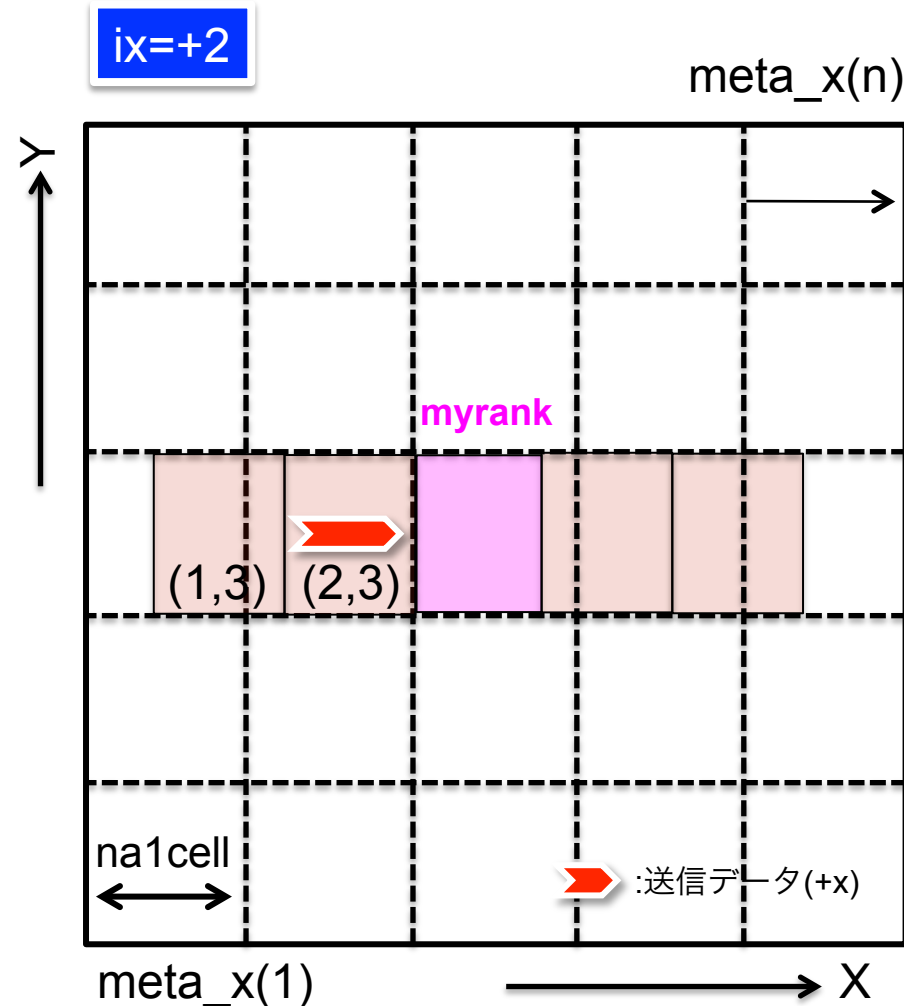
```
meta_x( ) = rbufp( )
```

```
enddo
```

送信バッファへ
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②: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

```

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
    
```

棒状データを構成するサブセル数を指定

```

call mpi_sendrecv(na_per_cell(1,3),5,...,ipy_mdest, ...,
                 na_per_cell(1,4),5,..., ipy_msrc, ...)
    
```

(1,3)セルの先頭アドレスを指定

```

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,naline),...,ipy_mdest,...,
                 meta_x(1,naline),...,ipy_msrc,...)
    
```

(1,4)セルの先頭アドレスを指定

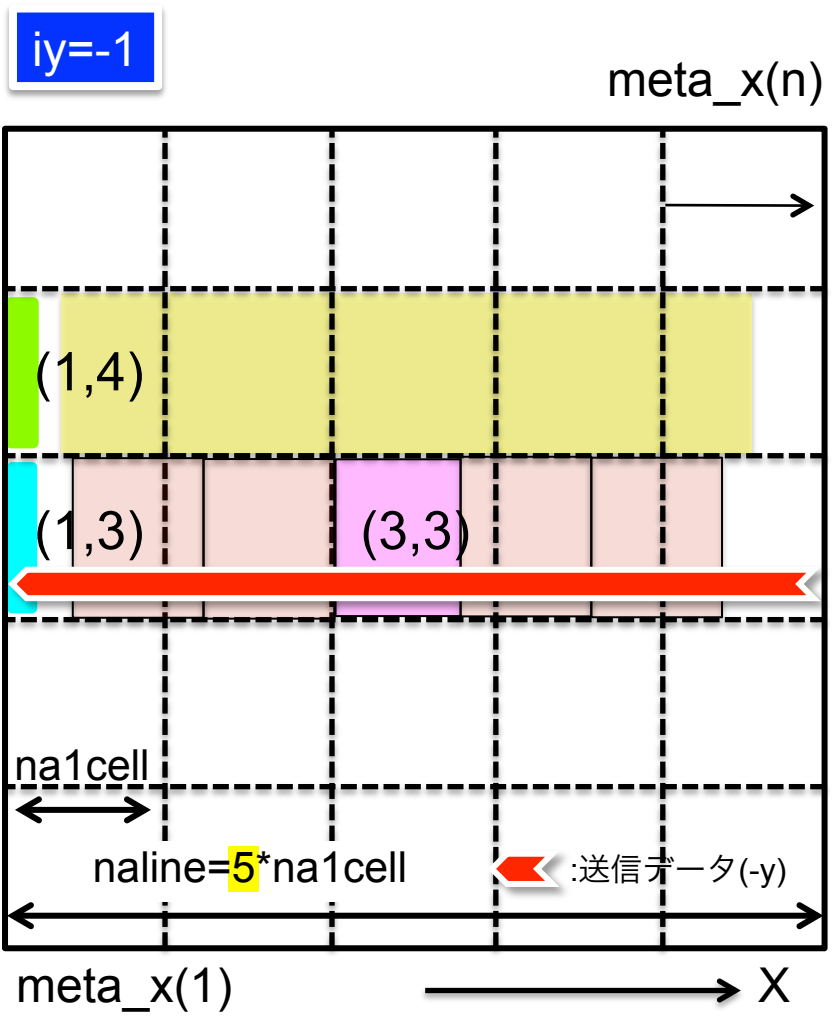
```

enddo
    
```

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

-y 軸方向通信

[サイズ一定の棒状データを送受信]



MPI 並列化技術②: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,naline),...,ipy_pdest,...,
meta_x(1,naline),...,ipy_psrc,...)
```

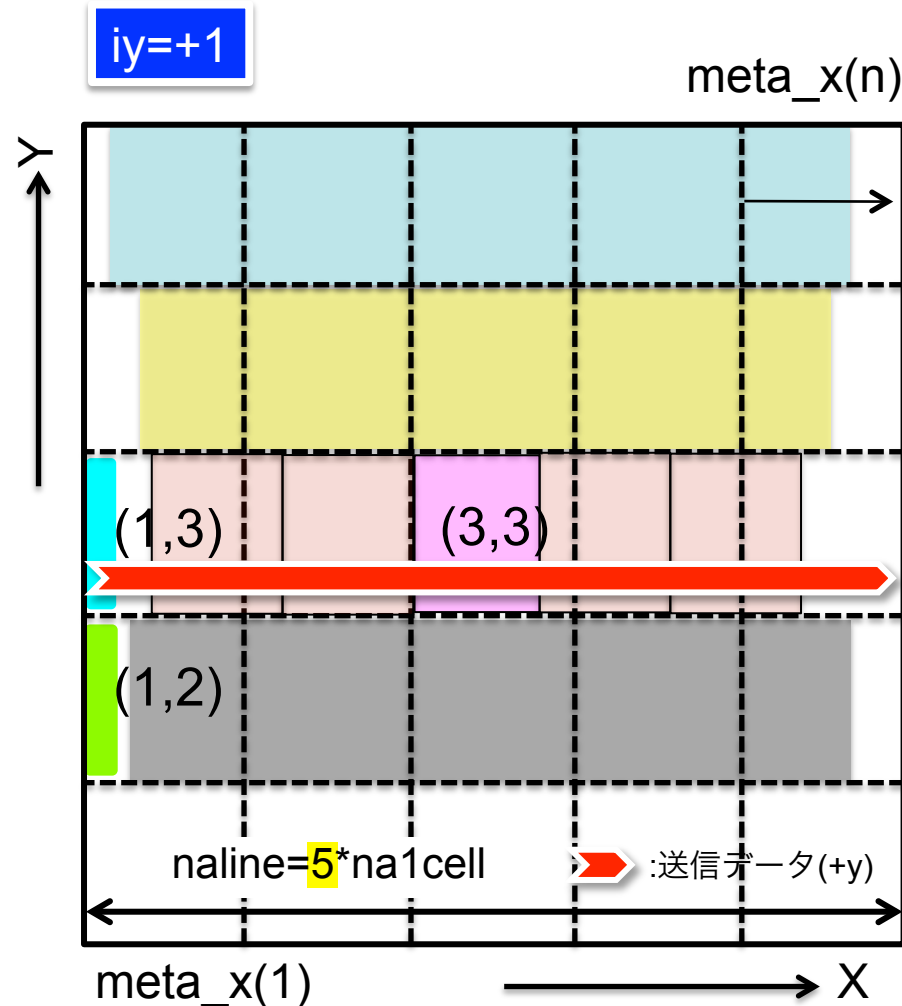
```
enddo
```

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②: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,naline),...,ipy_pdest,...,
meta_x(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

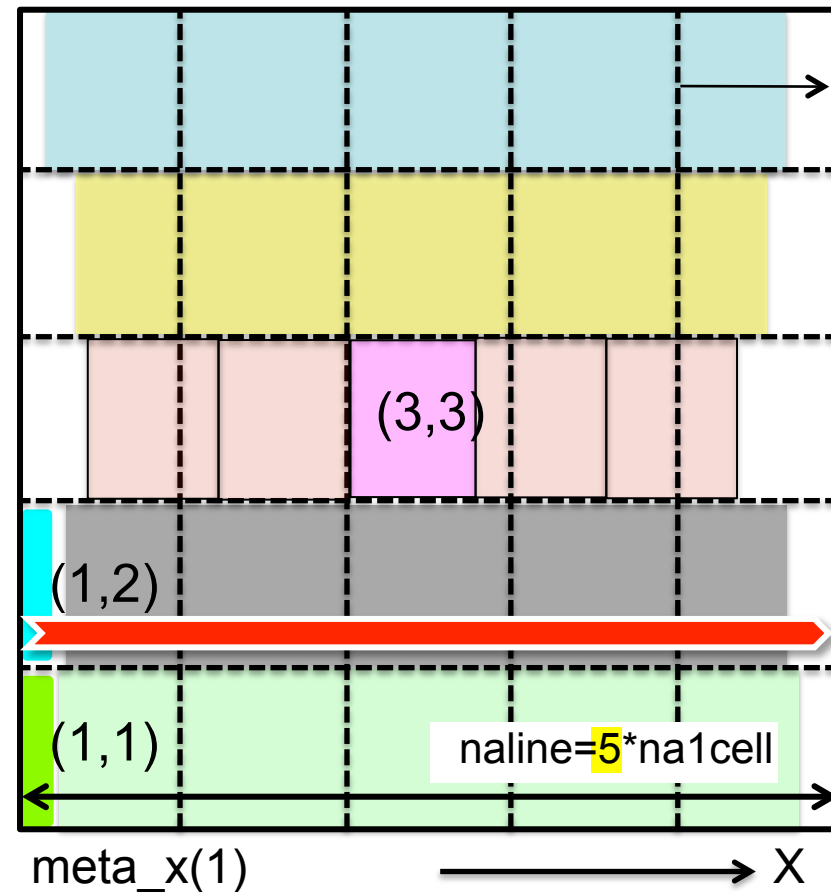
(1,1)セルの先頭
アドレスを指定

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

↑ Y

iy=+2

meta_x(n)



➡ :送信データ(+y)

MPI 並列化技術②: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( ),25,...,ipz_mdest, ...,
na_per_cell( ),25,..., ipz_msrc, ...)
```

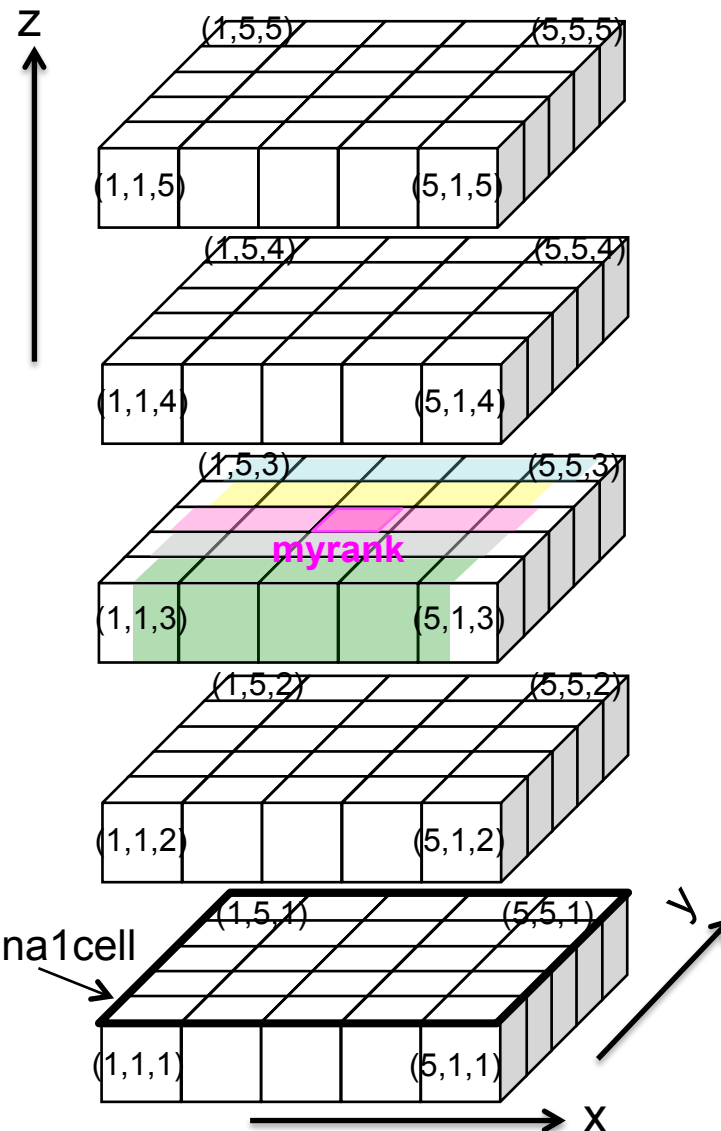
受信データのtagを作成

```
c 座標情報の送受信
```

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

```
enddo
```

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②: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

```
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(1,1,3),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
```

```
call mpi_sendrecv(meta_x(4,narea),...,ipz_mdest,...,
meta_x(,narea),...,ipz_msrc,...)
```

```
enddo
```

-z 軸方向通信

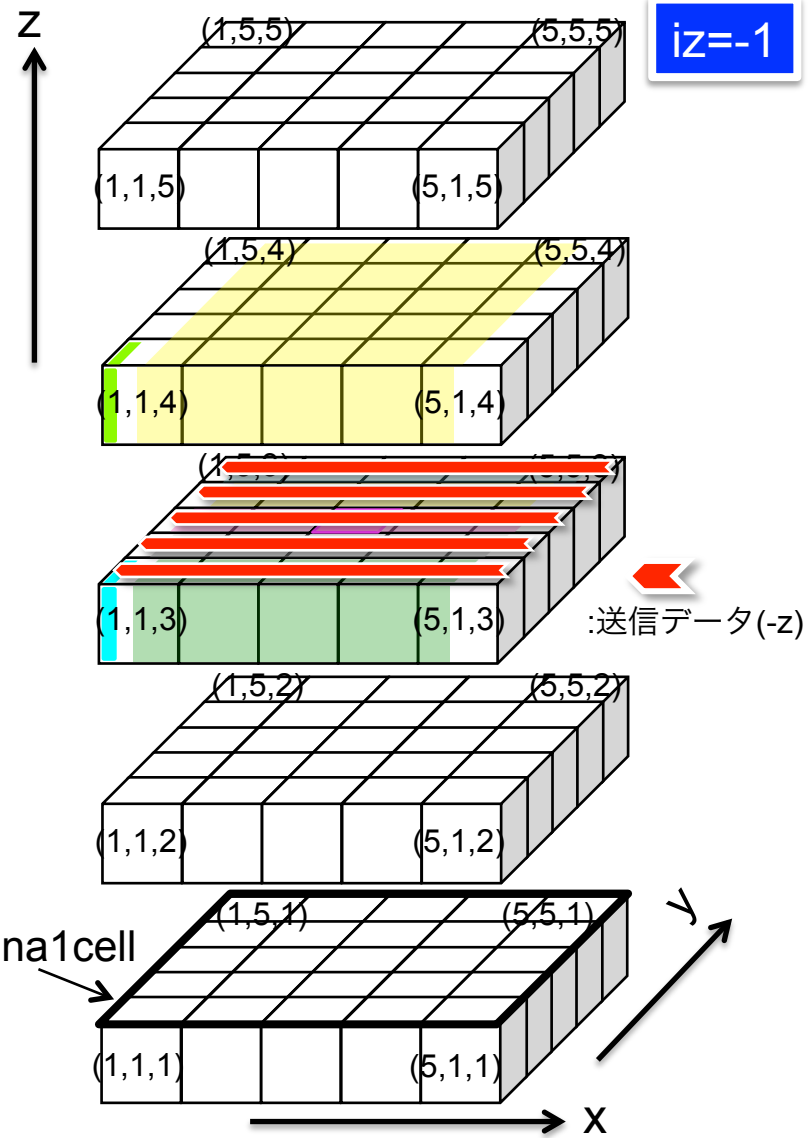
面状データを構成するサブセル数を指定

(1,1,3)セルの先頭アドレスを指定

(1,1,4)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

[サイズ一定の面状データを送受信]



MPI 並列化技術②: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

```

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(4,narea),...,ipz_mdest,...,
meta_x(,narea),...,ipz_msrc,...)

