TEXAS TECH UNIVERSITY™

# CS4379: Parallel and Concurrent Programming
# CS5379: Parallel Processing

## Lecture 24


**Dr. Yong Chen**

**Associate Professor**

**Computer Science Department**

**Texas Tech University**

# Lecture Video

- Please view the lecture video either from Teams or from the below link:

  https://texastechuniversity.sharepoint.com/sites/CS4379-CS5379/Shared%20Documents/General/Lecture24.mp4

# Course Info

- **Lecture Time**: TR, 12:30-1:50
- **Lecture Location**: ECE 217
- **Sessions**: CS4379-001, CS4379-002, CS5379-001, CS5379-D01
- **Instructor:** Yong Chen, Ph.D., Associate Professor
- **Email:** yong.chen@ttu.edu
- **Phone:** 806-834-0284
- **Office**: Engineering Center 315
- **Office Hours**: 2-4 p.m. on Wed., or by appointment
- **TA:** Mr. Ghazanfar Ali, Ghazanfar.Ali@ttu.edu
- **TA Office hours:** Tue. and Fri., 2-3 p.m., or by appointment
- **TA Office:** EC 201 A
- **More info:**
  - http://www.myweb.ttu.edu/yonchen
  - http://discl.cs.ttu.edu; http://cac.ttu.edu/; http://nsfcac.org

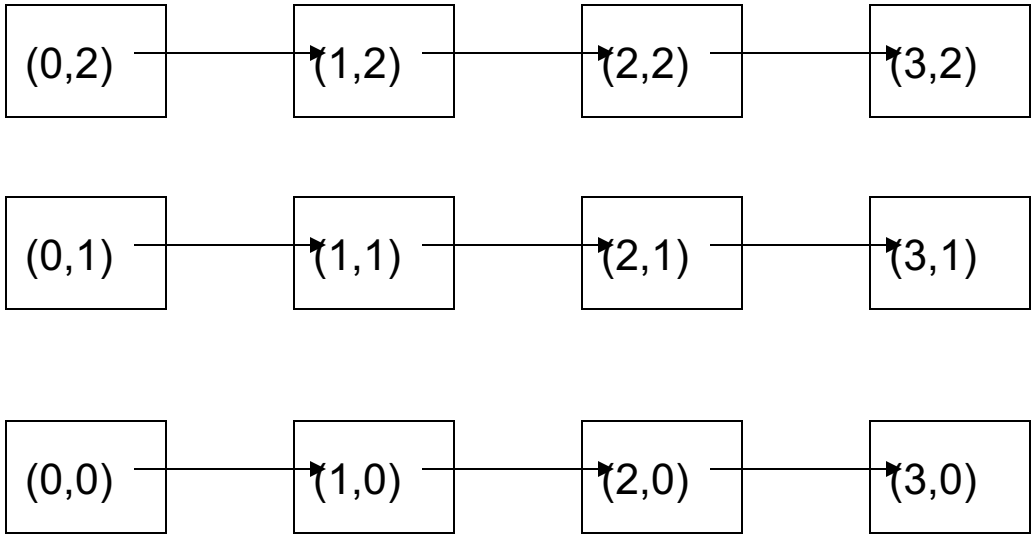
# Outline

- Questions?

- Topology and communicators
- Understanding the performance and behavior of MPI programs
- Compile and run MPI programs, demos

# Process Topology

- MPI lets user specify various application/virtual topologies
- A Cartesian topology is a mesh/grid
- Example: 3*4 Cartesian mesh with arrows pointing at the right neighbors:



# Defining a Cartesian Topology

- The routine MPI_Cart_create() creates a Cartesian decomposition of the processes, with the number of dimensions given by the ndim argument

```
dims[0]=4; dims[1]=3;
periods[0]=0; periods[1]=0; /* specify if wraparound */
reorder = 0;
ndim=2;
MPI_Cart_create(MPI_COMM_WORLD,ndim,*dims,*periods,reorder,comm2d);
```

- This creates a new communicator with the same processes as the input communicator, but with the specified topology

# Finding Neighbors

- The question “who are my neighbors?” can be answered with MPI_Cart_shift:

  *MPI_Cart_shift(comm2d, 0, 1, nbrleft, nbrright);*

  *MPI_Cart_shift(comm2d, 1, 1, nbrbottom, nbrtop);*

  *The values returned are the ranks, in the communicator comm2d, of the neighbors shifted by 1 in the two dimensions*

  *int MPI_Cart_shift(MPI_Comm comm, int direction, int disp, int *rank_source, int *rank_dest)*

- A neighbor may not exist. This is indicated by a rank of MPI_PROC_NULL

# Who Am I?

- This question can be answered with:

  *int coords[2];*

  *MPI_Comm_rank(comm2d, myrank);*

  *MPI_Cart_coords(comm2d, myrank,2, coords);*

  *returns the Cartesian coordinates of the calling process in coords.*

  *int MPI_Cart_coords ( MPI_Comm comm, int rank, int maxdims, int *coords )*

- Coordinate-to-rank translation

  *int MPI_Cart_rank ( MPI_Comm comm, int *coords, int *rank)*

# Other Topology Routines

- MPI_Graph_create allows the creation of a general graph topology
- MPI_Dist_graph_create is a more scalable version defined in MPI 2.2

- In summary, all these routines allow the MPI implementation to provide an ordering of processes in a virtual topology

# Communicators

- A communicator defines a *communication domain* - a set of processes that are allowed to communicate with each other.
- Information about communication domains is stored in variables of type `MPI_Comm`.
- Communicators are used as arguments to all message transfer MPI routines.
- A process can belong to many different (possibly overlapping) communication domains.
- MPI defines a default communicator called `MPI_COMM_WORLD` which includes all the processes.

```
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

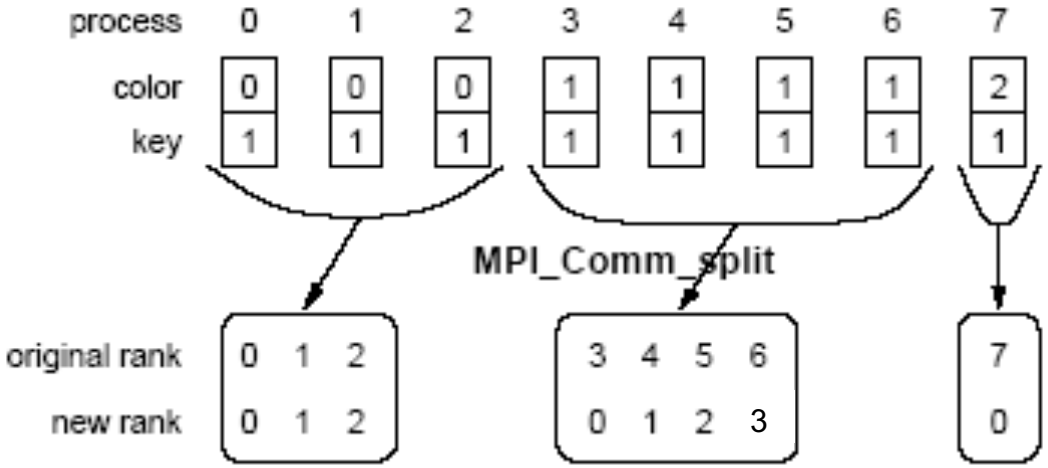
# Groups and Communicators

- In many parallel algorithms, communication operations need to be restricted to certain subsets of processes.
- MPI provides mechanisms for partitioning the group of processes that belong to a communicator into subgroups each corresponding to a different communicator.
- The simplest such mechanism is:

```
int MPI_Comm_split(MPI_Comm comm, int color, int key,
                   MPI_Comm *newcomm)
```

- This operation groups processes by color and sorts resulting groups on the key.

# Groups and Communicators




Using `MPI_Comm_split` to split a group of processes in a communicator into subgroups.

# Groups and Communicators

- MPI_Comm_dup() creates a duplicate of input communicator
- Duplicate has its own “context” – safe communication space
- Libraries should use MPI_Comm_dup() to get a private communicator

# More Advanced MPI Topics

- One-sided communication
- MPI I/O
- Dynamic process management
- Language bindings
- Etc.
- More can be found from: https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

# Outline

- Questions?

- Topology and communicators
- Understanding the performance and behavior of MPI programs
- Compile and run MPI programs, demos

# Timing MPI Programs

- The elapsed (wall-clock) time between two points in an MPI program can be computed using `MPI_Wtime`:

```
double t1, t2;
t1 = MPI_Wtime();
...
t2 = MPI_Wtime();
printf( "time is %d\n", t2 - t1 );
```

- The value returned by a single call to `MPI_Wtime` has little value.
- Times in general are local, but an implementation might offer synchronized times.
  - For advanced users: see the attribute `MPI_WTIME_IS_GLOBAL`.

# Sample Timing Harness

- Average times, make several trials

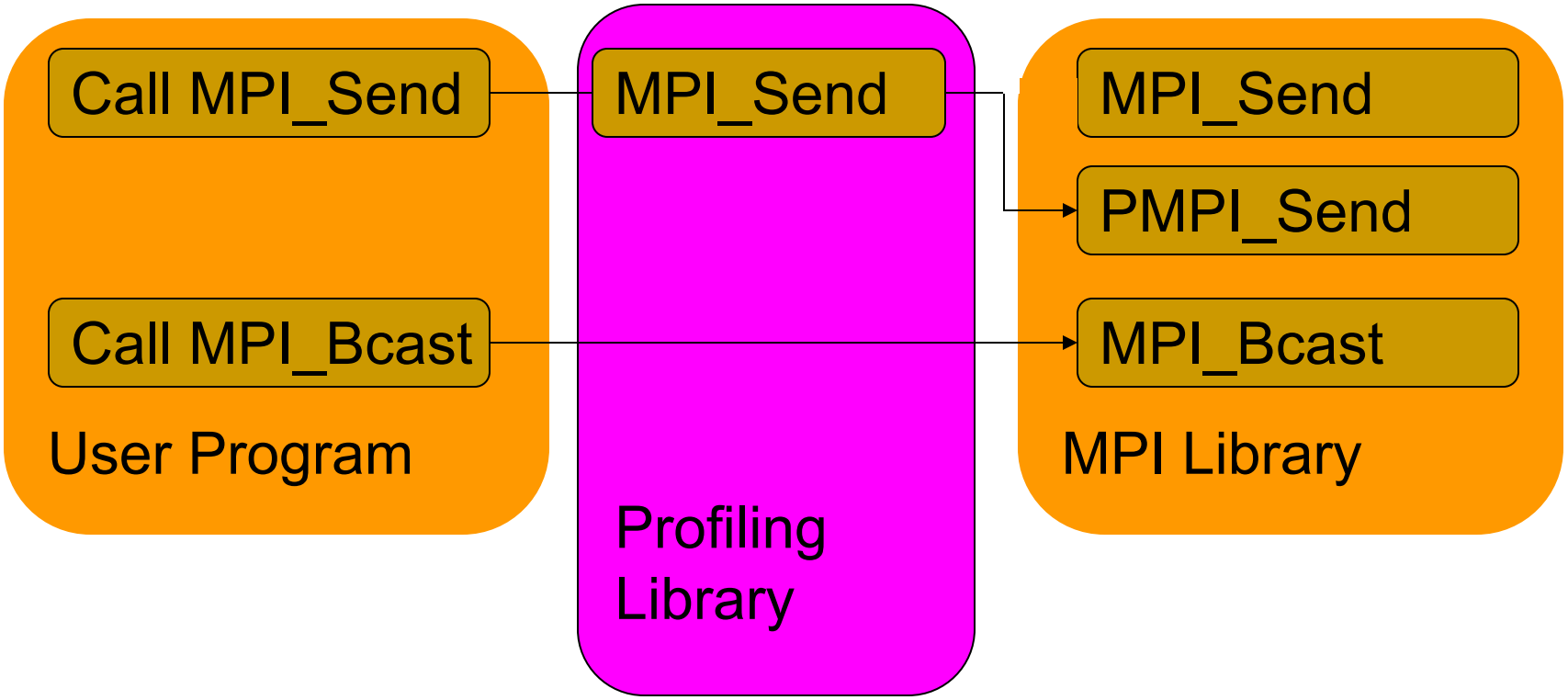
```
t1 = MPI_Wtime();
for (i < maxloop) {
    <operation to be timed>
}
time = (MPI_Wtime() - t1) / maxloop;
```

- Use MPI_Wtick to discover clock resolution
- Use getrusage (Unix) to get other effects (e.g., context switches, paging)

# MPI Profiling Interface (PMPI)

- Profiling intercepts calls and perform arbitrary actions
  - “wrappers”, perform time measurement, log record, or more
  - Then call the real functions
- PMPI allows selective replacement of MPI routines at link time
  - No need to recompile
- Every MPI function also exists under the name PMPI_

# Profiling Interface



# Example Use of Profiling Interface

