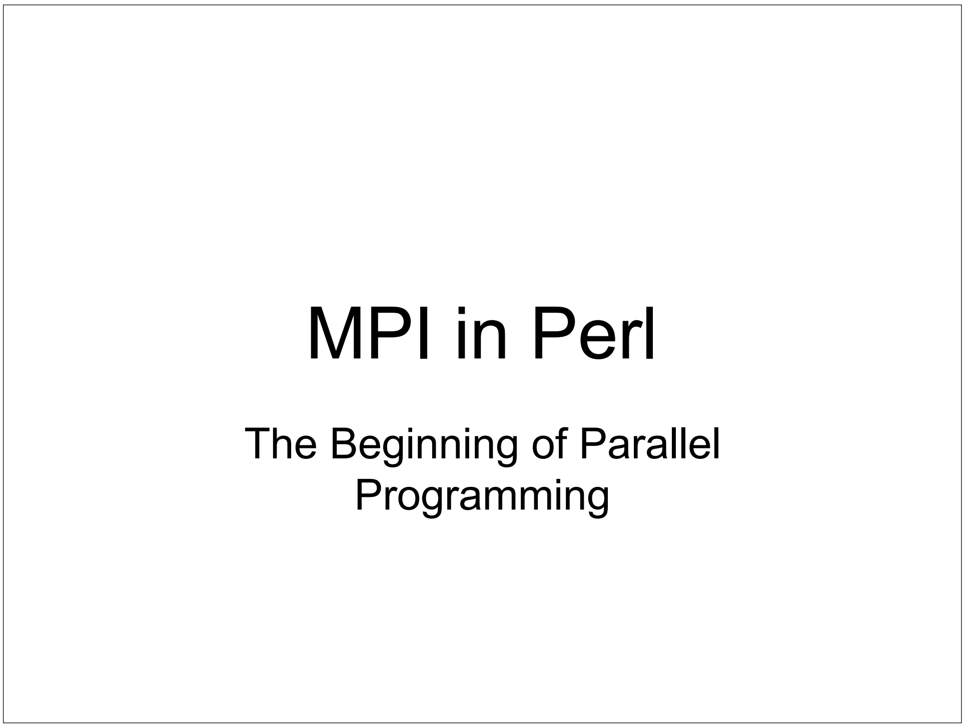
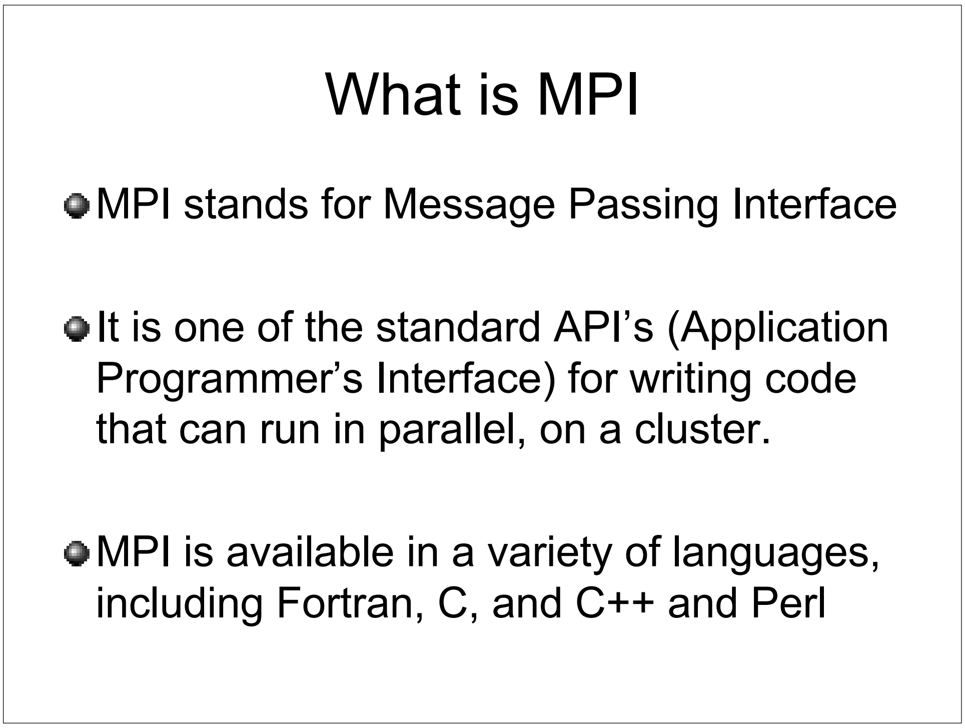


MPI in Perl

The Beginning of Parallel Programming



What is MPI

- MPI stands for Message Passing Interface
- It is one of the standard API's (Application Programmer's Interface) for writing code that can run in parallel, on a cluster.
- MPI is available in a variety of languages, including Fortran, C, and C++ and Perl

What About Perl?