```

```

enddo

```

-z 軸方向通信

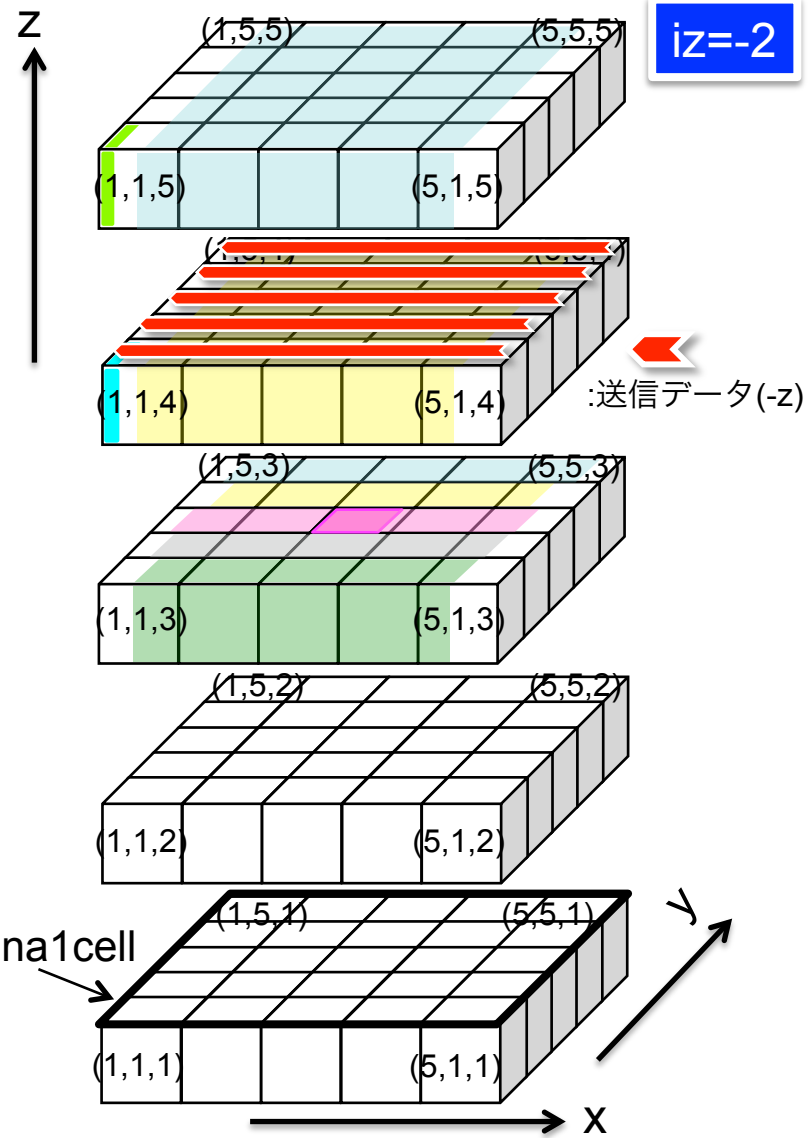
面状データを構成するサブセル数を指定

(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

[サイズ一定の面状データを送受信]



MPI 並列化技術②: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

```
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(1,1,3),25,...,ipz_mdest, ...,
na_per_cell(1,1,2),25,..., ipz_msrc, ...)
```

```
受信データのtag(1:5,1:5,2)を作成
c 座標情報の送受信
```

```
call mpi_sendrecv(meta_x(4,narea,...,ipz_mdest,...,
meta_x(,narea,...,ipz_msrc,...)
```

```
enddo
```

-z 軸方向通信

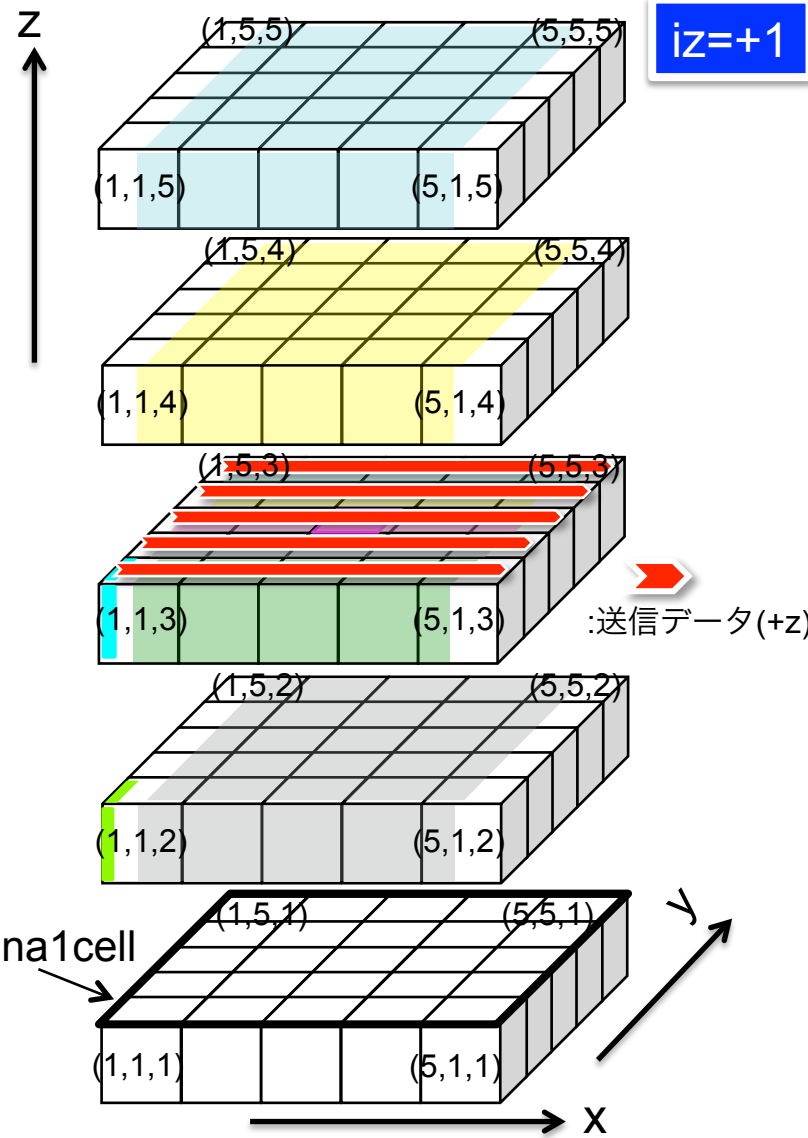
面状データを構成するサブセル数を指定

(1,1,3)セルの先頭アドレスを指定

(1,1,2)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

[サイズ一定の面状データを送受信]



MPI 並列化技術②: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

```
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(1,1,2),25,...,ipz_mdest, ...,
na_per_cell(1,1,1),25,..., ipz_msrc, ...)
```

受信データのtag(1:5,1:5,2)を作成

```
c 座標情報の送受信
```

```
call mpi_sendrecv(meta_x(4,narea,...,ipz_mdest,...,
meta_x(,narea,...,ipz_msrc,...)
```

```
enddo
```

-z 軸方向通信

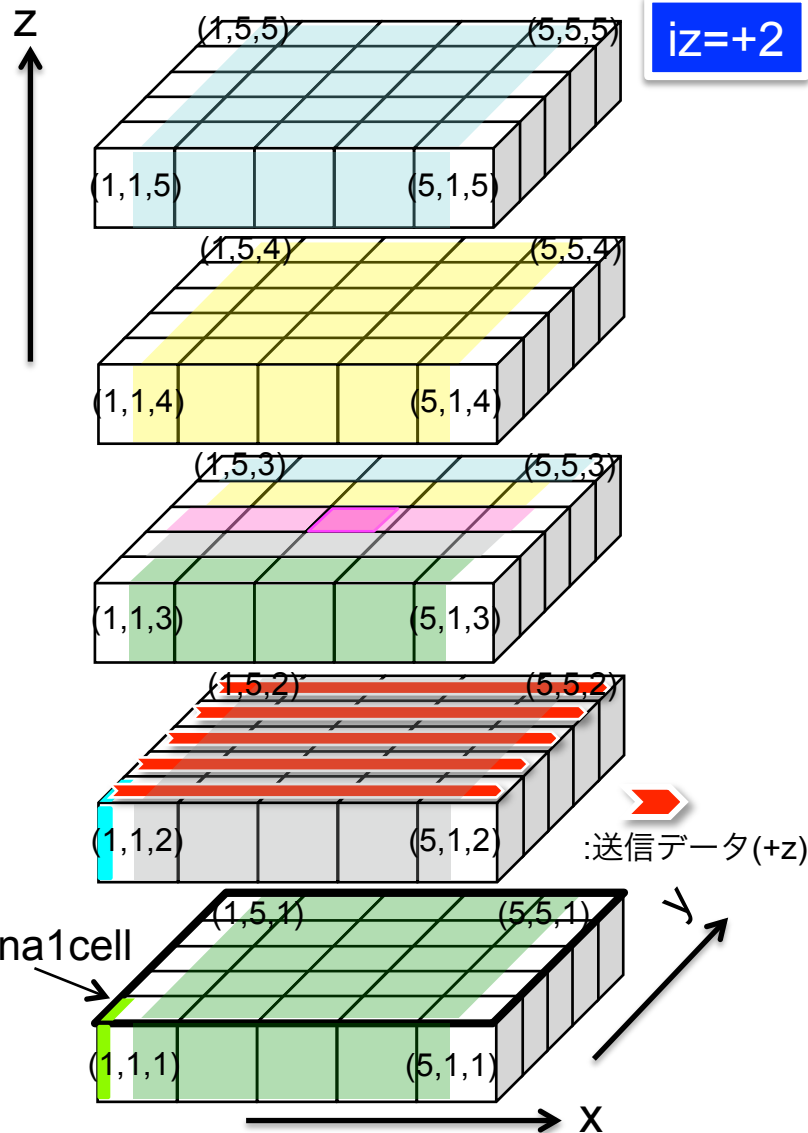
面状データを構成する
サブセル数を指定

(1,1,2)セルの先頭
アドレスを指定

(1,1,1)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

[サイズ一定の面状データを送受信]

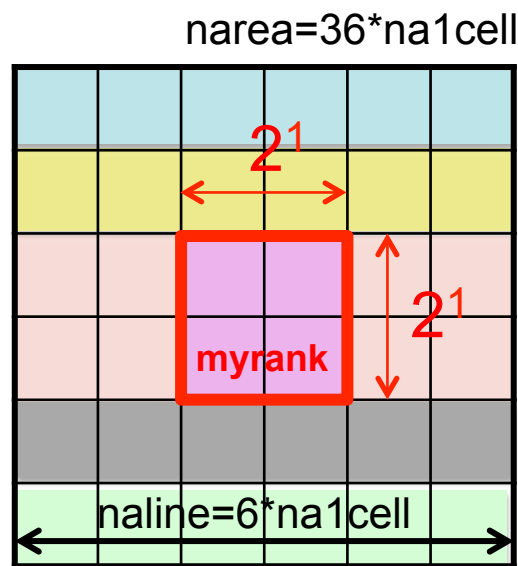


→ すべての袖部のデータが受信された

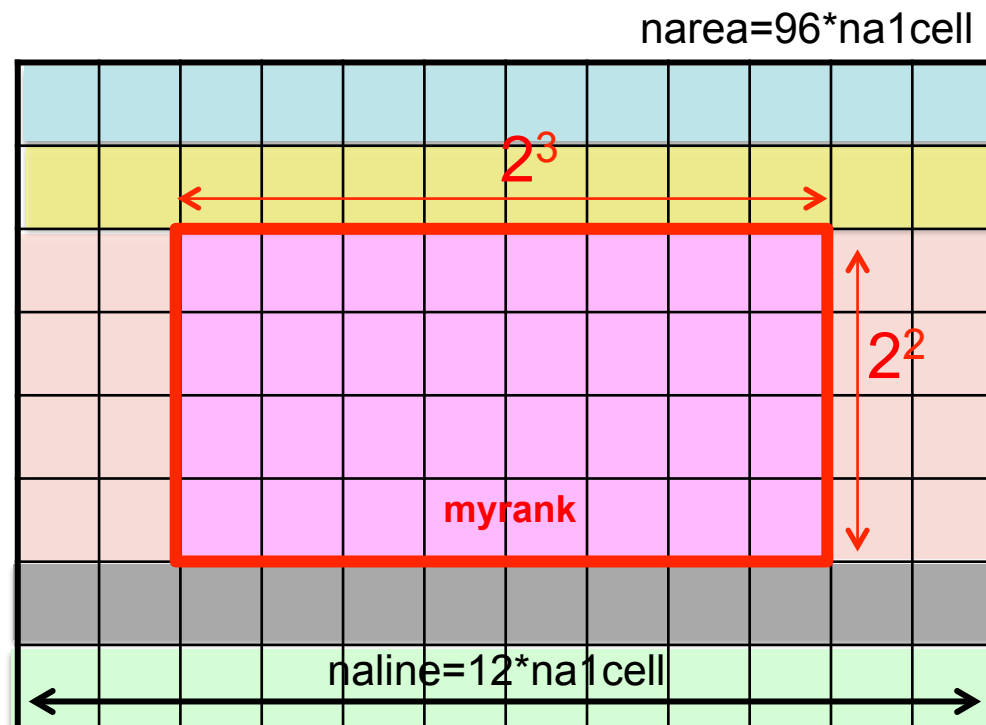


汎用性のため実際のコードでは以下の点にも対応.