```
static int nsend = 0;


int MPI_Send( void *start, int count, MPI_Datatype datatype, int dest,
    int tag, MPI_Comm comm )
{
   nsend++;
   return PMPI_send(start, count, datatype,
                    dest, tag, comm);
}
```

# Mechanisms of Using Profiling Interface

- Assume MPI and PMPI routines are in separate libraries
  - MPI: libmpi
  - PMPI: libpmpi
- At link time:
  - mpicc –o myprog myprog.o –lprof –lmpi –lpmpi
  - Resolve the reference to MPI routines from the profiling library first
- Usage:
  - Record logging
  - Finding deadlocks
  - Finding load imbalance

# MPI Implementations

- MPI is available on all platforms – from laptops to clusters to the largest supercomputers in the world

- Currently, three prominent open-source implementations
  - MPICH from Argonne
    - http://www.mpich.org/
  - Open MPI
    - www.open-mpi.org
  - MVAPICH from Ohio State Univ. for InfiniBand
    - http://mvapich.cse.ohio-state.edu/

# Outline

- Questions?

- Topology and communicators
- Understanding the performance and behavior of MPI programs
- Compile and run MPI programs, demos

# Using MPI Compilers on HPCC

| | | |
|---|---|---|
| **intel/18.0.3.222** | • **OpenMPI/1.10.7**<br>• impi/2018.3.222<br>• mvapich2/2.2 | • mpicc<br>• mpic++<br>• mpifort<br><br>• mpirun |
| gnu/5.4.0 | • OpenMPI/1.10.6<br>• impi/2018.3.222<br>• mvapich2/2.2 | |
| **gnu7/7.3.0** | • **OpenMPI/1.10.7**<br>• impi/2018.3.222<br>• mvapich2/2.2 | |

- **Load MPI Modules**
  - `$ module load intel openmpi`
- **Compile MPI program:**
  - `$ mpicc -o mpi-hello-world mpi-hello-world.c`
- **Run MPI program:**
  - `$ mpirun --machinefile machinefile.txt \`
    `-np 12 ./mpi_hello_world`

# Running MPI programs

- Create job submission script, e.g.

```
#!/bin/sh
#$ -V
#$ -cwd
#$ -S /bin/bash
#$ -N MPI_Job
#$ -o $JOB_NAME.o$JOB_ID
#$ -e $JOB_NAME.e$JOB_ID
#$ -q omni
#$ -pe mpi 72
#$ -l h_vmem=5.3G
#$ -l h_rt=48:00:00
#$ -P quanah

module load intel openmpi
mpirun --machinefile machinefile.$JOB_ID -np $NSLOTS ./mpi_hello_world
```

- Submit job
  - **qsub** <job submission script>

- Check job status
  - Command: **qstat**

# Demos

- mpi_hello.c
- mpi_hello_hostname.c

# Demos

- mpi_send.c
- mpi_bcast.c

# Demos

- mpi_matrixmul.c (using data decomposition and static, block distribution discussed in the earlier lecture)

# Readings

- Reference book ITPC – Chapter 6, 6.3-6.6
- Foster, DBPP, Chapter 8
  - http://www.mcs.anl.gov/~itf/dbpp/text/book.html
- MPI tutorials, https://computing.llnl.gov/tutorials/mpi/
- MPI 3.1 Specification, https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

# Questions?

## Questions/Suggestions/Comments are always welcome!

Write me: yong.chen@ttu.edu
Call me: 806-834-0284
See me: ENGCTR 315

*If you write me an email for this class, please start the email subject with [CS4379] or [CS5379].*