CME 213

Spring 2012-2013

Eric Darve

Stanford University

MPI

COLLECTIVE COMMUNICATION

Broadcast and reduce

$$
\begin{array}{l}
P_1 : \boxed{x} \\
P_2 : \boxed{-} \\
\vdots \\
P_p : \boxed{-}
\end{array}
\quad \overset{broadcast}{\Longrightarrow} \quad
\begin{array}{l}
P_1 : \boxed{x} \\
P_2 : \boxed{x} \\
\vdots \\
P_p : \boxed{x}
\end{array}
$$

$$
\begin{array}{l}
P_1 : \boxed{x_1} \\
P_2 : \boxed{x_2} \\
\vdots \\
P_p : \boxed{x_p}
\end{array}
\quad \overset{accumulation}{\Longrightarrow} \quad
\begin{array}{l}
P_1 : \boxed{x_1 + x_2 + \cdots + x_p} \\
P_2 : \boxed{x_2} \\
\vdots \\
P_p : \boxed{x_p}
\end{array}
$$

Gather/Scatter

$$\begin{array}{ll} P_1 : \boxed{x_1} \\ P_2 : \boxed{x_2} \\ \vdots \\ P_p : \boxed{x_p} \end{array} \quad \overset{gather}{\Longrightarrow} \quad \begin{array}{ll} P_1 : \boxed{x_1 \parallel x_2 \parallel \cdots \parallel x_p} \\ P_2 : \boxed{x_2} \\ \vdots \\ P_p : \boxed{x_p} \end{array}$$

`MPI_Gather()`

$$\begin{array}{ll} P_1 : \boxed{x_1 \parallel x_2 \parallel \cdots \parallel x_p} \\ P_2 : \boxed{-} \\ \vdots \\ P_p : \boxed{-} \end{array} \quad \overset{scatter}{\Longrightarrow} \quad \begin{array}{ll} P_1 : \boxed{x_1} \\ P_2 : \boxed{x_2} \\ \vdots \\ P_p : \boxed{x_p} \end{array}$$

`MPI_Scatter()`

Multi-broadcast/Multi-accumulation

$$
\begin{array}{ll}
P_1 : \boxed{x_1} \\
P_2 : \boxed{x_2} \\
\vdots \\
P_p : \boxed{x_p}
\end{array}
\xRightarrow{multi\text{-}broadcast}
\begin{array}{ll}
P_1 : \boxed{x_1 \parallel x_2 \parallel \cdots \parallel x_p} \\
P_2 : \boxed{x_1 \parallel x_2 \parallel \cdots \parallel x_p} \\
\vdots \\
P_p : \boxed{x_1 \parallel x_2 \parallel \cdots \parallel x_p}
\end{array}
$$

`MPI_Allgather()`

$$
\begin{array}{ll}
P_1 : \boxed{x_{11} \parallel x_{12} \parallel \cdots \parallel x_{1p}} \\
P_2 : \boxed{x_{21} \parallel x_{22} \parallel \cdots \parallel x_{2p}} \\
\vdots \\
P_p : \boxed{x_{p1} \parallel x_{p2} \parallel \cdots \parallel x_{pp}}
\end{array}
\xRightarrow{multi\text{-}accumulation}
\begin{array}{ll}
P_1 : \boxed{x_{11} + x_{21} + \cdots + x_{p1}} \\
P_2 : \boxed{x_{12} + x_{22} + \cdots + x_{p2}} \\
\vdots \\
P_p : \boxed{x_{1p} + x_{2p} + \cdots + x_{pp}}
\end{array}
$$

MPI provides a restricted version: **`MPI_Allreduce()`**, which is a reduce followed by a broadcast .

TOTAL EXCHANGE

$$
\begin{array}{rl}
P_1 : & \boxed{x_{11} \parallel x_{12} \parallel \cdots \parallel x_{1p}} \\
P_2 : & \boxed{x_{21} \parallel x_{22} \parallel \cdots \parallel x_{2p}} \\
\vdots & \\
P_p : & \boxed{x_{p1} \parallel x_{p2} \parallel \cdots \parallel x_{pp}}
\end{array}
\quad \xRightarrow{\textit{total exchange}} \quad
\begin{array}{rl}
P_1 : & \boxed{x_{11} \parallel x_{21} \parallel \cdots \parallel x_{p1}} \\
P_2 : & \boxed{x_{12} \parallel x_{22} \parallel \cdots \parallel x_{p2}} \\
\vdots & \\
P_p : & \boxed{x_{1p} \parallel x_{2p} \parallel \cdots \parallel x_{pp}}
\end{array}
$$