- 任意のmyrank所持サブセルブロック厚さ 2^n ($n \geq 0$)
- 直方体形状のmyrank所持サブセルブロック



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



```
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 $z \rightarrow y \rightarrow x$ の順に通信しています.

comm_3.f, comm_fmm.f

MPI 並列化技術③: 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)



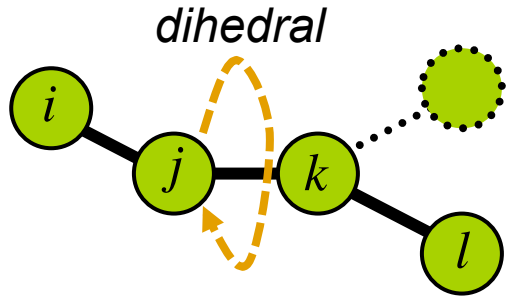
冗長な多重演算の例

- 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③: 通信の演算による代用

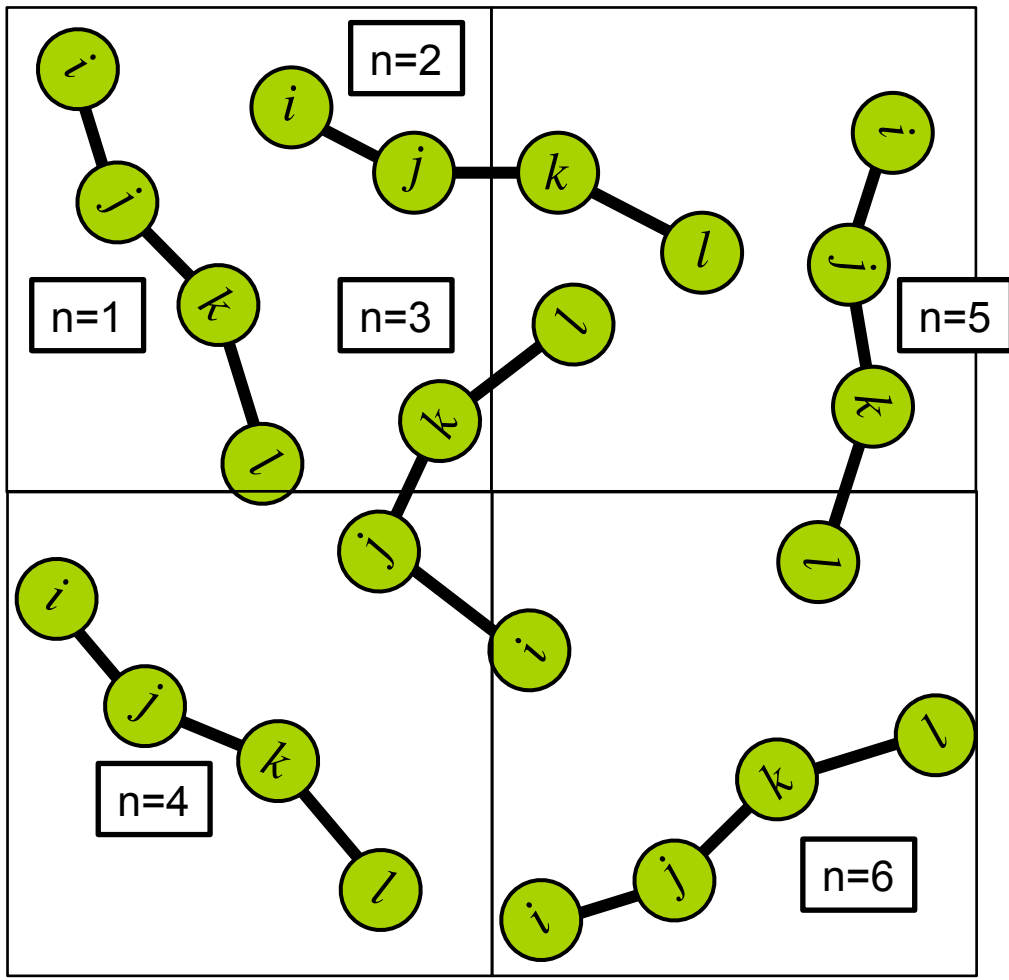
冗長な多重演算: 分子内相互作用計算



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

```
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
```

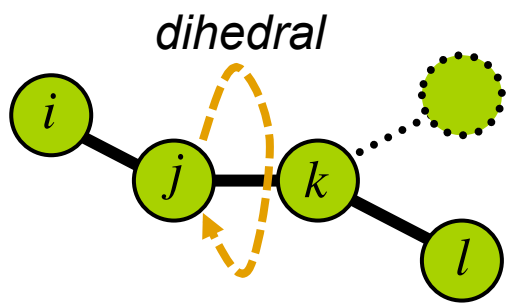
オリジナルコード





MPI 並列化技術③: 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

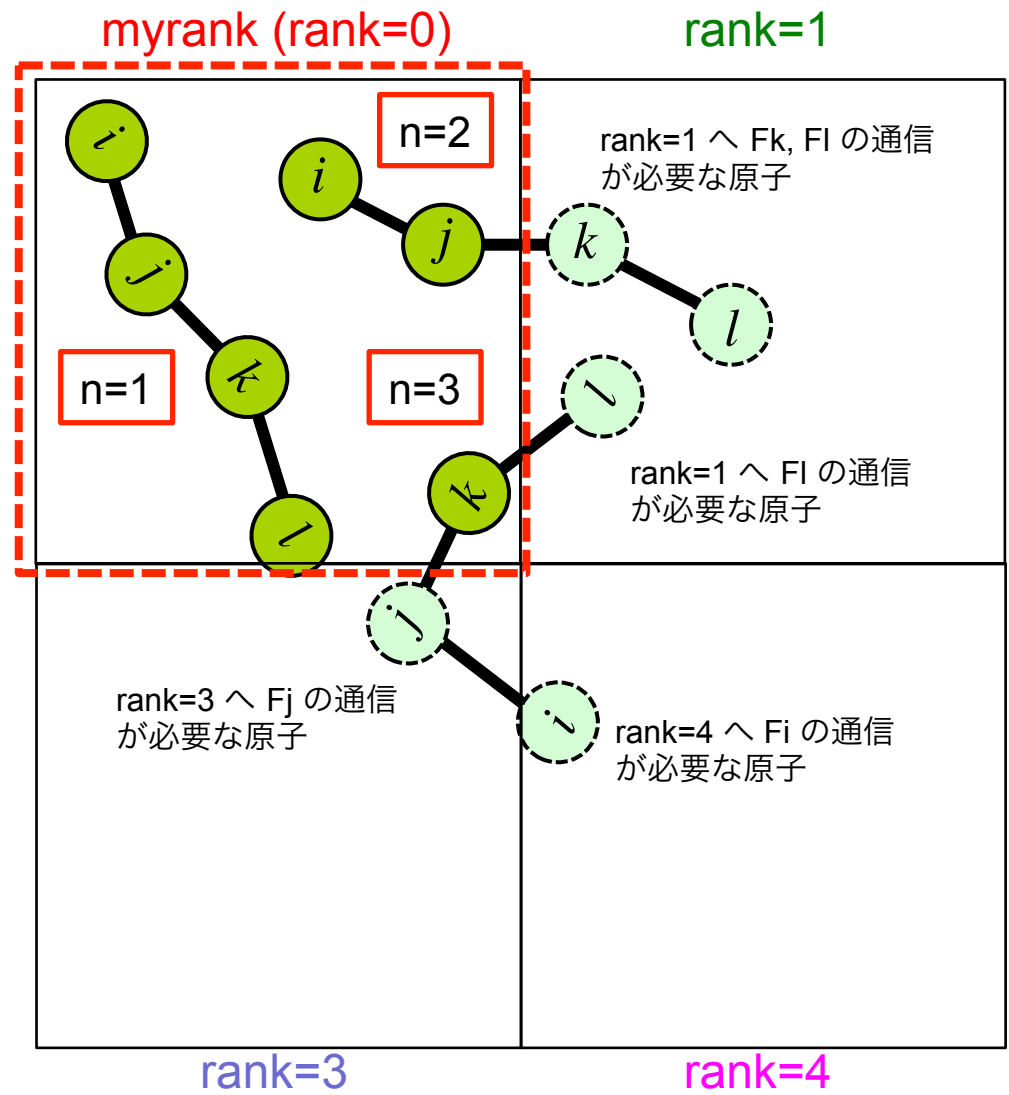


$$K_\phi [1 + \cos(n\phi(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k, \mathbf{r}_l) - \delta)]$$

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

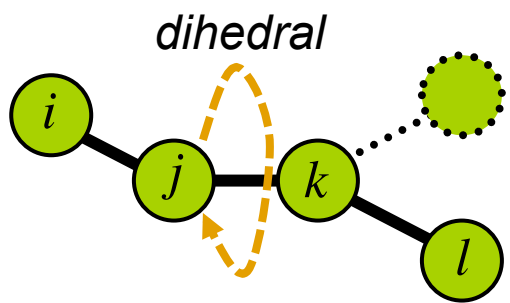




MPI 並列化技術③: 通信の演算による代用

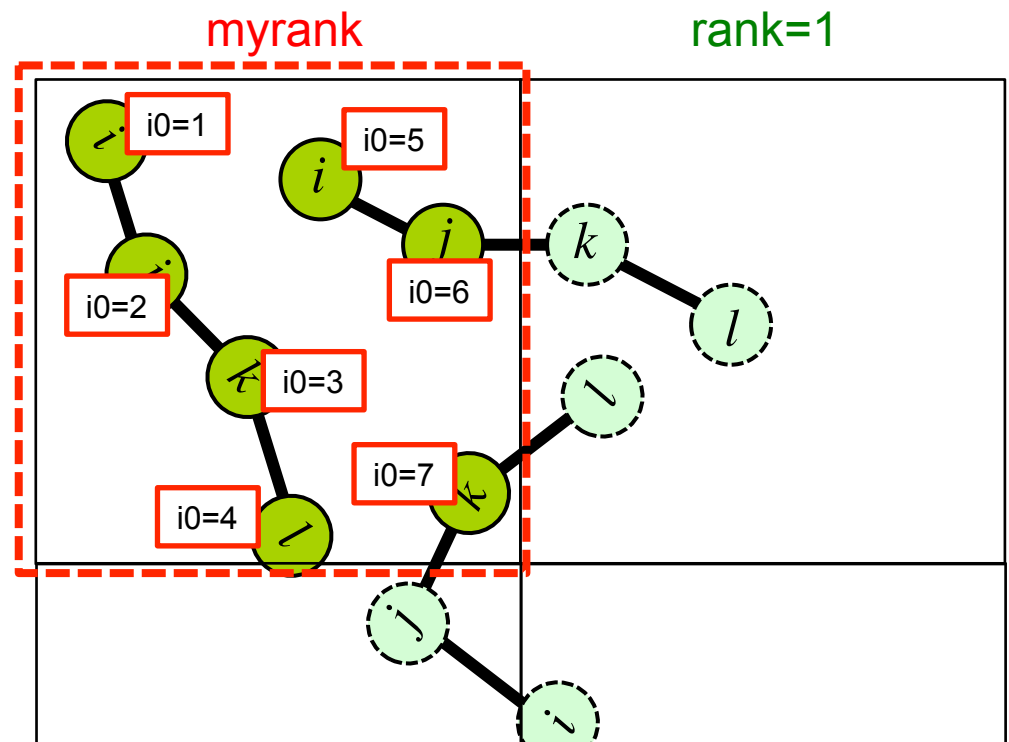
冗長な多重演算: 分子内相互作用計算

i0 : myrank内の原子通し番号



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

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(i)=f(i)+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

MPI 並列化技術③: 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 $F_{ij} = -F_{ji}$

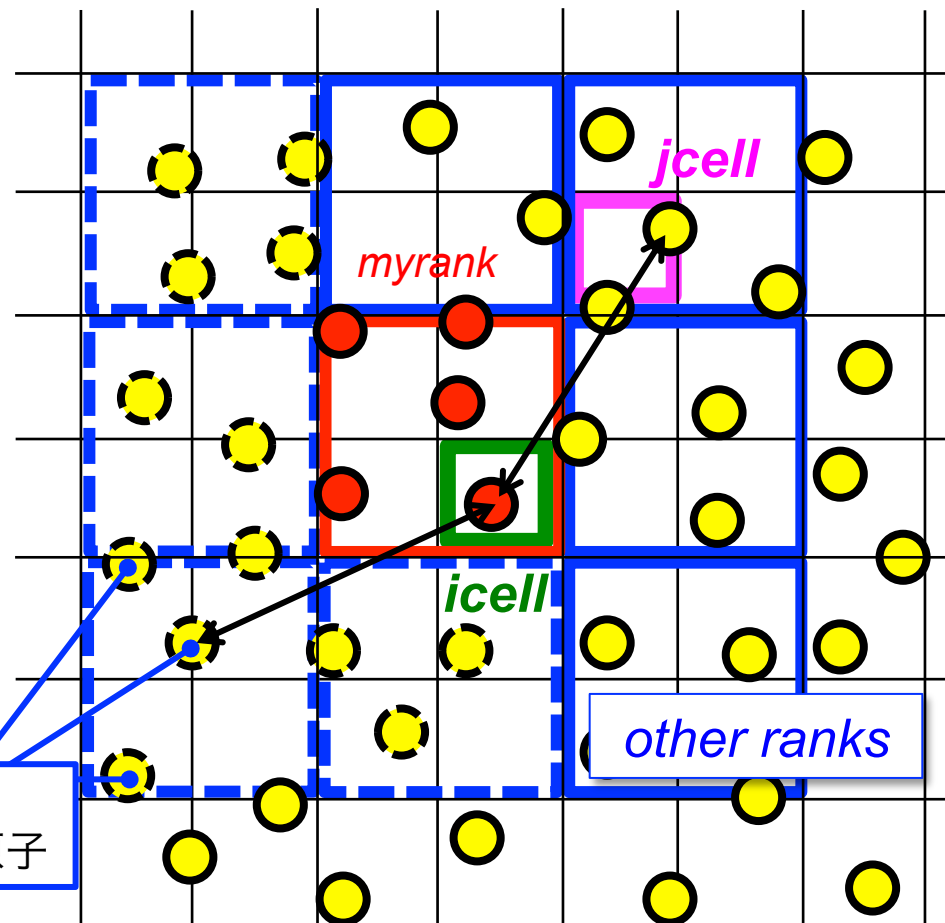
例) Lennard-Jones, Coulombの粒子対計算

