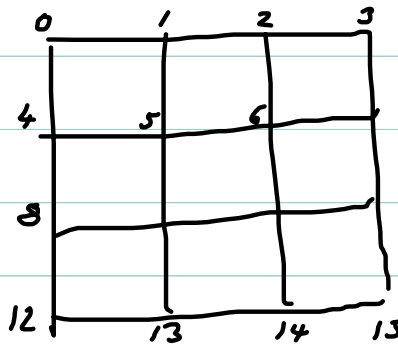
Tu. 05/22/12:

Ch. 7 Communicators and Topologies:
MPI_COMM_WORLD is the only communicator that exists initially. We can create additional ones as subsets of it.
Ex.: classic example to dedicate one process to do the I/O and exclude it from the calculations. If we want to run a method like power method, then you would call it only with the non-I/O processes.

Main example of Ch.7 is Fox's Algorithm → Sec. 7.2 and 7.3
Fox ⇐ matrix-matrix product $C = AB$, $A, B, C \in \mathbb{R}^{n \times n}$, with only local matrices l_A, l_B, l_C actually stored with a 2-D subdivision of A to l_A etc.
Assume p processes such that $p = q^2$ for integer q
ex.: $q = 1, 2, 3, 4, 5, 6, 7, 8$
$\Leftrightarrow p = 1, 4, 9, 16, 25, 36, 49, 64$
only possible ones currently
If $A \in \mathbb{R}^{n \times n}$, then $l\_A \in \mathbb{R}^{l\_n \times l\_n}$ with $l\_n = n/q$
⇒ reasonable q values only even ones
Also want to use all cores on each node, i.e., want
$p = 1, 4, 16, 36, 64$
$\Leftrightarrow$ 1, 1, 2, 9, 16 nodes needed
Message simply: We need to put some thought into designing the numerical experiments by choosing good n and p.

Communicators: how to create subdivisions

Idea: Think of p processes as a 2-D grid



Want to create communicators for each row and column.

Option 1: most general MPI command

Specify MPI ranks (id) to include in the communicator

```
MPI_Group group_world
MPI_Comm_group ( MPI_COMM_WORLD, & group_world )
```

We cannot access a communicator directly, but we have to do it via a Group object.

Ex.: create Comm for 2nd row: In example above, 2nd row is id = 4, 5, 6, 7 if $q = 4$ $\Leftrightarrow$ id = k for $q \le k < 2q$

```
for ( k = 0; k < q; k++)
  process_rank[k] = q + k
MPI_Comm_incl ( group_world, q, process_rank,
                & row2_comm_group )
MPI_Comm_create ( & row_comm_group)
```

→ Need to do this manually for each row and column.

Option 2 : Use condition to split up communicator

=> Can create several communicator at once, all with same variable name

```
MPI_Comm_split(
    MPI_Comm        old_comm,
    int             split_key
    int             rank_key
    MPI_Comm        *new_comm )
```

All process in old_comm with same split_key are put in one communicator, with their ranks ordered according to rank_key.

```
my_row = id / q
my_col = id % q
MPI_Comm_split ( MPI_COMM_WORLD,
    my_row, my_col, &row_comm )
MPI_Comm_split ( MPI_COMM_WORLD,
    my_col, my_row, &col_comm )
```

Notice all communicator created in one shot.

Option 3: Topologies of communicator explicitly part of MPI

Command: Cartesian grids

```
MPI_Comm    grid_comm
int         ndims, dimsize[2], wrap_around[2],
            reorder
```

ndims = 2

dim_size[0] = dimsize[1] = q

wrap_around[0] = 0 <=> do not wrap around

wrap_around[1] = 1 <=> wrap around in 2nd dimension

reorder = 1 <=> allow for ordering of the MPI process at run time

```
MPI_Cart_create( MPI_COMM_WORLD, ndims,
  dim_size, wrap_around, reorder, &grid_comm)
MPI_Cart_coords(
      MPI_Comm      grid_comm,
      int           my_grid_rank,
input → int         ndims,
vector → int        *coords)
and output
```

Ex.: q=4, np=16, ndims=2, my_grid_rank=5 => coords={1,1}

```
MPI_Cart_rank(
      MPI_Comm      grid_comm,
vector input → int  *coords
scalar output → int *grid_rank)
```

MPI_Cart_sub (

MPI_Comm grid_comm
int *free_coords
MPI_Comm *new_comm)

if free_coords[i] = 0, then this coordinate is not fro
⇐ if remain_coords[i] = 1, then this coordinate remains in the new comunicator

⇒ row_comm by { remain_coords[0] = 0
[1] = 1

col_comm by { [0] = 1
[1] = 0

→ pp. 125 - 126 setup_grid

Sec. 7.2 Fox's Algorithm

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} & \cdots \\ A_{10} & A_{11} & A_{12} & \cdots \\ \vdots & & & \\ A_{q-1\,0} & \cdots & & \end{bmatrix}$$

$$AB = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

$$= \begin{bmatrix} A_{00}B_{00} + A_{01}B_{10} & A_{00}B_{01} + A_{01}B_{11} \\ A_{10}B_{00} + A_{11}B_{10} & \cdots \end{bmatrix}$$

$= C = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix}$

$\Rightarrow$ Message: block matrix algebra works!
(Pacheco shows this with scalar blocks.)

$\Leftrightarrow$ For each block in C:

$C_{ij} = A_{i0}B_{0j} + A_{i1}B_{1j} + \cdots + A_{ii}B_{ij} + \cdots + A_{ij-1}B_{j-1j}$

$= A_{ii}B_{ij} + A_{ii+1}B_{i+1j} + \cdots + A_{ij-1}B_{j-1j} + A_{i0}B_{0j} + \cdots$

This shows the idea of Fox:
(i,j) is the 2-D process ID, have B_{ij} on the local process, so start accumulating the sum by the $A_{ii}B_{ij}$ term. Then communication will be needed to get the right pieces of A and B for each step.

Want as result l_C = C_{ij} on Process (i,j).
by accumulating local products: l_C += l_A*l_B
Initially l_B = B_{ij}, so no communication needed,
but we need to communicate along row to l_A = A_{ii}

Code:

```
l_A_original = l_A
l_C = 0        initialize
for (istep = 0; istep < q; istep++) {
    MPI_Bcast(l_A, l_n*l_n, MPI_DOUBLE, (i+istep)%q,
              row_comm)
    l_C += l_A * l_B
    MPI_Sendrecv_replace(l_B, l_n*l_n, MPI_DOUBLE,
        dest = (i-1+q)%q, source = (i+1)%q, col_comm)
    l_A = l_A_original
}
```

This uses some ideas from the suggested in Ch. 3

Pacheco does <u>not</u> have l_A_original, but he uses if (i == (i+istep)%q) MPI_Bcast(l_A, ...)
else MPI_Bcast(l_D, ...)
then l_C += l_D * l_B

MPI_Sendrecv_replace <u>needs</u> an auxiliary buffer =>
This code needs memory for <u>5</u> matrices of size $(l\_n)^2$
(l_A, l_B, l_C, l_A_original, plus aux. buffer)

BLAS = basic linear algebra subprograms

level 1 (=) 1 for loop like in a dot product

2 (=) 2 " " " " matrix-vector product

3 (=) 3 " " " " matrix-matrix product

BLAS dgemm = matrix-matrix product

function uses a block algorithm like Fox and is much more efficient than any ordering of for loops can be.

When doing code development, think carefully about each sub-problem, e.g., here the L_C += L_A * L_B ⟶ realize BLAS3 is better than any naive code, so use it!