`MPI_Alltoall()`

Other collective operations

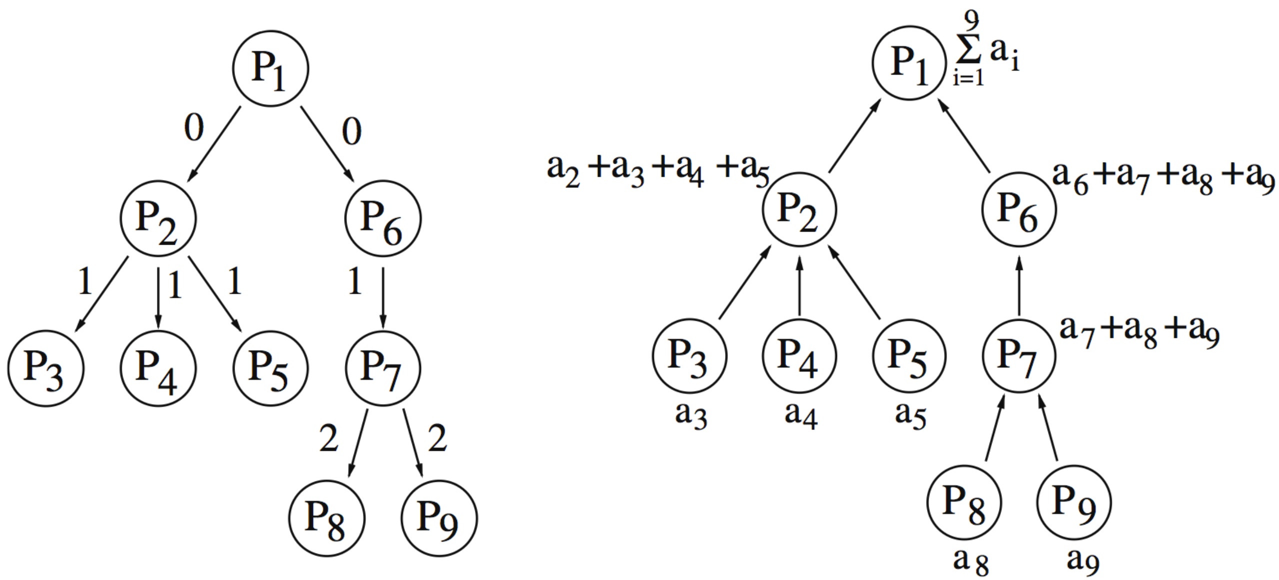
```
MPI_Barrier(MPI_Comm comm)
```

All processes belonging to `comm` are blocked until all other processes of this group have called this operation.

CONCEPTUAL RELATIONS BETWEEN COLLECTIVE COMMUNICATION

DUALITY

- Some communication operations are dual of each other.
- Communication operations are dual if one can be obtained by reversing the direction and the sequence of communication of the other.