```

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

Fij の通信
が必要な原子



MPI 並列化技術③: 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

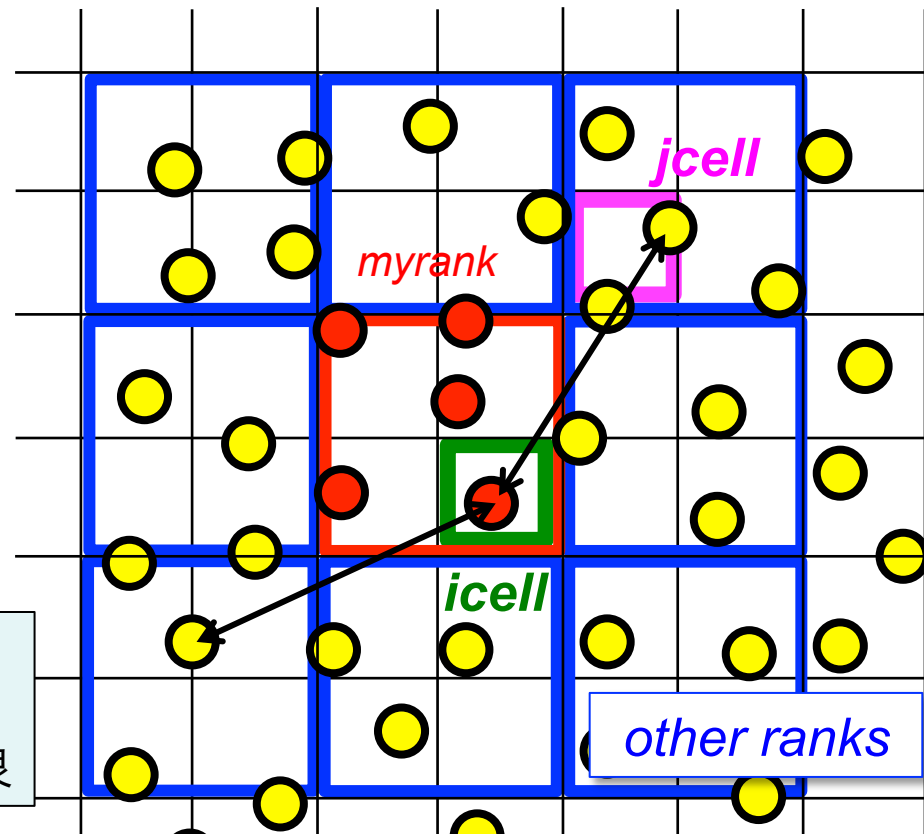
二体力 $F_{ij} = -F_{ji}$

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

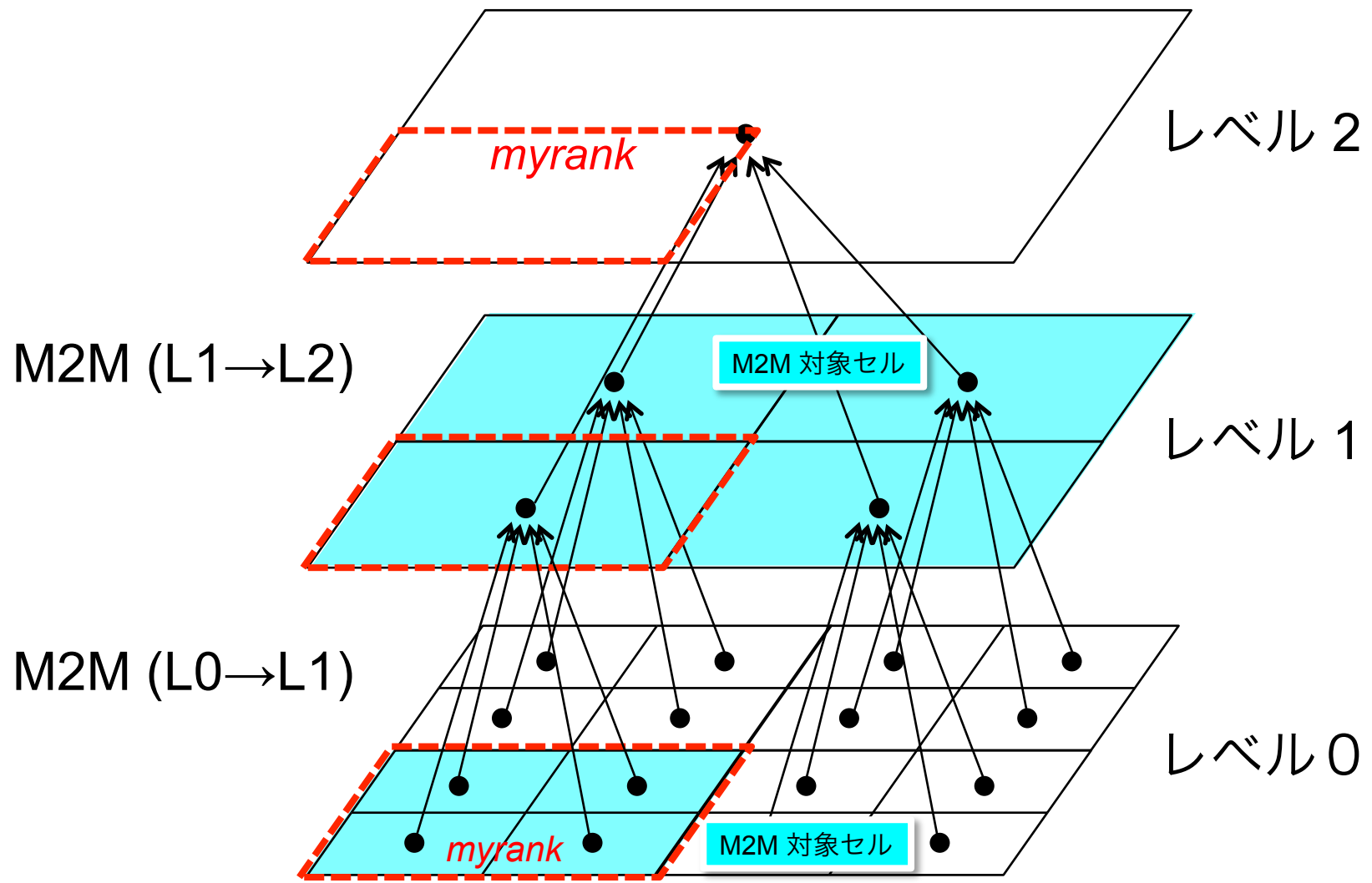


実演算時間 < 実通信時間 の範囲で有効 (例えばノード数が多く演算時間が短いとき)



MPI 並列化技術③: 通信の演算による代用

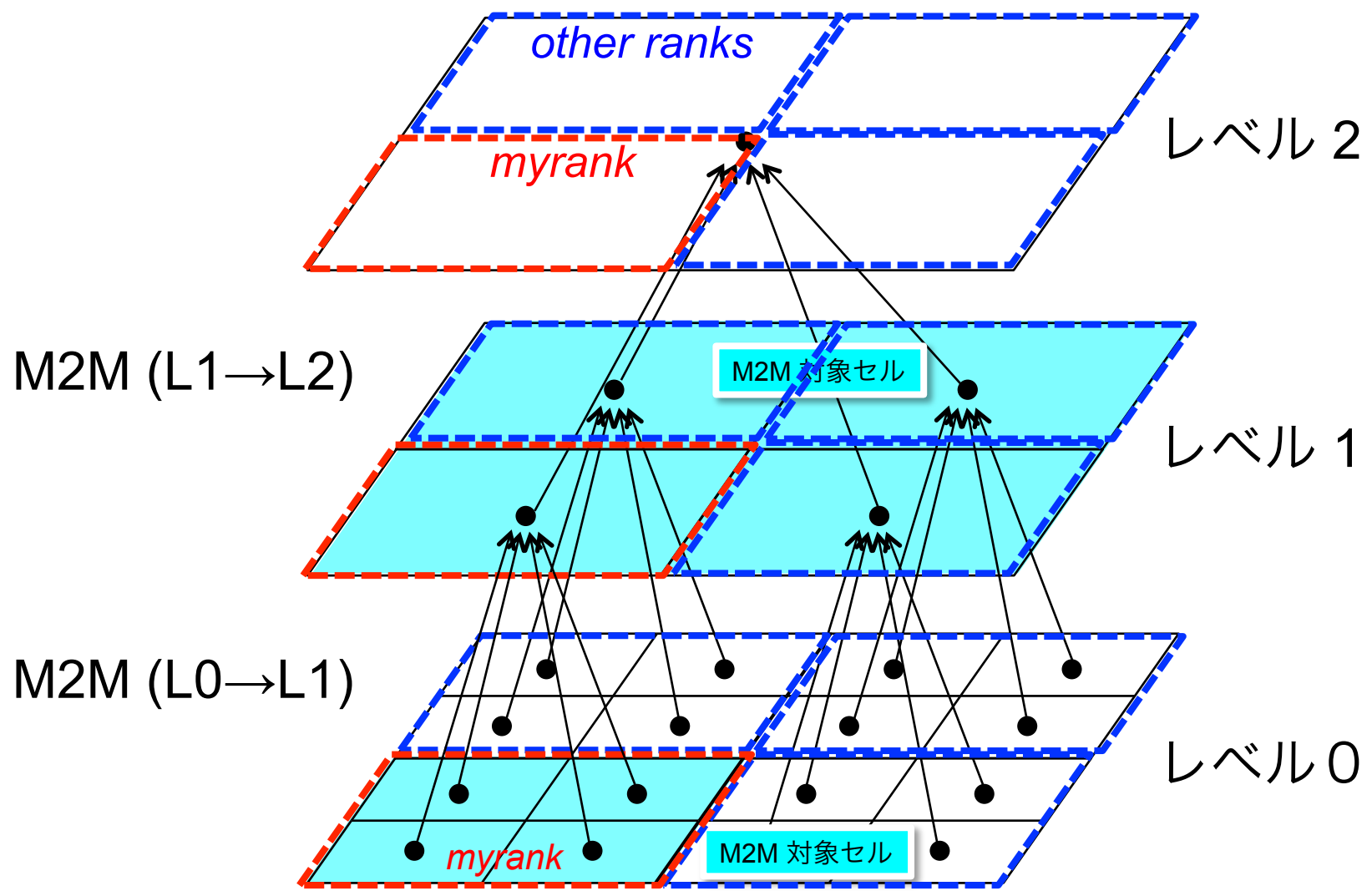
冗長な多重演算: FMM 上位階層の M2M





MPI 並列化技術③: 通信の演算による代用

冗長な多重演算: FMM 上位階層の M2M



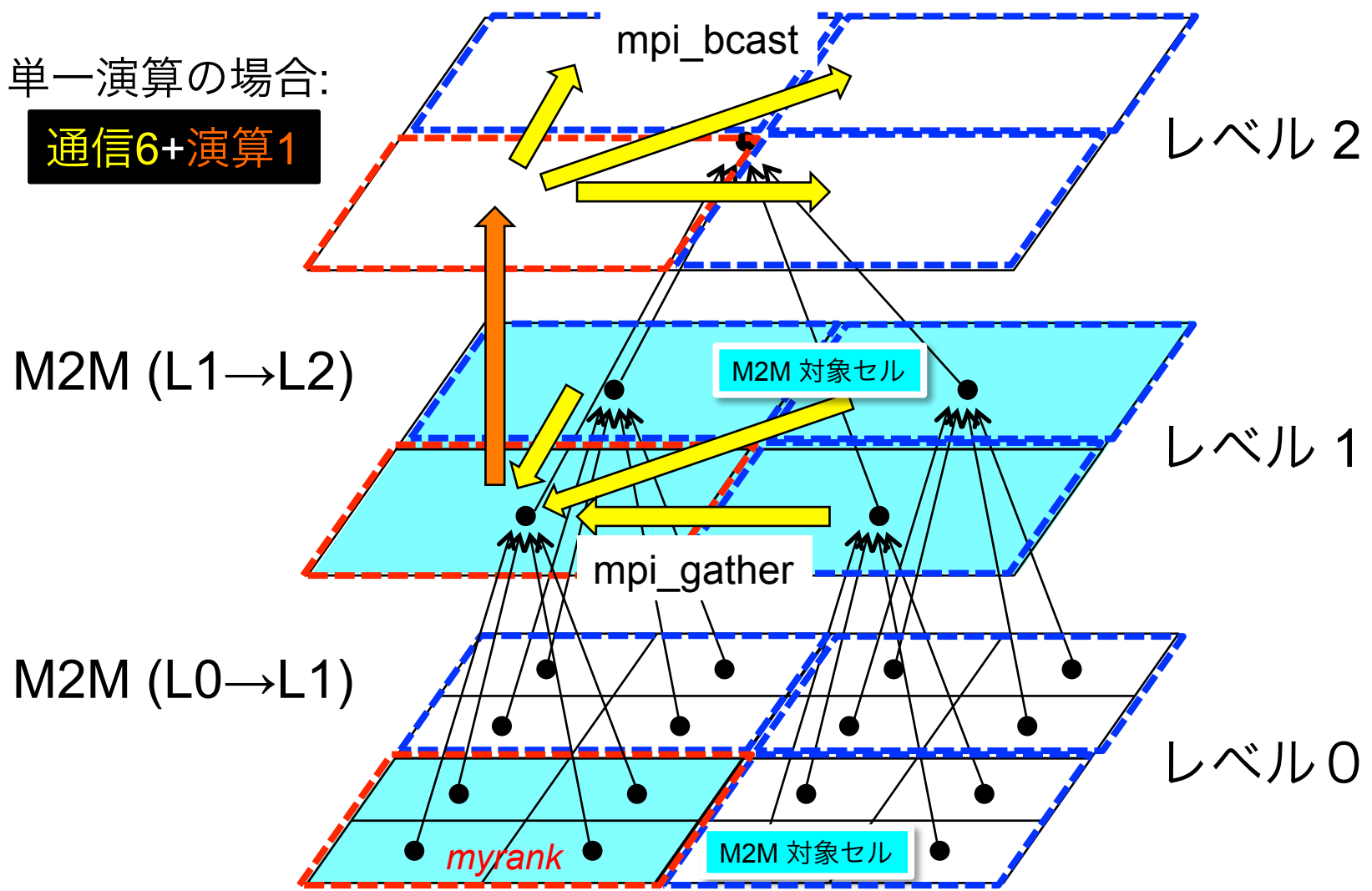


MPI 並列化技術③: 通信の演算による代用

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



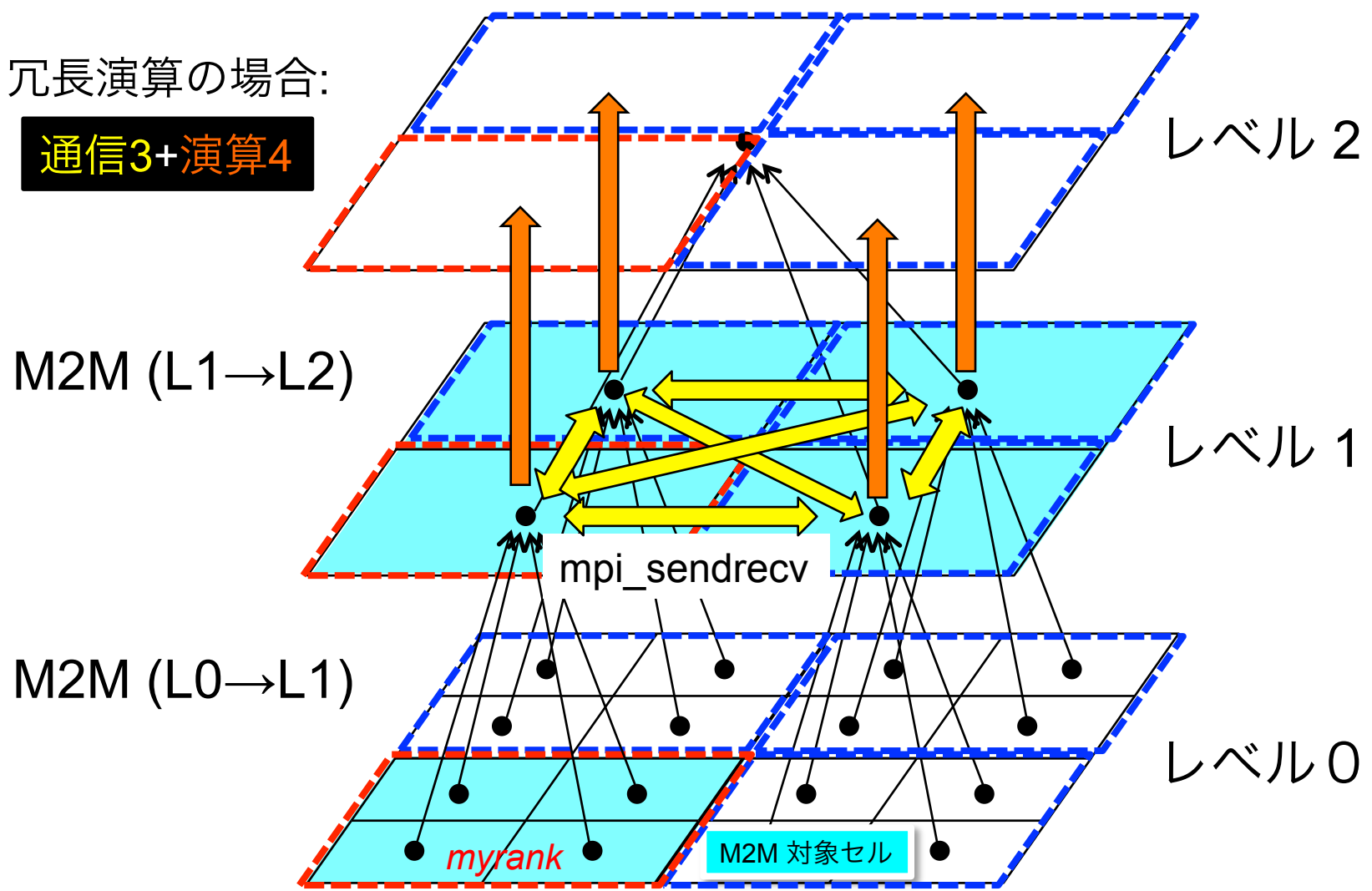


MPI 並列化技術③: 通信の演算による代用

冗長な多重演算: FMM 上位階層の M2M

冗長演算の場合:

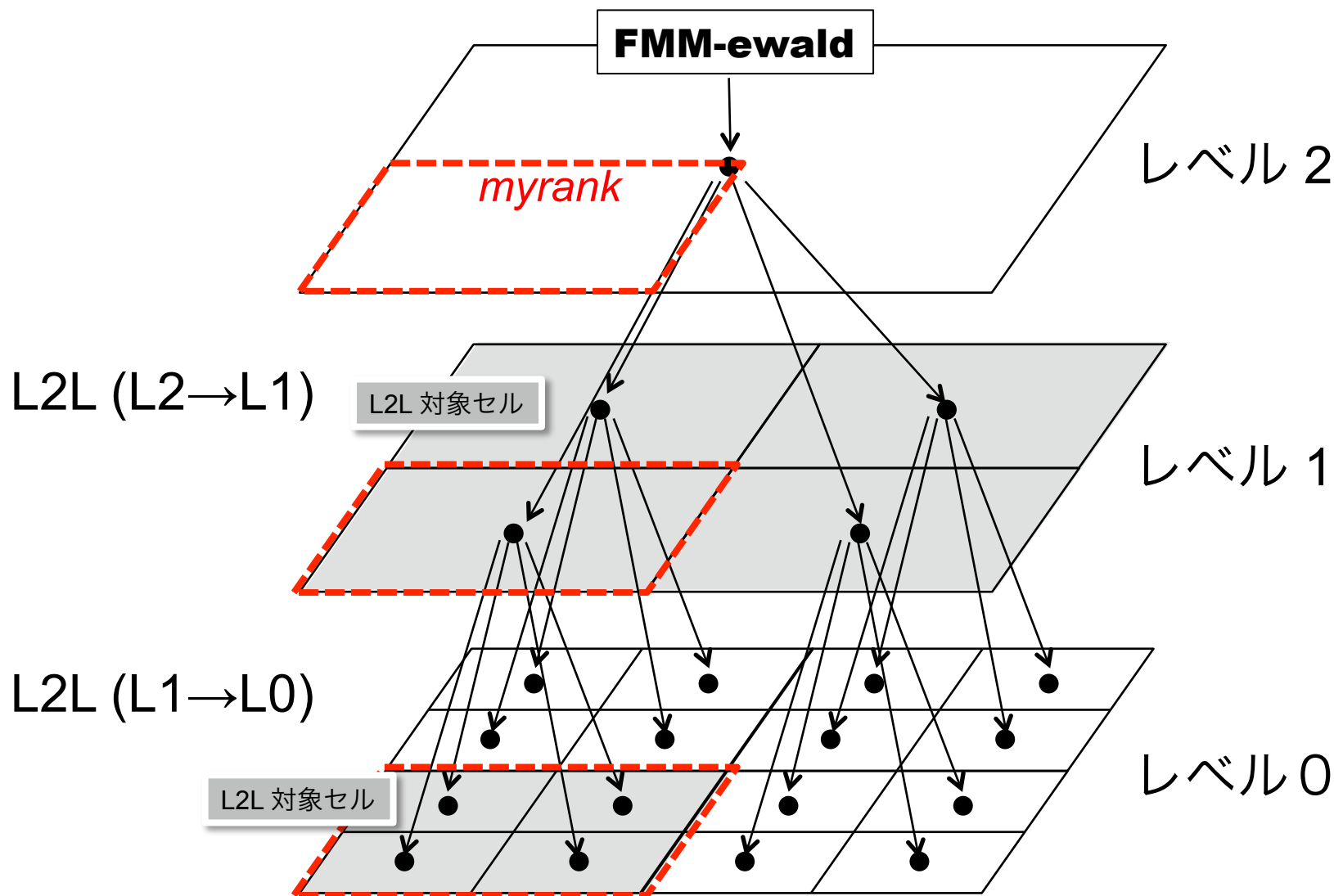
通信3+演算4





MPI 並列化技術③: 通信の演算による代用

冗長な多重演算: FMM 上位階層の L2L

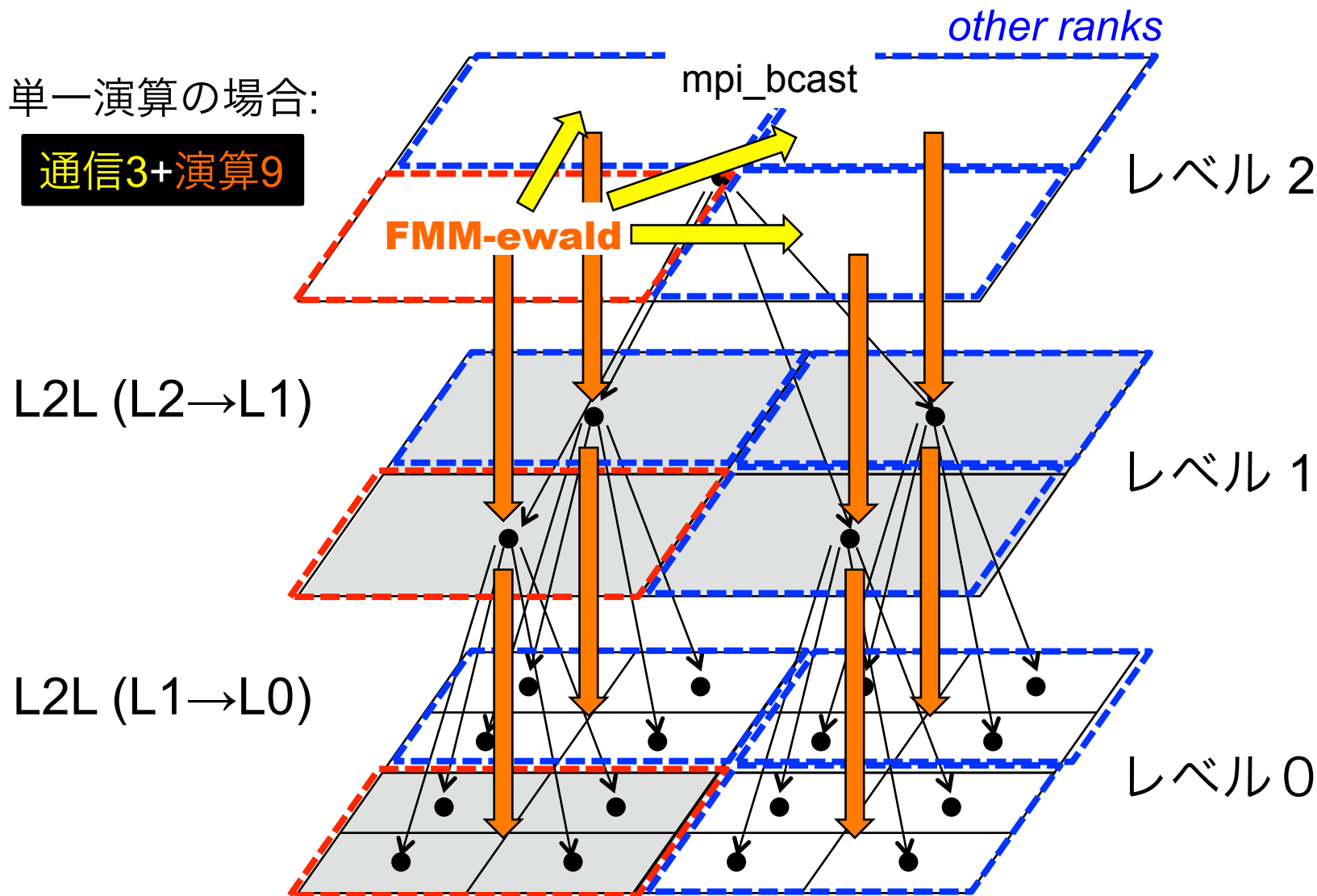


MPI 並列化技術③: 通信の演算による代用

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





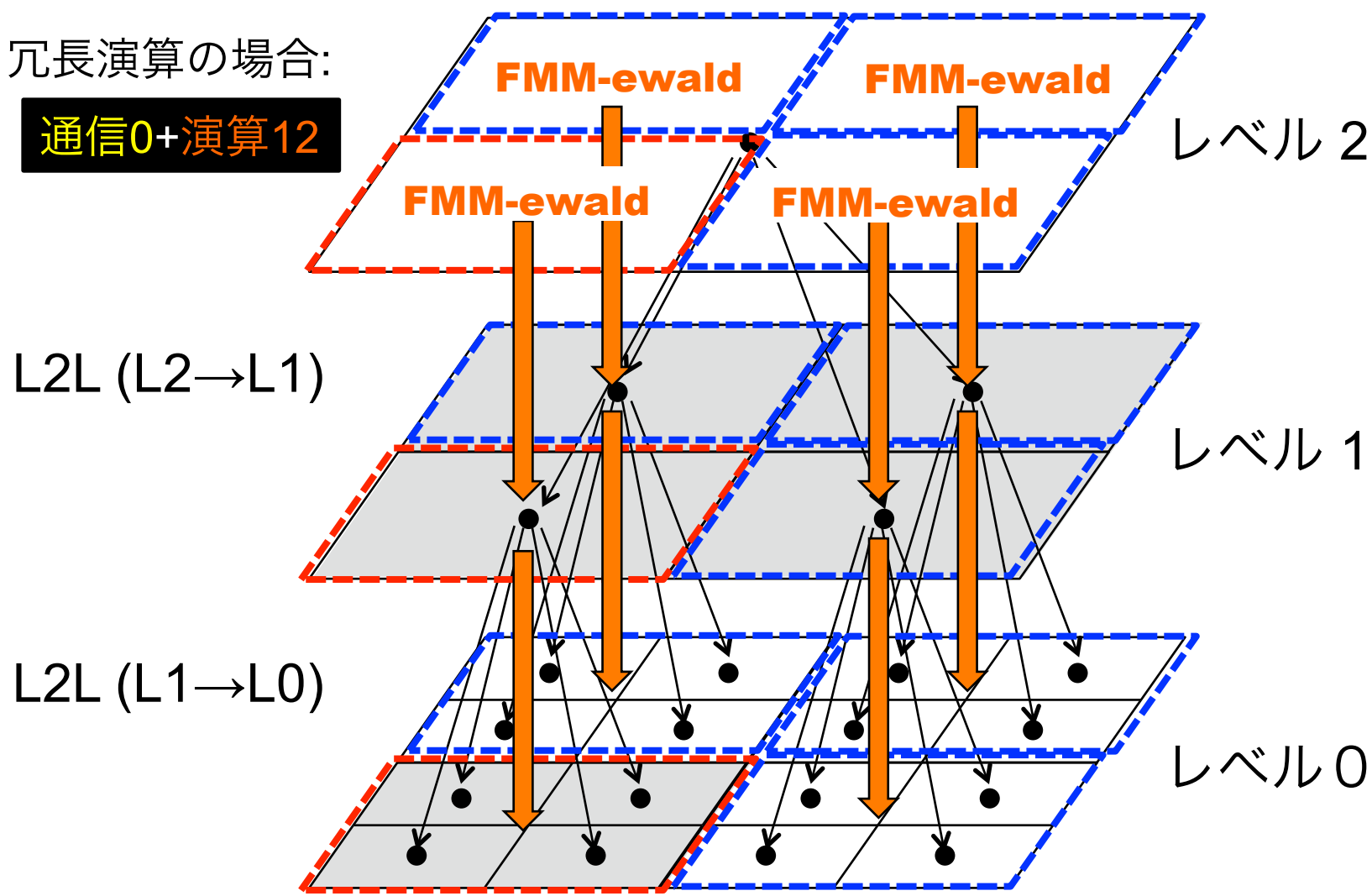
MPI 並列化技術③: 通信の演算による代用

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



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

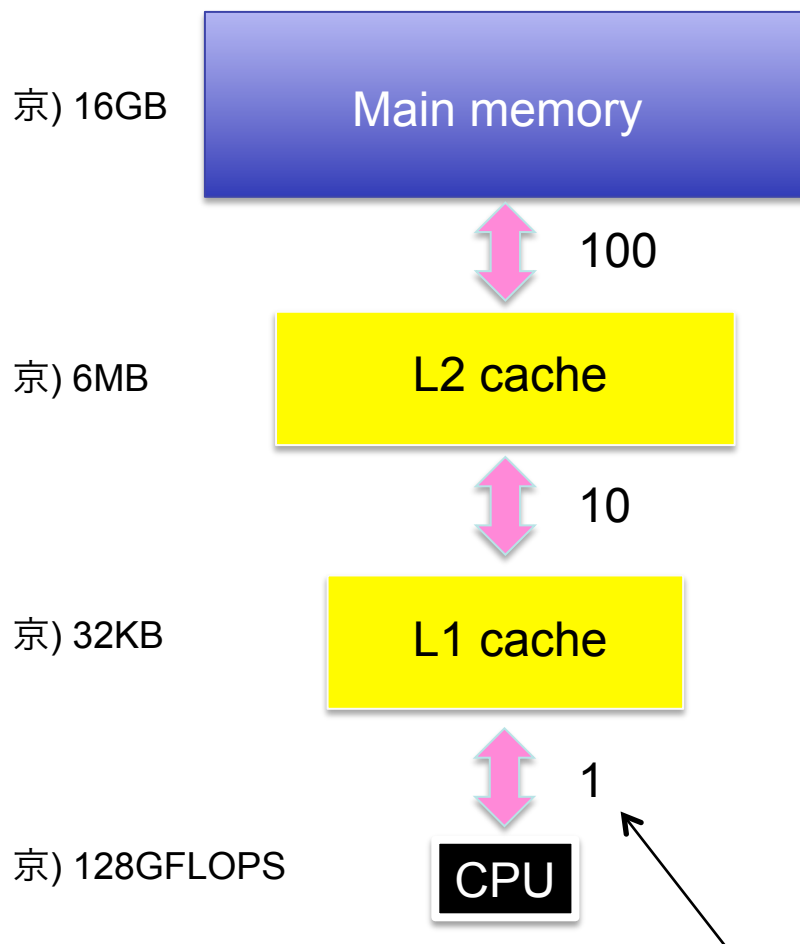
OpenMP 並列化技術

- ・ スレッド間のロードバランス調整
- ・ 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 w_m , $M2L$ 変換行列の再利用促進

