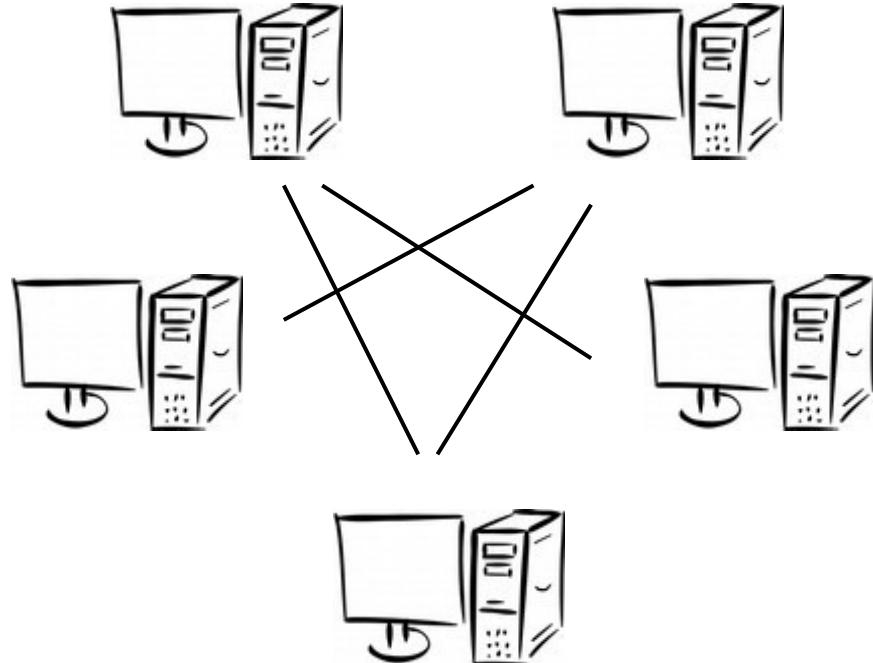


CS 470

Spring 2017

Mike Lam, Professor



Advanced MPI Topics

MPI datatypes

- MPI provides basic datatypes
 - `MPI_INT`, `MPI_LONG`, `MPI_CHAR`, etc.
- MPI also provides ways to create new datatypes
 - `MPI_Type_contiguous`: simple arrays
 - `int MPI_Type_contiguous(int count, MPI_Datatype oldtype, MPI_Datatype *newtype)`
 - `MPI_Type_vector`: blocked and strided arrays
 - Useful for cyclic or block-cyclic data distributions
 - `int MPI_Type_vector(int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)`
 - **Derived datatypes**: records
 - New datatypes must be committed before they are used
 - `int MPI_Type_commit(MPI_Datatype *datatype)`

Derived datatypes

- Goal: Pack related data together to reduce total messages
 - Very similar to C structs, but more detailed
 - Allows MPI to optimize internal representations



```
MPI_Type_create_struct(5, array_of_block_lengths,  
                      array_of_displacements,  
                      array_of_types,  
                      &new_type)
```

```
array_of_block_lengths = (1, 2, 2, 1, 1)  
array_of_displacements = (0, 4, 8, 16, 20)  
array_of_types = (MPI_LB, MPI_CHAR, MPI_FLOAT, MPI_SHORT, MPI_UB)
```