- Perl, although not directly supported by MPI, can use an exported version of a C library.

- For our purposes, we will be using a port by Josh Wilmes and Chris Stevens. *Link*

Subroutine Walkthrough

- The following slides will contain a brief description and syntax usage for each of the functions.

Basic Functions Needed to Write MPI Programs

- MPI_Init = Initialize MPI
- MPI_Finalize = Finalize MPI
- MPI_Comm_size = # of Processors working
- MPI_Comm_rank = Identification Number
- MPI_Send = Send Message
- MPI_Recv = Receive Message

MPI Initalization

MPI_Init()

- Initializes the MPI execution environment. This function must be called before any MPI functions are used. It is called just once in the program.

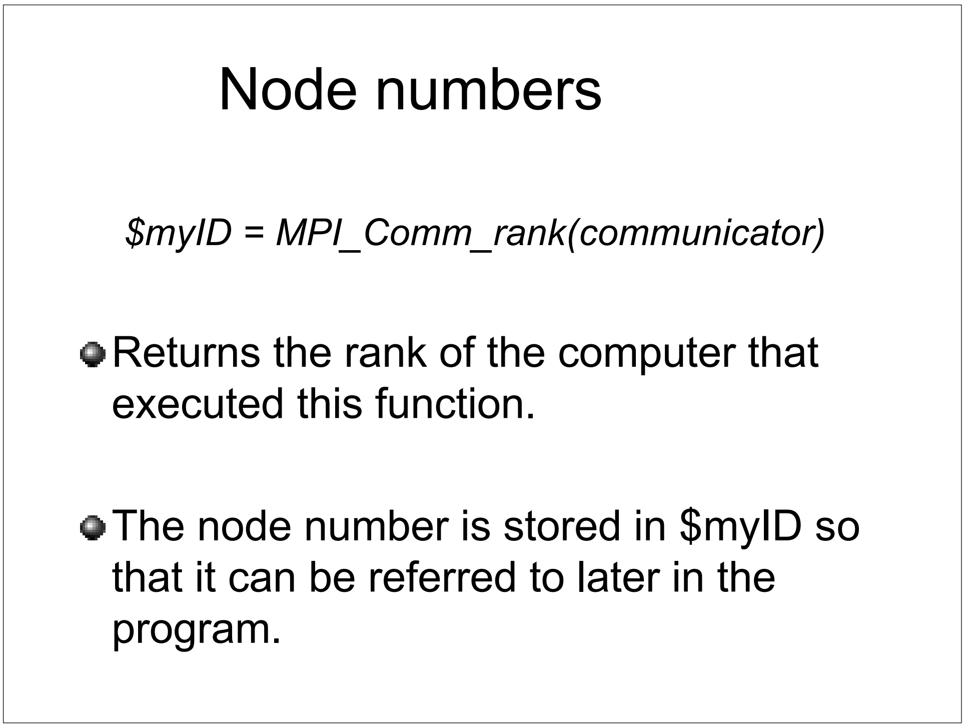
Communicator

- A MPI environment is defined by its communicator.
- The default communicator is MPI_COMM_WORLD
- The default communicator is aptly named because it describes the processing “world”. Processes can only communicate inside of this.

