Communicators and Topologies: Matrix Multiplication Example

Fox’a algorithm

A and B are n x n matrices

Compute C = AB in parallel

Let q = sqrt(p) be an integer such that it divides n, where p is the number of processes

Create a Cartesian topology with processes (i,j), i,j= 0,...,q-1

Denote nb = n/q

Distribute A and B by blocks on p processes such that $A_{i,j}$ and $B_{i,j}$ are nb x nb blocks stored on process (i,j)

Algorithm goal

Perform the operation with as few communications as possible

Have a clear communication scheme that results in effective code

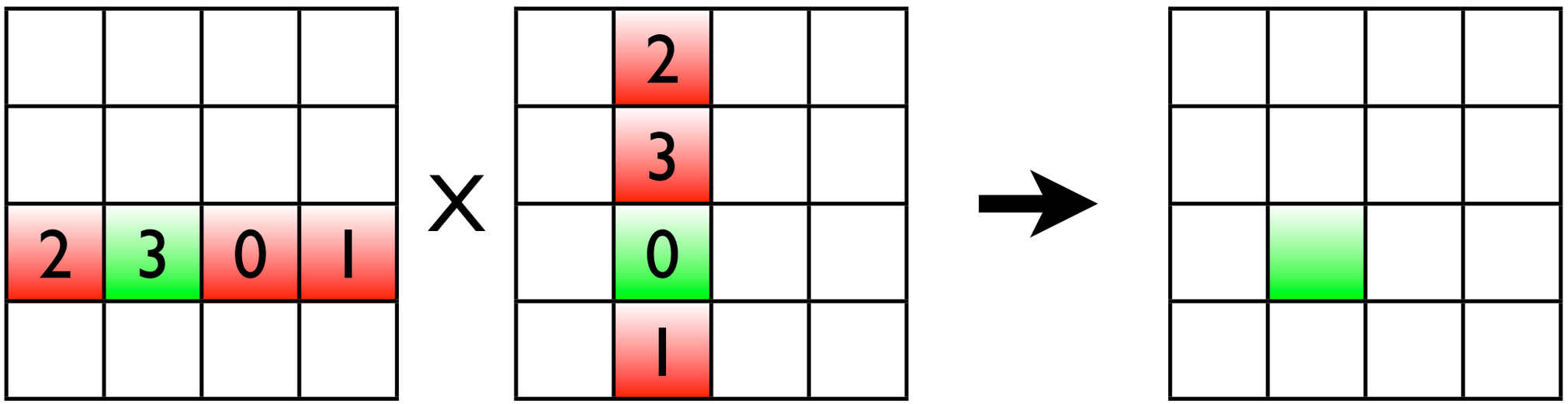
On process (i,j), we want to compute

$C_{i,j} = A_{i,0}B_{0,j} + A_{i,1}B_{1,j} + \ldots + A_{i,n-1}B_{n-1,j}$

Rewrite this as

stage 0	$C_{i,j} \leftarrow A_{i,i}B_{i,j}$
stage 1	$C_{i,j} \leftarrow C_{i,j} + A_{i,i+1}B_{i+1,j}$
...	...
stage	$C_{i,j} -< C_{i,j} + A_{i,q-1}B_{q-1,j}$
stage	$C_{i,j} -< C_{i,j} + A_{i,0}B_{0,j}$
stage	$C_{i,j} -< C_{i,j} + A_{i,1}B_{1,j}$
...	...
stage q-1	$C_{i,j} -< C_{i,j} + A_{i,i-1}B_{i-1,j}$

Example: 4 x 4 block matrix



Highlighted blocks need to compute result block for process

Available to process at start

Has to be received from other processors

Numbers indicate order in which operations are done

Initially, assume $C_{i,j} = 0$, the zero block, on each (i,j)

Process (i,j) at stage 0:
- (i,j) has $B_{i,j}$
- (i,j) needs $A_{i,i}$ which it has to get from process (i,i)
- (i,i) broadcasts $A_{i,i}$ across processor row i
- $C_{i,j}$ <- $C_{i,j}$ + $A_{i,i}B_{i,j}$