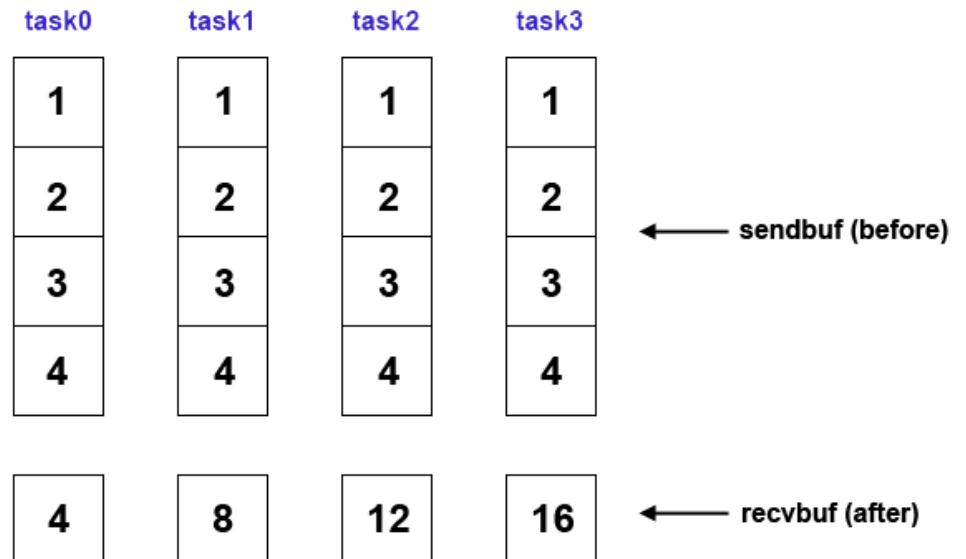
MPI safety

- An MPI program is **unsafe** if it relies on MPI-provided buffering
 - Recall that `MPI_Send` has ambiguous blocking/buffering behavior
 - If you rely on it to buffer every time, your program is not safe
 - Use `MPI_Ssend` to test for safety
 - Use `MPI_Sendrecv` for matching send/recv pairs if possible
 - Or use `MPI_Isend` / `MPI_Recv` pairs

More collectives

- **MPI_Reduce_scatter**
 - Reduce on a vector, then distribute result

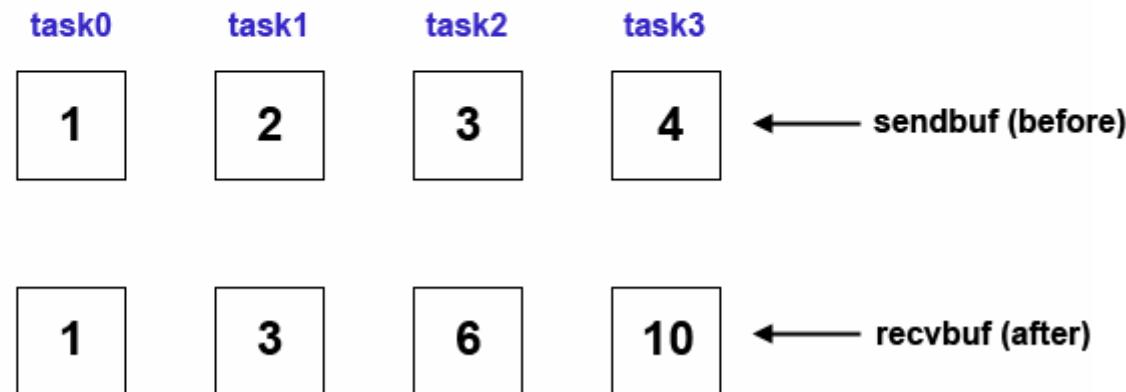
```
recvcnt = 1;  
MPI_Reduce_scatter(sendbuf, recvbuf, recvcount,  
                    MPI_INT, MPI_SUM, MPI_COMM_WORLD);
```



More collectives

- MPI_Scan
 - Compute partial reductions

```
count = 1;  
MPI_Scan(sendbuf, recvbuf, count, MPI_INT,  
         MPI_SUM, MPI_COMM_WORLD);
```



Virtual topologies

- It is often convenient for MPI to be aware of data decomposition details
- MPI provides built-in Cartesian system support.
 - `MPI_Dims_create()`
 - `MPI_Cart_create()`
 - `MPI_Cart_get()`
 - `MPI_Cart_coords()`
 - `MPI_Cart_shift()`

0 (0,0)	1 (0,1)	2 (0,2)	3 (0,3)
4 (1,0)	5 (1,1)	6 (1,2)	7 (1,3)
8 (2,0)	9 (2,1)	10 (2,2)	11 (2,3)
12 (3,0)	13 (3,1)	14 (3,2)	15 (3,3)

Parallel file I/O (MPI-2)

- MPI provides a parallel file I/O interface
 - Uses derived data types to create per-process views of a file on disk
 - `MPI_File_open()`
 - `MPI_File_set_view()`
 - `MPI_File_read_at()`
 - `MPI_File_read()`
 - `MPI_File_read_shared()`
 - `MPI_File_write_at()`
 - `MPI_File_write()`
 - `MPI_File_write_shared()`
 - `MPI_File_close()`

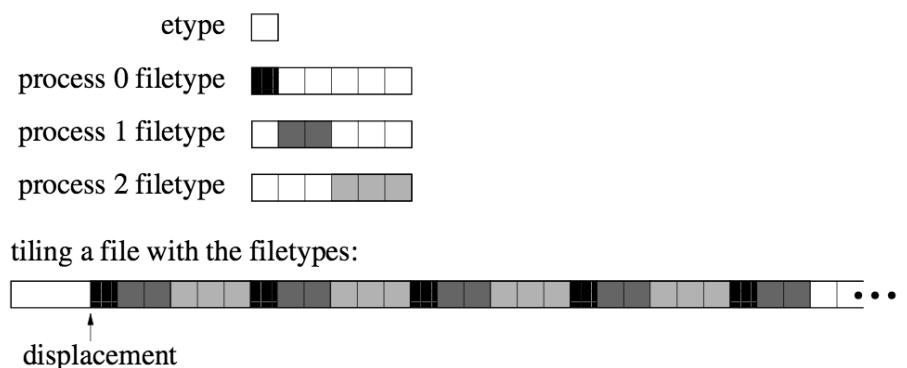


Figure 13.2: Partitioning a file among parallel processes

One-sided communication (MPI-2)