Cluster Size

$size = MPI_Comm_size(communicator)

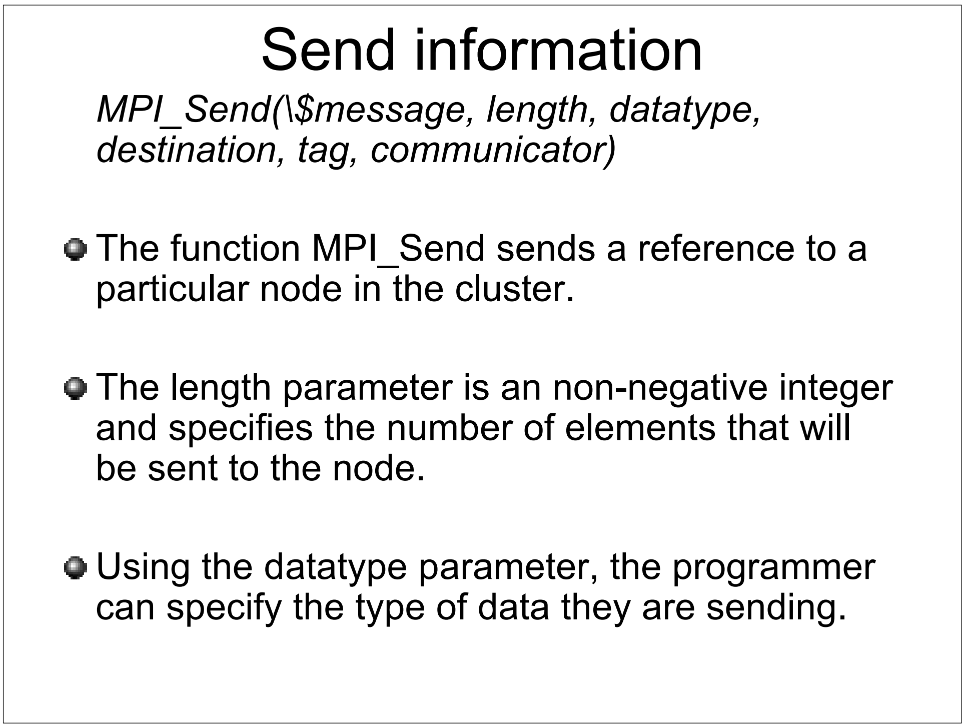
- Returns the number of computers in the cluster available to be used.
- $size stores this number so that it can be referred to later in a program.



Node numbers

$myID = MPI_Comm_rank(communicator)

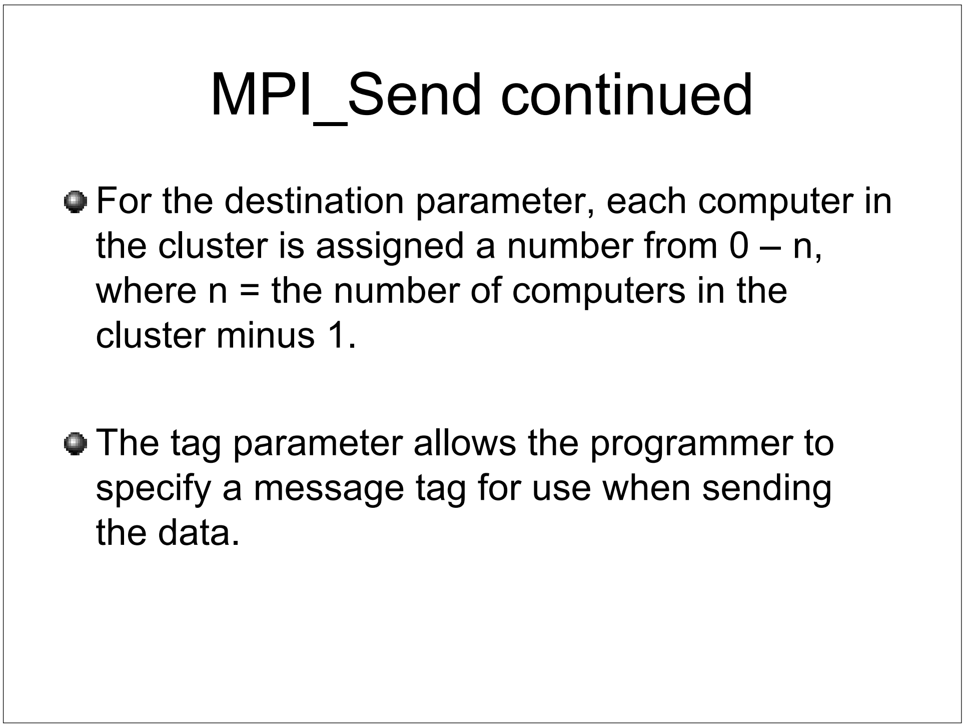
- Returns the rank of the computer that executed this function.
- The node number is stored in $myID so that it can be referred to later in the program.