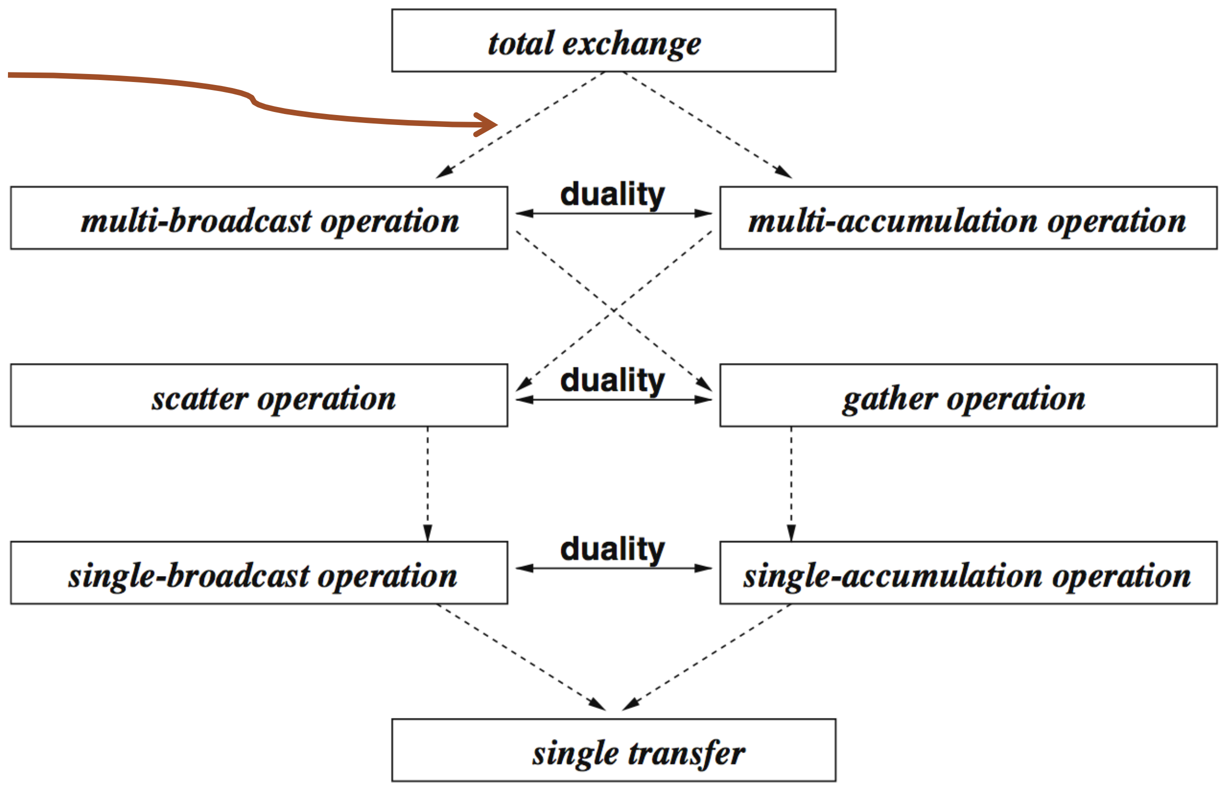


Left: single-broadcast operation using a spanning tree
Right: single-accumulation that uses the same communication tree.

Relation between collective communication operations

Important to understand performance and if you are interested in how these collective communication operations are implemented.

Specialization, e.g., multi-broadcast is the same as total exchange if the p data blocks of a process are the same.