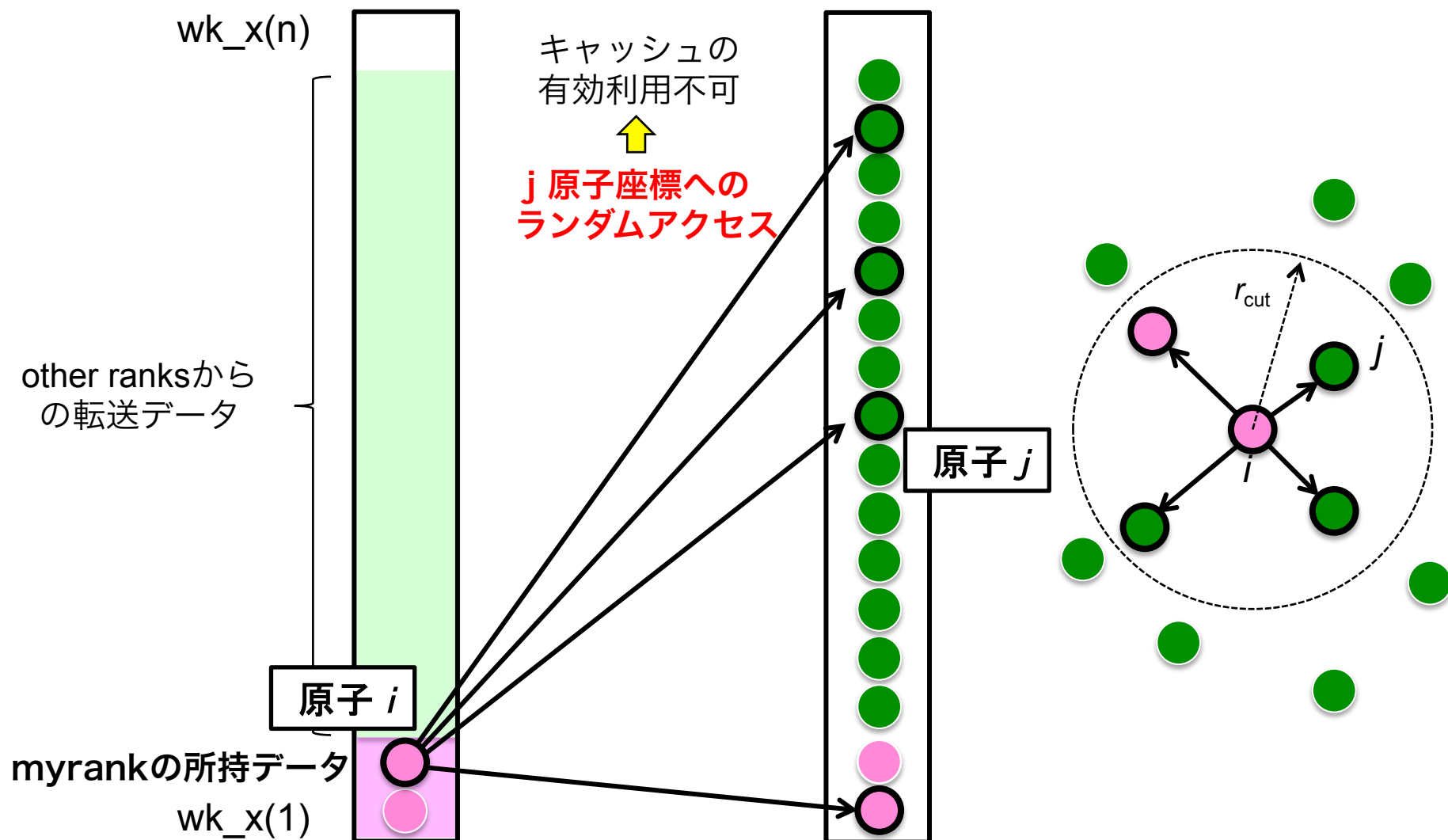
前提

- (1) メモリ上の演算対象データの連続化
- (2) ブロック化

相対アクセス速度

データの連続化 [1] 座標

従来の配列

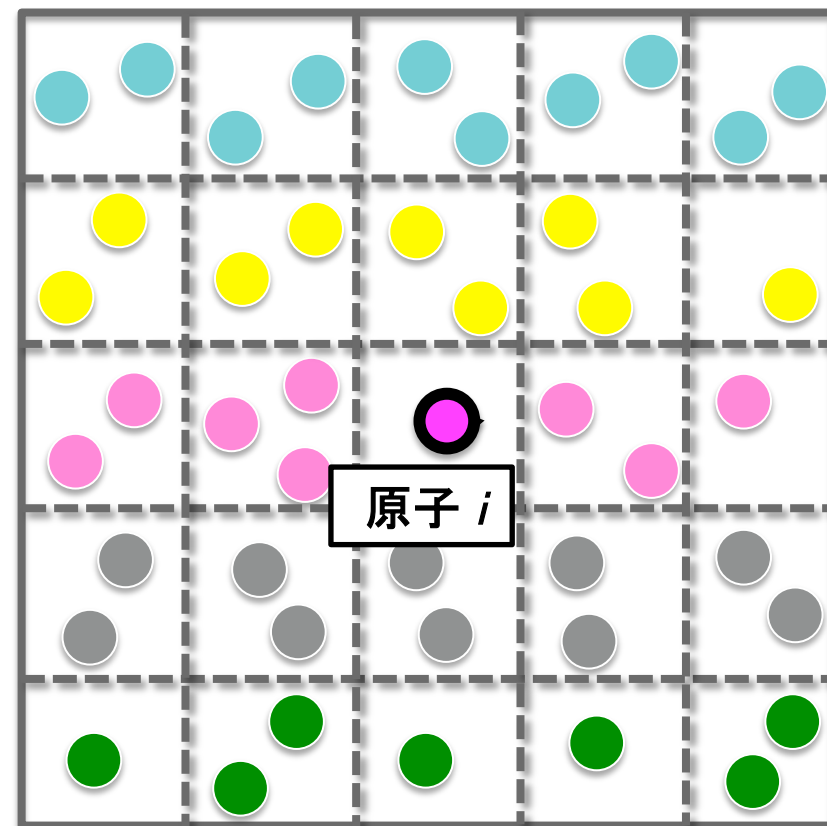
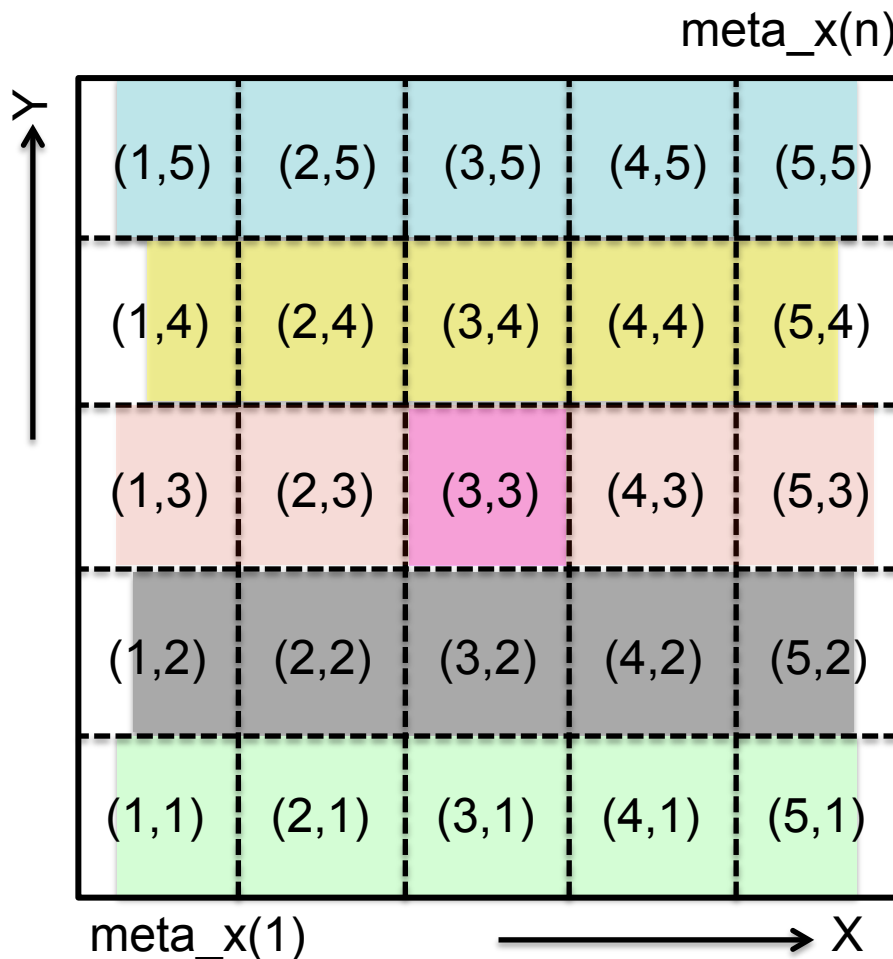


データの連続化 [1] 座標

メタデータ配列

前回説明

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



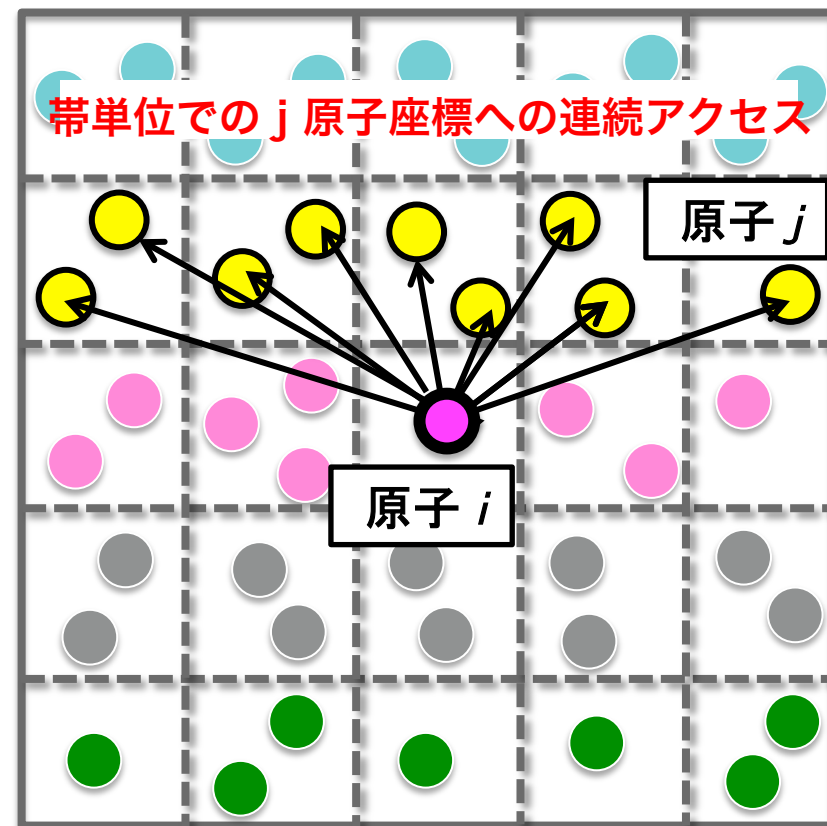
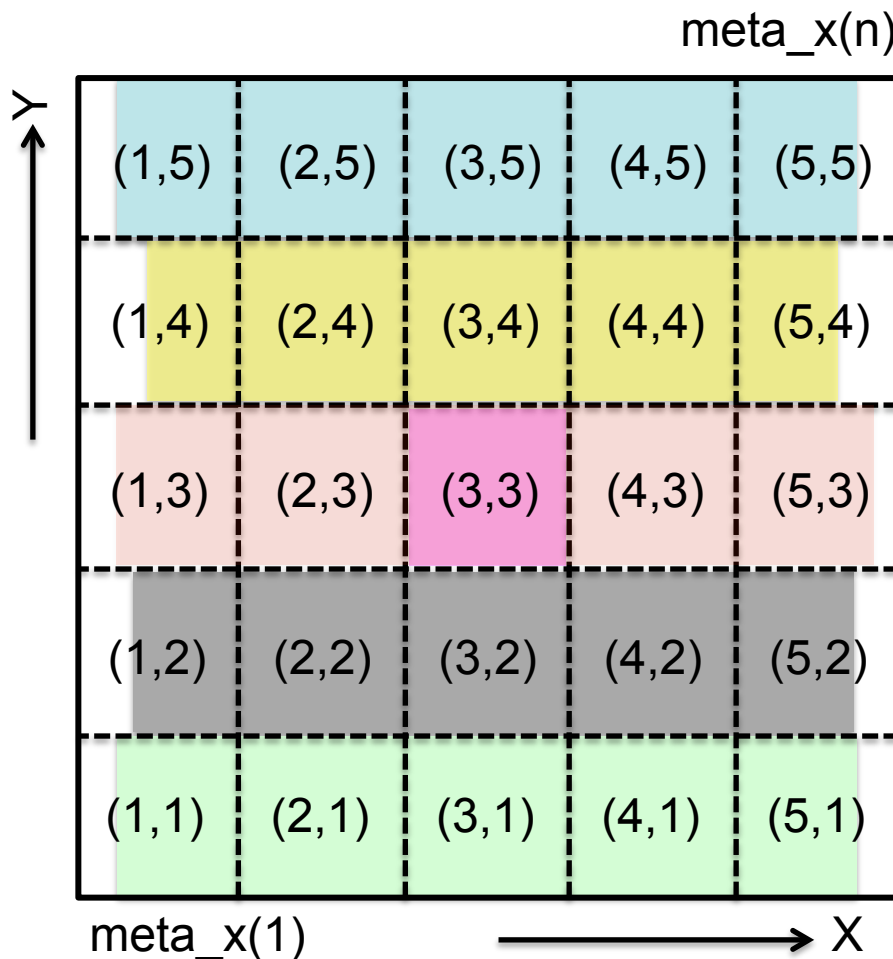
- 1) 自プロセスを中心にデータを局所化
- 2) 袖部に直にデータ装填

データの連続化 [1] 座標

メタデータ配列

前回説明

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

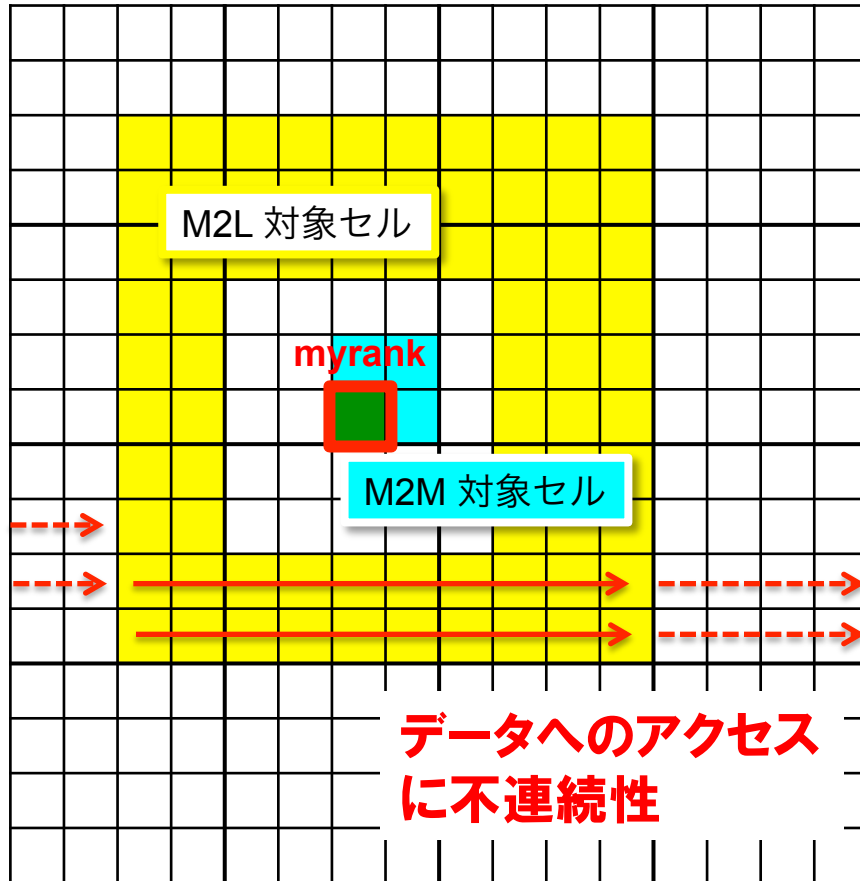


- 1) 自プロセスを中心にデータを局所化
- 2) 袖部に直にデータ装填

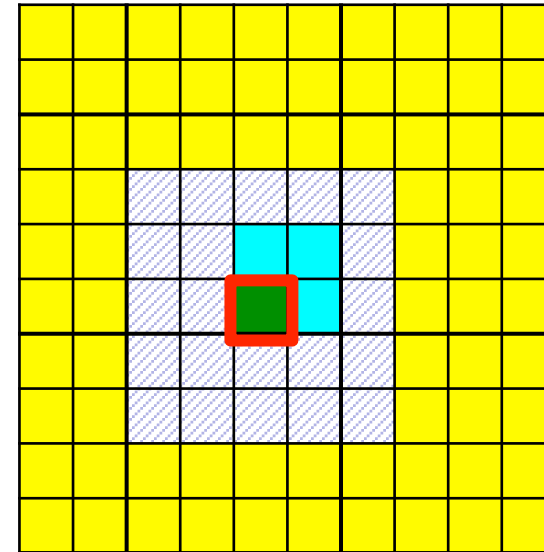
データの連続化 [2] 多極子

前回説明

従来の配列



袖部付き局所化配列

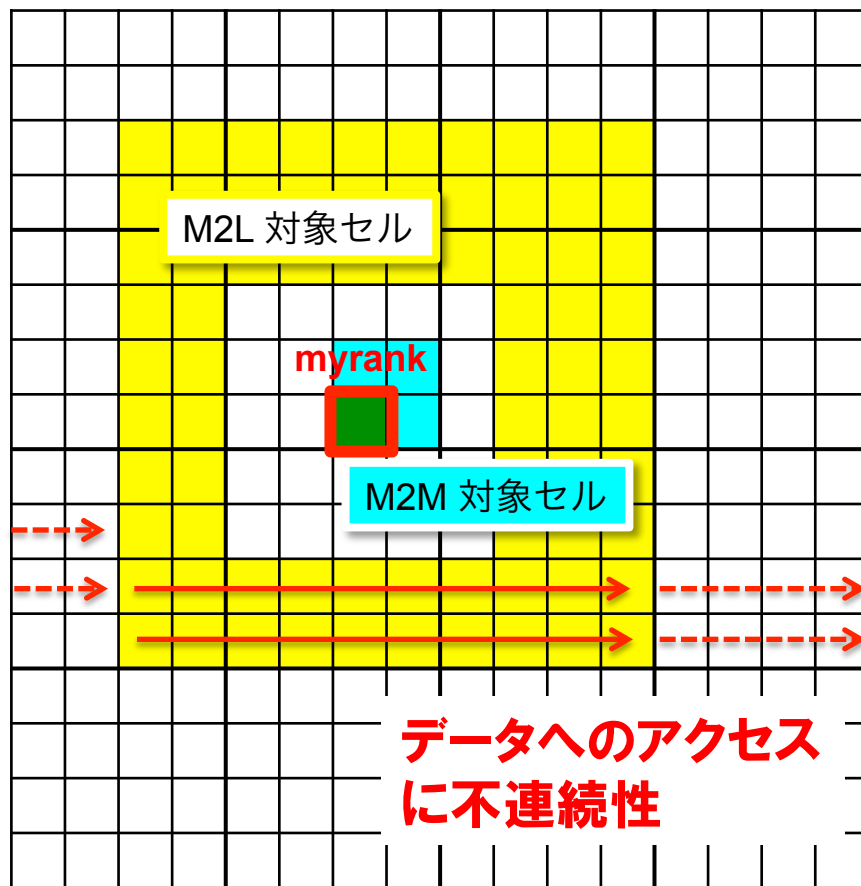


- 1) 自プロセスを中心にデータを局所化
- 2) 袖部に直にデータ装填

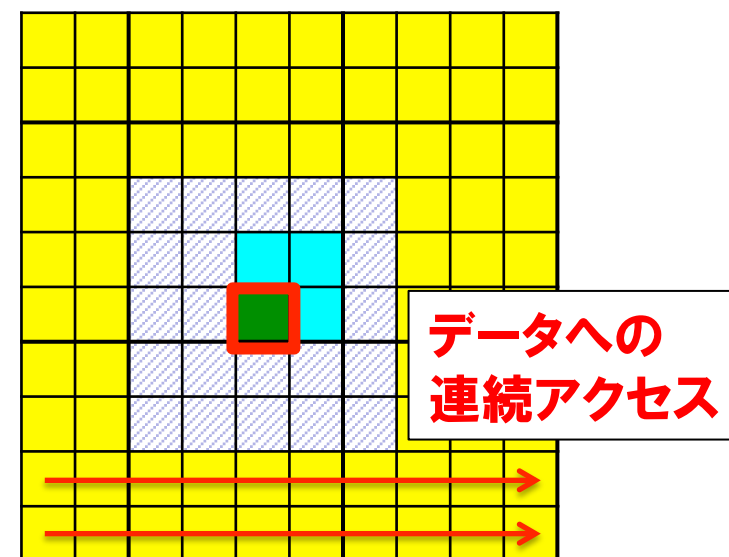
データの連続化 [2] 多極子

前回説明

従来の配列



袖部付き局所化配列

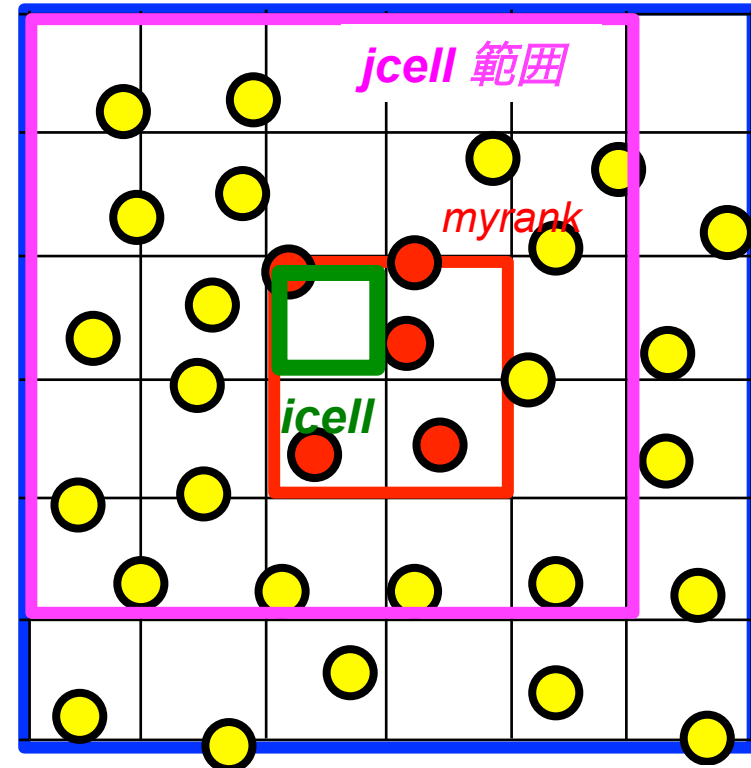


- 1) 自プロセスを中心にデータを局所化
- 2) 袖部に直にデータ装填

ブロック化によるキャッシュの有効利用 (1)

