**Intro to Distributed Memory**

**Parallel Computing with MPI**

NCSA Research Facilitation Services Workshop

**MPI Cheatsheet - C/C++**

## MPI Program Structure

| #include <mpi.h> . . int main() { . .  int mpierr;  .  .  mpierr = MPI\_Init(NULL, NULL);  .  .  .  mpierr = MPI\_Finalize()    return 0; } |
| --- |

## Information about PEs

| mpierr = MPI\_Comm\_size(MPI\_COMM\_WORLD, &nranks); |
| --- |

* Returns the total number of PEs in the World communicator into the integer variable **nranks**

| mpierr = MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myrank); |
| --- |

* Returns the ID of each PE into the variable **myrank**

## MPI Send message

| mpierr = MPI\_Send(void \*buf, int count, MPI\_Datatype dtype, int dst, int tag, MPI\_Comm comm); |
| --- |

* **buf**: initial memory address from where the message is being created
* **count**: number of data elements in the message
* **dtype:** data type of each element
* **dst**: ID of the destination PE
* **tag**: can be used to classify messages
* **comm**: the communicator over which the message is passed

## MPI Receive message

| mpierr = MPI\_Recv(void \*buf, int count, MPI\_Datatype dtype, int src, int tag, MPI\_Comm comm, MPI\_Status \*status); |
| --- |

* **buf**: initial memory address to where the message is going to be copied
* **count**: number of data elements in the message
* **dtype:** data type of each element
* **src**: ID of the source PE
* **tag**: can be used to classify messages
* **comm**: the communicator over which the message is passed
* **status:** stores information about the message that was received

## MPI Broadcast

| mpierr = MPI\_Bcast(void \*send\_buf, int send\_count, MPI\_Datatype dtype, int root\_id, MPI\_Comm comm); |
| --- |

* **send\_buf**: initial memory address of the message that is being sent to all PEs
* **send\_count:** number of elements sent
* **dtype**: data type of each element
* **root\_id**: ID of the PE from which message is being broadcast
* **comm**: communicator over which broadcasting will take place

## MPI Reduce

| mpierr = MPI\_Reduce(void \*send\_buf, void \*recv\_buf, int count, MPI\_Datatype dtype, MPI\_Op op, int root\_id, MPI\_Comm comm); |
| --- |

* **send\_buf**: initial memory address of the message that is going to be reduced
* **recv\_buf**: memory address to where the reduced result is going to be stored
* **count**: number of elements in each **send\_buf**
* **dtype**: data type of each element
* **op**: MPI reduce operation (e.g. MPI\_SUM)
* **root\_id**: ID of the PE that is going to receive the reduction
* **comm**: Communication over which reduction will happen