Matrix-vector product

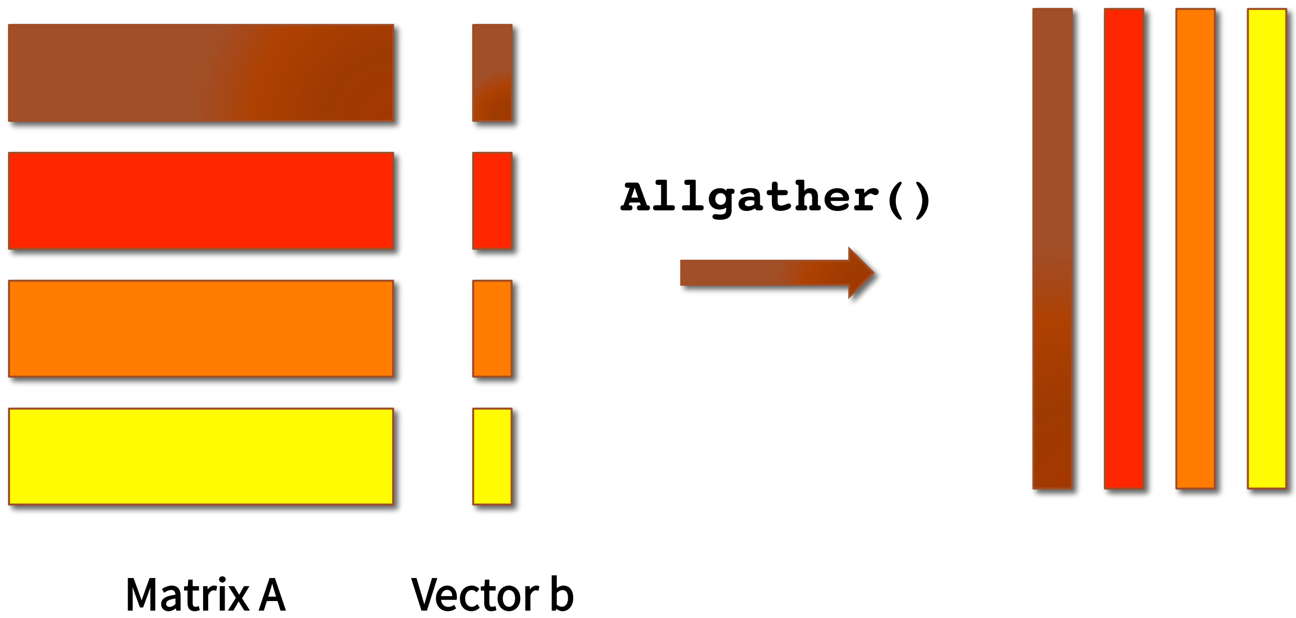
Matrix-vector product

- We are going to use that example to illustrate additional MPI functionalities.
- Before explaining process groups and topologies, we go over two implementations that use the functionalities we already know.
- Two simple approaches:

1. Row partitioning of the matrix, or
2. Column partitioning

Row partitioning

This is the most natural.



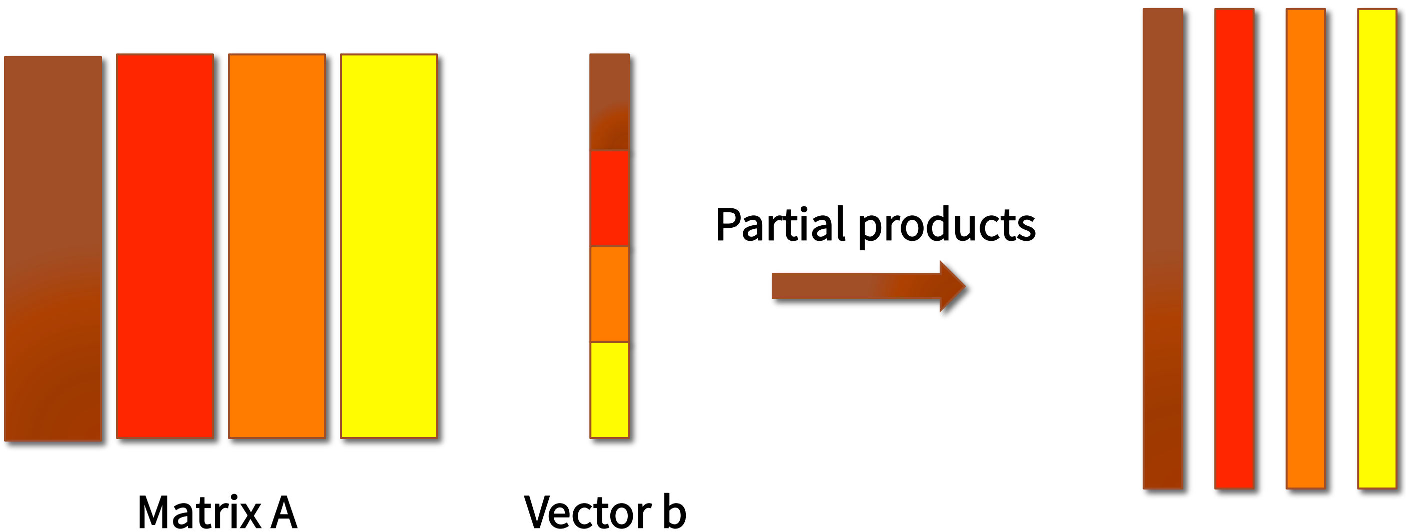
Step 1: replicate b on each process:

```
MPI_Allgather()
```