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
```

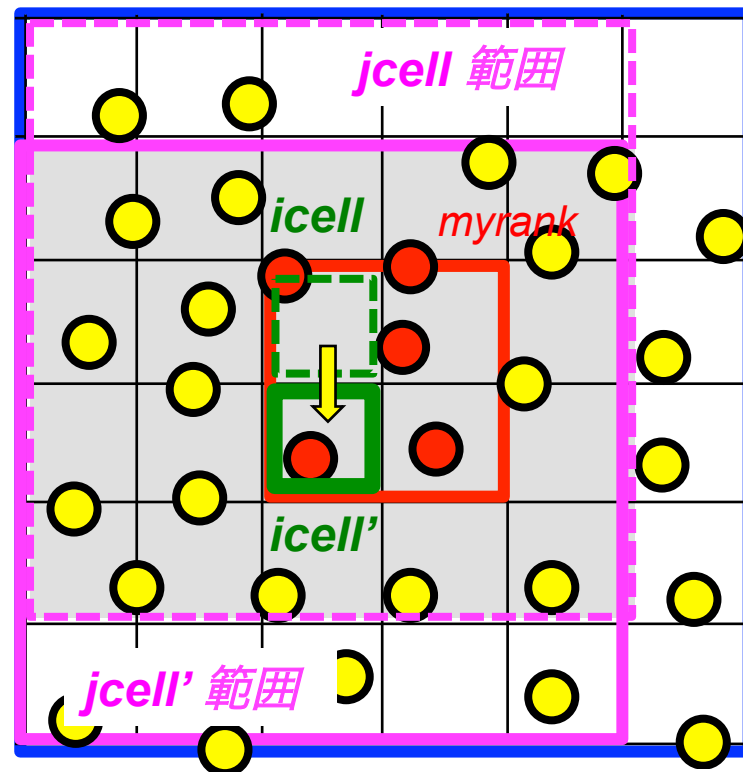
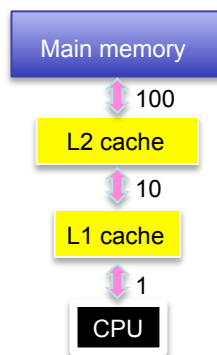


座標データ転送済範囲

ブロック化によるキャッシュの有効利用 (1)

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 (1)

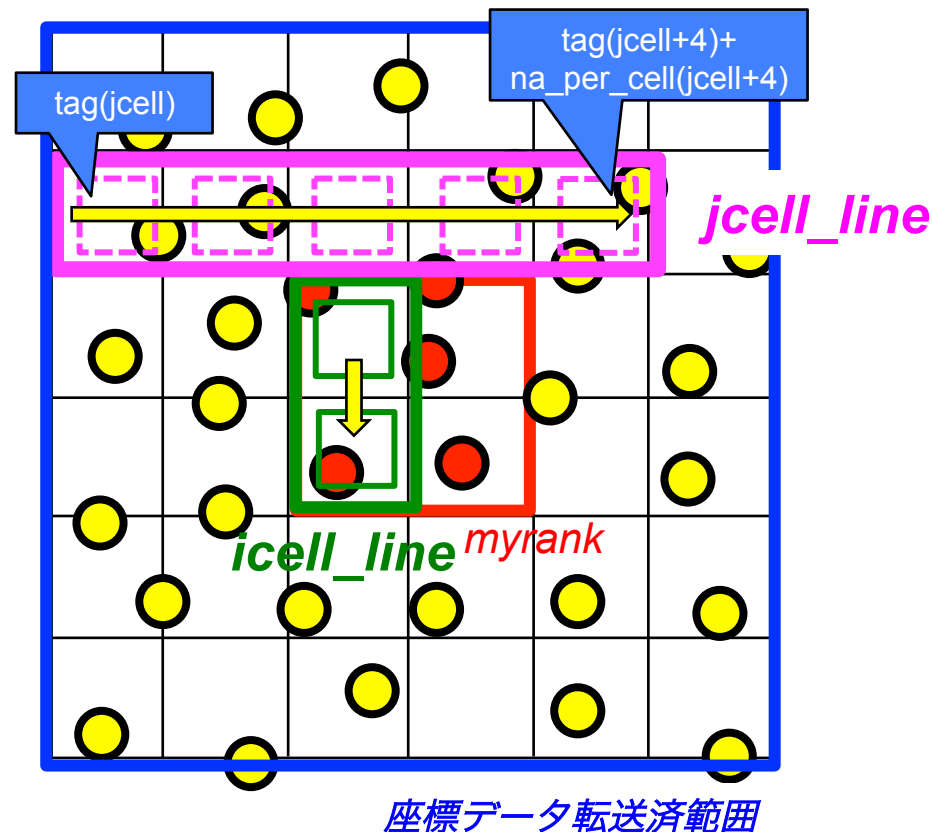
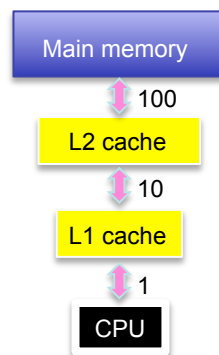
ブロック化後のループ構造:

```

do jcell_line
  do icell(myrank)
    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

```

$jcell$ を外側へ移動
 $[along\ icell_line]$



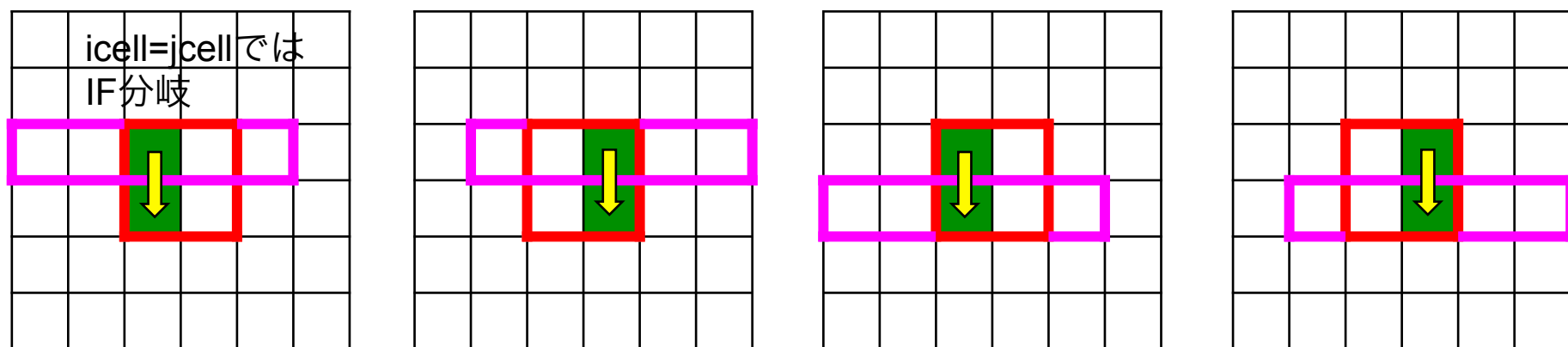
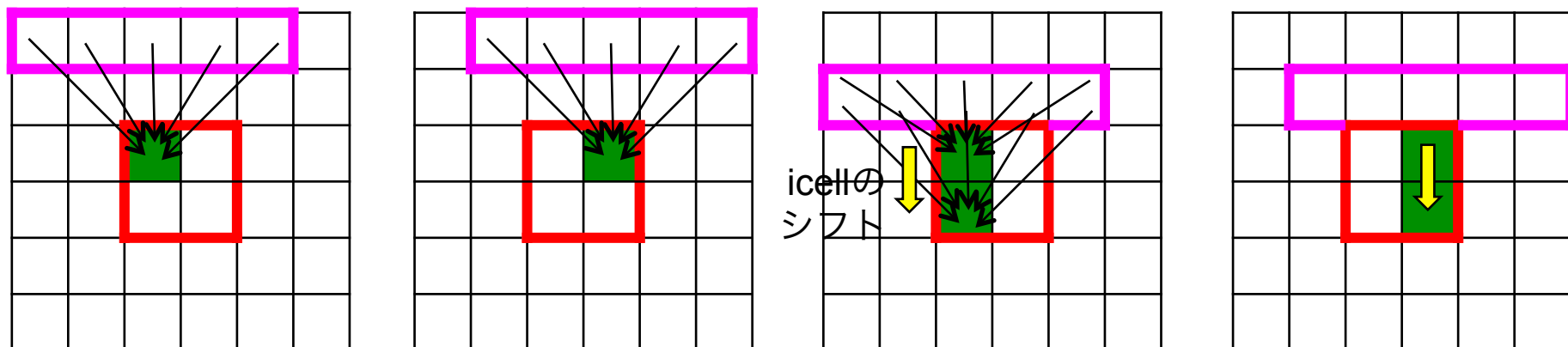
(1) $jcell_line$ の原子座標データは 1 回のみロードされる。

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 の性能向上

ブロック化によるキャッシュの有効利用 (1)

最外DOループによる jcell_line シフトの概念図



以下同様

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 (2)

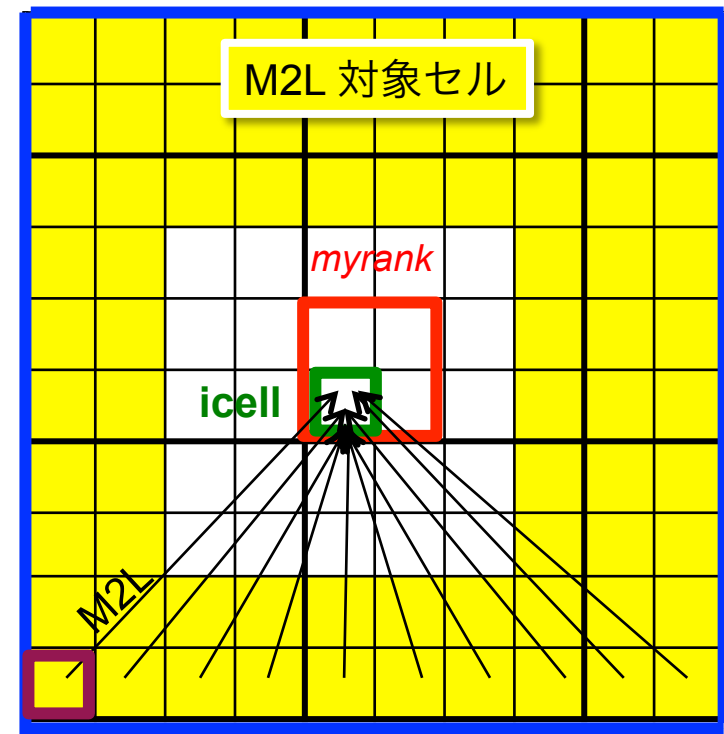
ブロック化前のループ構造 (M2L):

```

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

```

$wm_local((nmax+1)^2, 10, 10)$



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 (2)

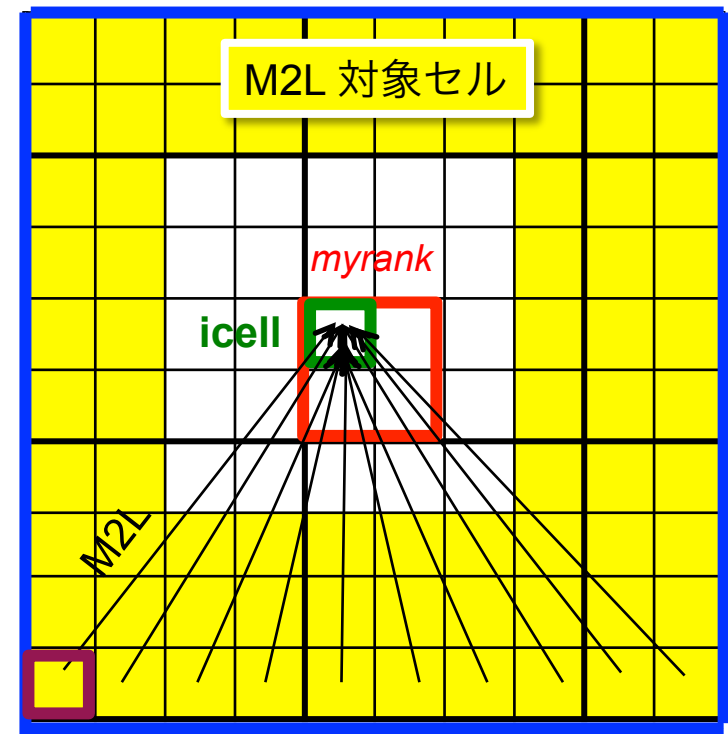
ブロック化前のループ構造 (M2L):

```

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

```

$wm_local((nmax+1)^2, 10, 10)$



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 (2)

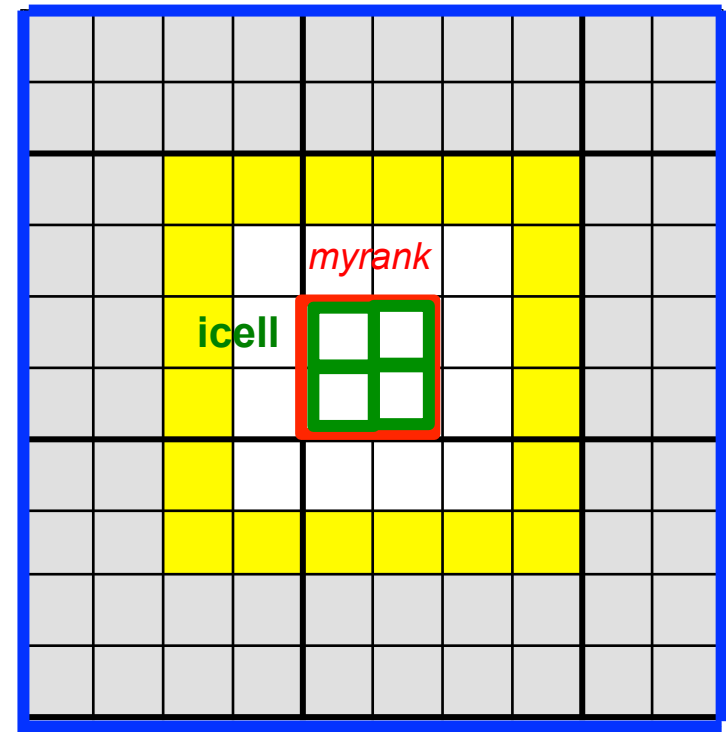
ブロック化前のループ構造 (M2L):