Process (i,j) at stage 1:
- (i,j) has $B_{i,j}$, but needs $B_{i+1,j}$
  Shift the j-th block column of B by one block up
  Block 0 goes to block q-1
- (i,j) needs $A_{i,i+1}$ which it has to get from process (i,i+1)
- (i,i+1) broadcasts $A_{i,i+1}$ across processor row i
- $C_{i,j}$ <- $C_{i,j}$ + $A_{i,i+1}B_{i+1,j}$

Similarly on next stages

Algorithm outline

On process (i,j)

q=sqrt(p)

for s=0 to q-1

k = (i+s) mod q

broadcast $A_{i,k}$ across row i

$C_{i,j}$ <- $C_{i,j}$ + $A_{i,k}B_{k,j}$

send $B_{k,j}$ to ((i-1) mod q , j)

receive $B_{k+1,j}$ from ((i+1) mod q,j)

Implementation (just essential parts)

```
/* setupgrid.c */
void Setup_grid(GRID_INFO_T*  grid)
{
  int old_rank;
  int dimensions[2];
  int wrap_around[2];
  int coordinates[2];
  int free_coords[2];

/* Set up Global Grid Information */
MPI_Comm_size(MPI_COMM_WORLD, &(grid->p));
MPI_Comm_rank(MPI_COMM_WORLD, &old_rank);

/* We assume p is a perfect square */
grid->q = (int) sqrt((double) grid->p);
dimensions[0] = dimensions[1] = grid->q;

/* We want a circular shift in second dimension. */
/* Don't care about first                        */
wrap_around[0] = wrap_around[1] = 1;
```

```
MPI_Cart_create(MPI_COMM_WORLD, 2, dimensions,
                wrap_around, 1, &(grid->comm));

MPI_Comm_rank(grid->comm, &(grid->my_rank));
MPI_Cart_coords(grid->comm, grid->my_rank, 2,
                coordinates);

grid->my_row = coordinates[0];
grid->my_col = coordinates[1];

/* Set up row communicators */
free_coords[0] = 0;
  free_coords[1] = 1;
  MPI_Cart_sub(grid->comm, free_coords,
               &(grid->row_comm));

  /* Set up column communicators */
  free_coords[0] = 1;
  free_coords[1] = 0;
  MPI_Cart_sub(grid->comm, free_coords,
               &(grid->col_comm));
}
```

```
void Fox(int n,GRID_INFO_T* grid,
         LOCAL_MATRIX_T* local_A, LOCAL_MATRIX_T* local_B,
         LOCAL_MATRIX_T* local_C)
{
  LOCAL_MATRIX_T*  temp_A;
  int              stage;
  int              bcast_root;
  int              n_bar;  /* n/sqrt(p)                */
  int              source;
  int              dest;
  MPI_Status       status;

n_bar = n/grid->q;
Set_to_zero(local_C);

/* Calculate addresses for circular shift of B */
source = (grid->my_row + 1) % grid->q;
dest = (grid->my_row + grid->q - 1) % grid->q;

/* Set aside storage for the broadcast block of A */
temp_A = Local_matrix_allocate(n_bar);
```

```
for (stage = 0; stage < grid->q; stage++)
  {
    bcast_root = (grid->my_row + stage) % grid->q;
    if (bcast_root == grid->my_col)
      {
        MPI_Bcast(local_A, 1, local_matrix_mpi_t,
                  bcast_root, grid->row_comm);
        Local_matrix_multiply(local_A, local_B,
                              local_C);
      }
    else
      {
        MPI_Bcast(temp_A, 1, local_matrix_mpi_t,
                  bcast_root, grid->row_comm);
        Local_matrix_multiply(temp_A, local_B,
                              local_C);
      }
    MPI_Sendrecv_replace(local_B, 1, local_matrix_mpi_t,
                         dest, 0, source, 0,
                         grid->col_comm, &status);
  }
}
```