Step 2: perform product

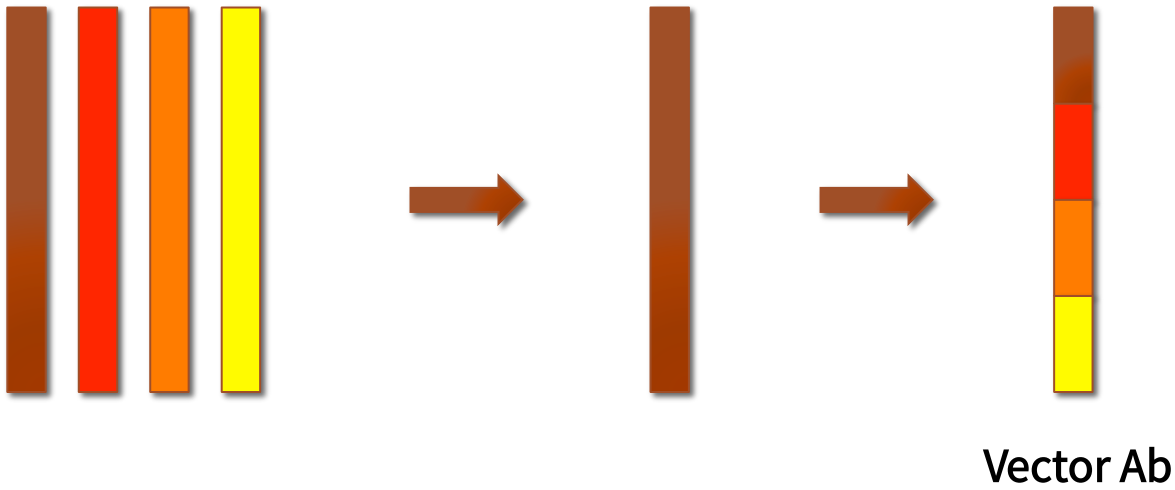
See MPI code

COLUMN PARTITIONING



Step 1: calculate partial products with each process

COLUMN PARTITIONING (CONT'D)

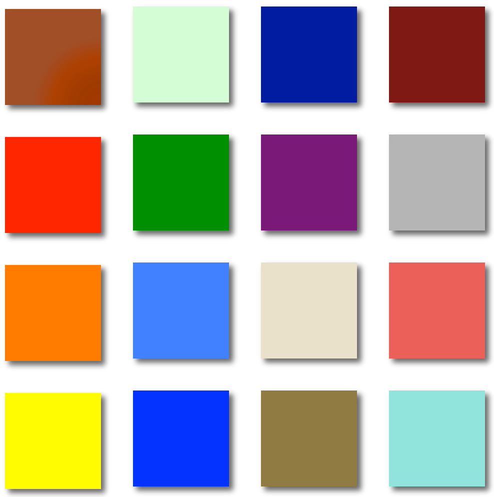


Step 2: reduce all partial results **`MPI_Reduce()`**

Step 3: send sub-blocks to all processes **`MPI_Scatter()`**

A BETTER PARTITIONING

If the number of processes becomes large compared to the matrix size, we need a 2D partitioning:



To operate on such a data partitioning, we need two things:

1. Communicators (process groups)
2. Process topologies

PROCESS GROUPS AND COMMUNICATORS

GROUPS AND COMMUNICATORS

- A group is an ordered set of processes.
- Each process in a group is associated with a unique integer rank. Rank values start at zero and go to N-1, where N is the number of processes in the group.
- A group is always associated with a communicator object.
- A communicator encompasses a group of processes that may communicate with each other. All MPI messages must specify a communicator. In the simplest sense, the communicator is an extra "tag" that must be included with MPI calls.
- For example, the handle for the communicator that comprises all tasks is **`MPI_COMM_WORLD`**.
- From the programmer's perspective, a group and a communicator are one. The group routines are primarily used to specify which processes should be used to construct a communicator.

PURPOSE OF GROUPS AND COMMUNICATORS

- Enables collective communications operations across a subset of processes.
- Provides basis for implementing user defined virtual topologies, e.g., to perform our matrix-vector product with 2d partitioning.
- Allows to easily assign independent tasks to different groups of processes.
- Provide for safe communications, e.g., to avoid interference with parallel libraries.

Note: processes may be in more than one group/communicator. They will have a unique rank within each group/communicator.

MAIN FUNCTIONS

MPI provides over 40 routines related to groups, communicators, and virtual topologies!

```
int MPI_Comm_group(MPI_Comm comm, MPI_Group *group)
```

Extract group associated with communicator, e.g., **`MPI_COMM_WORLD`**

```
int MPI_Group_incl(MPI_Group group, int p, int *ranks,
       MPI_Group *new_group)
```

`ranks` integer array with p entries.

Creates a new group **`new_group`** with p processes which have ranks from 0 to p-1. Process i is the process which has rank **`ranks[i]`** in **`group`**.

```
int MPI_Comm_create(MPI_Comm comm, MPI_Group group,
       MPI_Comm *new_comm)
```

New communicator based on **`group`**.

See MPI code.