- MPI provides remote memory access (RMA)
 - This allows programmers to take advantage of hardware-specific direct memory access features like DMA
 - `MPI_Win_create()`
 - `MPI_Win_allocate()`
 - `MPI_Put()`
 - `MPI_Get()`
 - `MPI_Accumulate()`
 - `MPI_Win_free()`

Non-blocking collectives (MPI-3)

- MPI now provides non-blocking forms of major collective operations
- Like `MPI_Irecv()`, these calls begin the communication and should be concluded with a call to `MPI_Wait()`
 - `MPI_Ibarrier()`
 - `MPI_Ibcast()`
 - `MPI_Igather()`
 - `MPI_Iscatter()`
 - `MPI_Iallgather()`
 - `MPI_Ialltoall()`
 - `MPI_Ireduce()`
 - `MPI_Iallreduce()`
 - `MPI_Ireduce_scatter()`
 - `MPI_Iscan()`

Why MPI_Ibarrier?

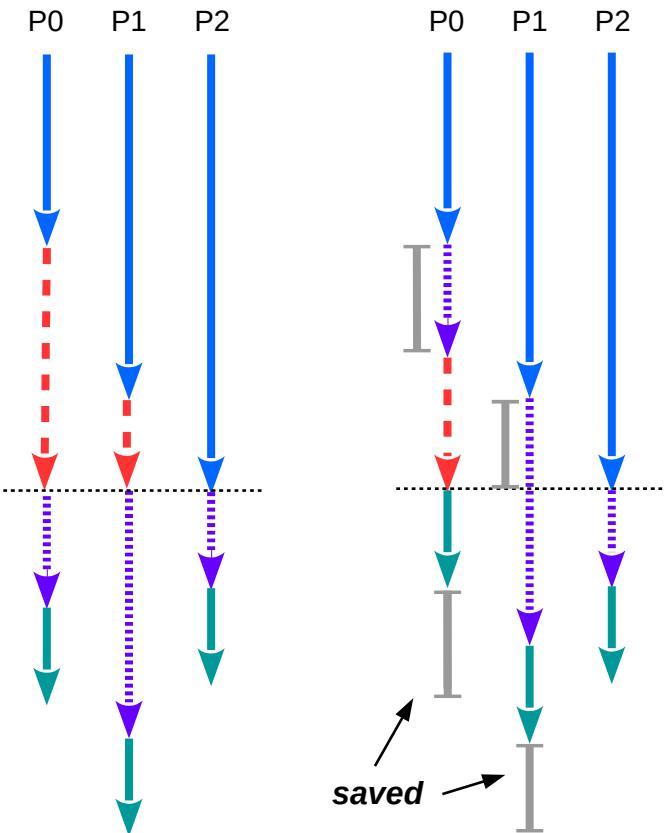
- Why would you want a *non-blocking* barrier?

```
work1();
MPI_Barrier(MPI_COMM_WORLD);
work2();          // independent
work3();          // dependent on work1()
```

Version 1

```
work1();
MPI_Request rq;
MPI_Ibarrier(MPI_COMM_WORLD, &rq);
work2();          // independent
MPI_Wait(&rq, MPI_STATUS_IGNORE);
work3();          // dependent on work1()
```

Version 2



Version 1

Version 2

Tools interface (MPI-3)

- MPI now provides a way to tweak parameters and access monitoring information in a cross-platform manner
- Control variables (cvar)
 - Startup options
 - Buffer sizes
- Performance variables (pvar)
 - Packets sent
 - Time spent blocking
 - Memory allocated

`MPI_T_cvar_get_info()`
`MPI_T_cvar_handle_alloc()`
`MPI_T_cvar_read()`
`MPI_T_cvar_write()`

`MPI_T_pvar_get_info()`
`MPI_T_pvar_session_create()`
`MPI_T_pvar_start() / stop()`
`MPI_T_pvar_handle_alloc()`
`MPI_T_pvar_read()`
`MPI_T_pvar_reset()`