```

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

```

$wm_local((nmax+1)^2, 10, 10)$



問題点

- **icell** が変わると右図灰色の領域の **wm_local**, 変換行列 **m2l** は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。

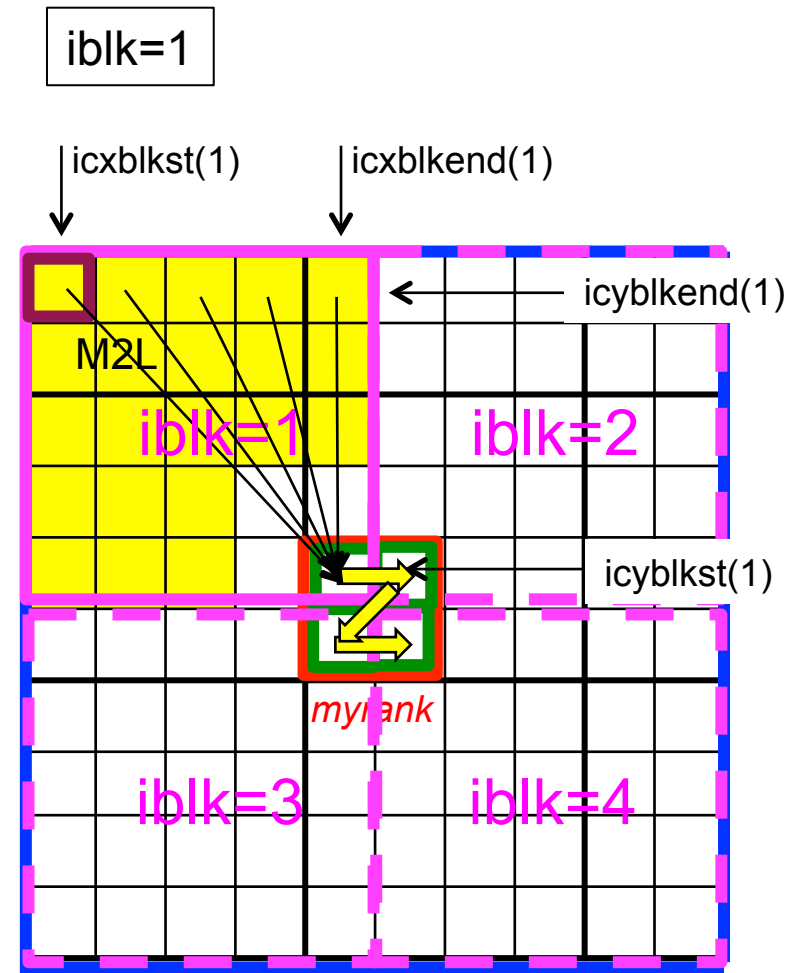
ブロック化によるキャッシュの有効利用 (2)

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

```



- **iblk**番目のブロック内**wm_local**, 変換行列**m2l**は **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。

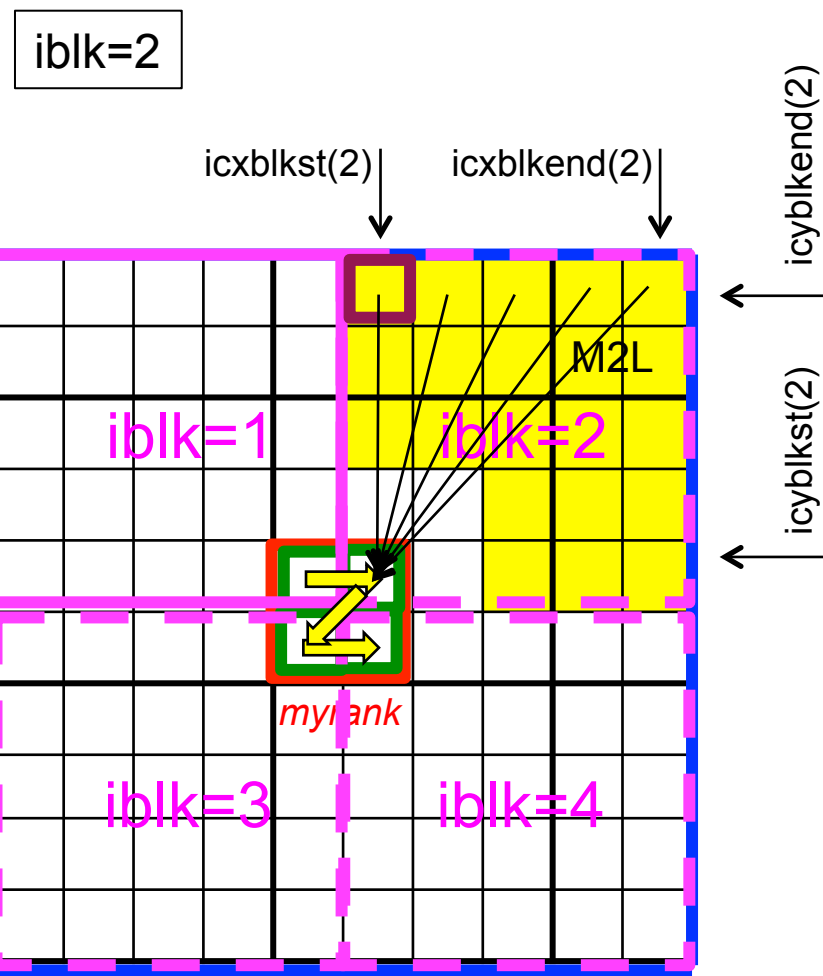
ブロック化によるキャッシュの有効利用 (2)

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

```



- **iblk**番目のブロック内**wm_local**, 変換行列**m2l**は **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

OpenMP 言語拡張機能

- 共有メモリー型並列化のための規格
- 商用コンパイラ (ifort, pgf90, frtpx) に標準的に備わっている
→ 表2のコンパイルオプションを入れることで有効化
- 「**スレッド**」と呼ばれる単位にタスクが割り振られる
- !\$omp (Fortran) ないし#pragma omp (C言語) ではじまる各種の指示文 (ディレクティブ) をプログラムに追記する **[do/for ループごと]**
- コンパイラに自動 OpenMP 並列化のオプションがある
→ ただし実用上は配列への初期値代入程度にしか使えない

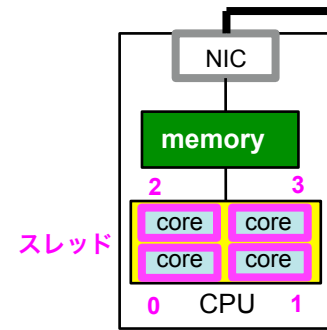


表2 OpenMP, SIMD並列化に関わるコンパイルオプション

コンパイラ名	OpenMP 並列化	OpenMP 自動並列化	SIMD 自動並列化	メッセージ出力
ifort	-openmp	-parallel	-xHOST	-openmp-report, -vec-report
pgf90	-mp	-Mconcur	-fastsse	-Minfo
frtpx	-Kopenmp	-Kparallel	-Ksimd	-Koptmsg

OpenMP 並列化技術

サンプル hello_omp.f:

```
include 'omp_lib.h'
```

OpenMP変数,関数の読み込み

```
nomp = omp_get_max_threads() スレッド数(nomp)取得 ← 環境変数 OMP_NUM_THREADS
```

```
!$omp parallel
```

```
iam = omp_get_thread_num()
```

自スレッド番号(iam)取得 *0 始まり

```
!$omp do
```

```
Do i=1,nomp
```

```
WRITE(*,*) 'Hello', iam
```

```
Enddo
```

```
!$omp end do
```

```
!$omp end parallel
```

```
STOP
```

```
END
```

コンパイル

```
> ifort -openmp hello_omp.f
```

実行

```
> export OMP_NUM_THREADS=4  
> ./a.out
```

```
Hello 0
```

```
Hello 1
```

```
Hello 2
```

```
Hello 3
```


OpenMP 並列化技術

• ロードインバランス調整 (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



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

```
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
```

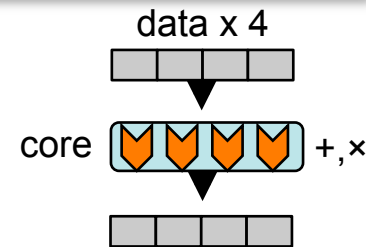
\$!omp parallelを大外に出す

```
!$omp parallel
do jcell_line
do icell      [along icell_line]
!$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
```

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

SIMD 並列化技術

SIMD (single instruction multiple data)



- 複数のデータに対する単一命令実行
- ハードウェアに固有の拡張命令セットを利用 (/proc/cpuinfo参照)
例) intel SSE pentium III~ 128bit (32bit X 4, 64 bit X 2)
AVX Sandy Bridge~ 256bit (32bit X 8, 64 bit X 4)
- SIMD 化を行なう方法:
 - コンパイラの SIMD 自動並列化機能にまかせる
コンパイルメッセージ, プロファイル結果を読みながら,
Fortran/C コードを調整し最適化 → 一般人
 - intrinsic 関数でコーディング → プロフェッショナル

* SIMDの高効率化はデータの連続化、キャッシュの有効利用がされている前提

表2 OpenMP, SIMD並列化に関わるコンパイルオプション

コンパイラ名	OpenMP 並列化	OpenMP 自動並列化	SIMD 自動並列化	メッセージ出力
ifort	-openmp	-parallel	-xHOST	-openmp-report, -vec-report
pgf90	-mp	-Mconcur	-fastsse	-Minfo
frtpx	-Kopenmp	-Kparallel	-Ksimd	-Koptmsg



SIMD 並列化技術

SIMD自動並列化の例

サンプル simd.f:

```
real(8)::a(10000,10000),b(10000)
real(8)::c(10000)
a=1d0
b=1d0
do i=1,10000
  c(i)=0d0
  do j=1,10000
    c(i)=c(i)+a(i,j)*b(j)
  enddo
enddo

stop
end
```

行列ベクトル積
 $C(i) = \sum A(i,j) * B(j)$

SIMD自動並列化無しコンパイル
 >ifort -O0 simd.f

```
>time ./a.out
real      0m1.782s
user      0m1.274s
sys       0m0.508s
```

SIMD自動並列化コンパイル

>ifort -xHOST -vec-report simd.f

```
simd.f(3): (col. 2) remark: LOOP WAS VECTORIZED.
simd.f(4): (col. 2) remark: LOOP WAS VECTORIZED.
simd.f(5): (col. 2) remark: PERMUTED LOOP WAS
VECTORIZED.
```

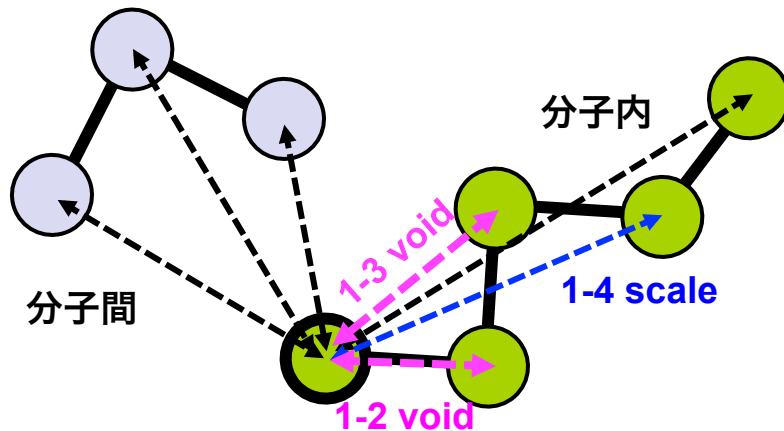
```
>time ./a.out
real      0m0.718s
user      0m0.397s
sys       0m0.321s
```

このコンパイルメッセージを読みながらコードを調整し最適化していく。
 ・計算の多い部分がVECTORIZEDされているか？

SIMD 並列化技術

• IF文の削除, ベクトル長の確保

話をだいぶ遡れば・・・



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

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

分子間

分子内

```
do imol=1,nmol-1
do jmol=imol+1,nmol
do i=1,natom(imol)
do j=1,natom(jmol)
```

$r_{ij}=r_{ij}(r_i, r_j)$

LJカットオフ判定

$\phi_{nonbond}=\phi_{nonbond}+\phi_{ij}$

$f(i)=f(i)+F_i$

$f(j)=f(j)+F_j$

enddo

enddo

enddo

enddo

```
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

$$x = \begin{cases} 0 & \text{if 1-2, -3 void} \\ s & \text{if 1-4 scale} \\ 1 & \text{else} \end{cases}$$

$\phi_{nonbond}=\phi_{nonbond}+x*\phi_{ij}$

$f(i)=f(i)+x*F_i$

$f(j)=f(j)+x*F_j$

enddo

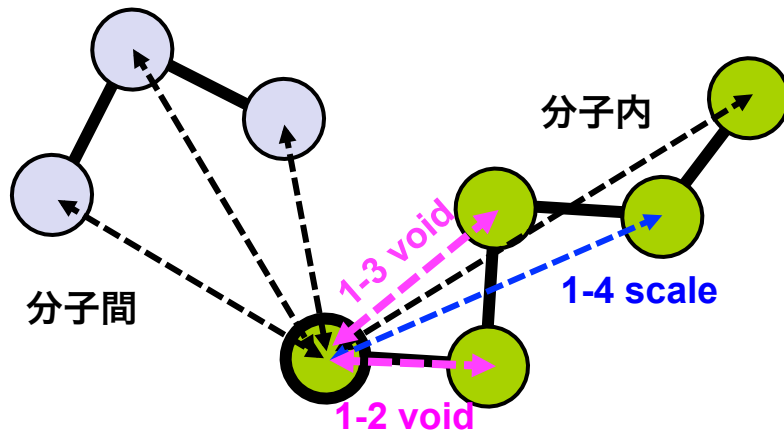
enddo

enddo

SIMD 並列化技術

・IF文の削除, ベクトル長の確保

話をだいぶ遡れば・・・



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

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

$$x = \begin{cases} 0 & \text{if 1-2, -3 void} \\ s & \text{if 1-4 scale} \\ 1 & \text{else} \end{cases}$$

$$\phi_{\text{nonbond}} = \phi_{\text{nonbond}} + x * \phi_{ij}$$

$$f(i) = f(i) + x * F_i$$

$$f(j) = f(j) + x * F_j$$

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

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
phi_nonbond=phi_nonbond+phi_ij
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)
phi_nonbond=phi_nonbond-phi_ij
f(i)=f(i)-Fi
f(j)=f(j)-Fj
enddo
do jatom=1,scalepair14(iatom)
rij=rij(ri,rj)
x=1-s
phi_nonbond=phi_nonbond-x*phi_ij
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           $     + wwl_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

MODYLAS
Molecular DYNAMics software for LARge System

www.modylas.org

HOME OVERVIEW **DOWNLOAD** **DOCUMENTATION** RELEASE NOTE FORUMS LITERATURE DEVELOPERS CONTACT LINKS

Home » MODYLAS

MODYLAS

VERSION: 0.9.0beta
COMMENT: work on the K-Computer

If you agree [our license](#), please write your email address to send a URL for download, and then press the button below.

Your personal information filled out below will be used to send MODYLAS news and report activities.

For use of MODYLAS, please cite the following:

"MODYLAS: A Highly Parallelized General-Purpose Molecular Dynamics Simulation Program with Highly Scalable Fine-Grained New Parallel Processing Algorithms", *J. Chem. Theo. Comp.* 9, 32 (2009) Kojima, Atsushi Yamada, Susumu Okazaki, Kazutomo Kawaguchi, Hidemi Nagao, Kensuke Iwata, Yasuhiro Takeda, and Masao Fukushima

NAME * **ユーザー登録**

EMAIL *

Do NOT use free email address and mobile phone's email address.

ORGANIZATION *

TITLE *

CONFIRM PUBLICATION *

YES
 NO

Do you confirm that the name of your organization is published in modylas report? Please answer "Yes" if possible.

Submit

- ・マニュアル(PDF)
- ・チュートリアル(PDF)のダウンロード

差出人 modylas-admin@draco.ims.ac.jp★
件名 Download link for <http://www.modylas.org/>
宛先 yoshimichi.andoh@apchem.nagoya-u.ac.jp★

登録後このようなメールが送られてくる。
リンク先からソースコードをダウンロード

Dear visitor,

Thank you for your interest.

Please use the following link to download the files:

<http://www.modylas.org/node/19/download/a233f49281a202d6de3ca1a9ccfd2538>

This link will be accessible until Wed, 01/22/2014 - 13:31. If you need access after the link expires, don't hesitate to revisit the download page on <http://www.modylas.org/>

使い方については

第21回CMSI神戸ハンズオン:
MODYLASチュートリアル 資料
なども参照。