Send information

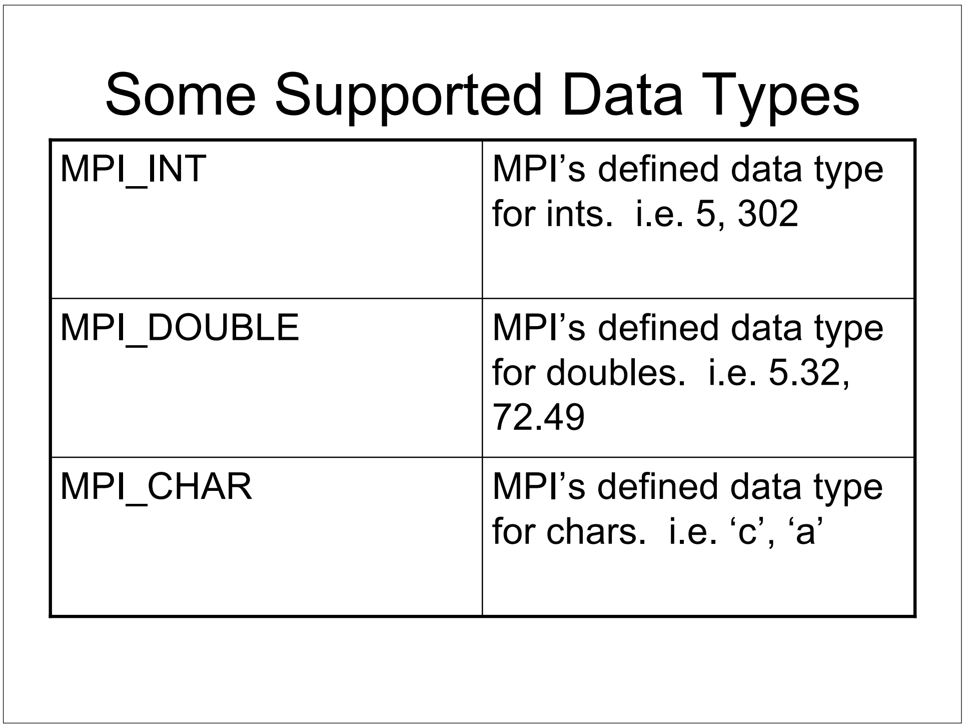
MPI_Send(\$message, length, datatype, destination, tag, communicator)

- The function MPI_Send sends a reference to a particular node in the cluster.
- The length parameter is an non-negative integer and specifies the number of elements that will be sent to the node.
- Using the datatype parameter, the programmer can specify the type of data they are sending.



MPI_Send continued

- For the destination parameter, each computer in the cluster is assigned a number from 0 – n, where n = the number of computers in the cluster minus 1.
- The tag parameter allows the programmer to specify a message tag for use when sending the data.



Some Supported Data Types

MPI_INT	MPI's defined data type for ints. i.e. 5, 302
MPI_DOUBLE	MPI's defined data type for doubles. i.e. 5.32, 72.49
MPI_CHAR	MPI's defined data type for chars. i.e. 'c', 'a'

Receiving Information

MPI_Recv(\$message, length, datatype, source, tag, communicator)

- The function MPI_Recv receives a particular reference from another node in the cluster.
- The length, datatype, tag and communicator parameters are the same as those for MPI_Send, i.e. datatype specifies the type of data that is being received.
- The source parameter is the number of the node that the computer is receiving the reference from, i.e. source = 2 if node 2 is sending information to the computer that receives it.

Checking to See if a Message is Waiting

MPI_Iprobe(source, tag, communicator, \%status)

- Status is a reference to a hash. If you want to see if any message is waiting for a node, it could use: MPI_Iprobe(MPI_ANY_SOURCE, MPI_ANY_TAG,MPI_COMM_WORLD, \%status)
- MPI_Iprobe returns a 1 if it found something, and a 0 if it had not. This means it could be used in an if statement.
- MPI_Iprobe only checks to see if there is a message, it does not receive that message. To clear the message from the receive buffer, an MPI_Recv must be called.

Status

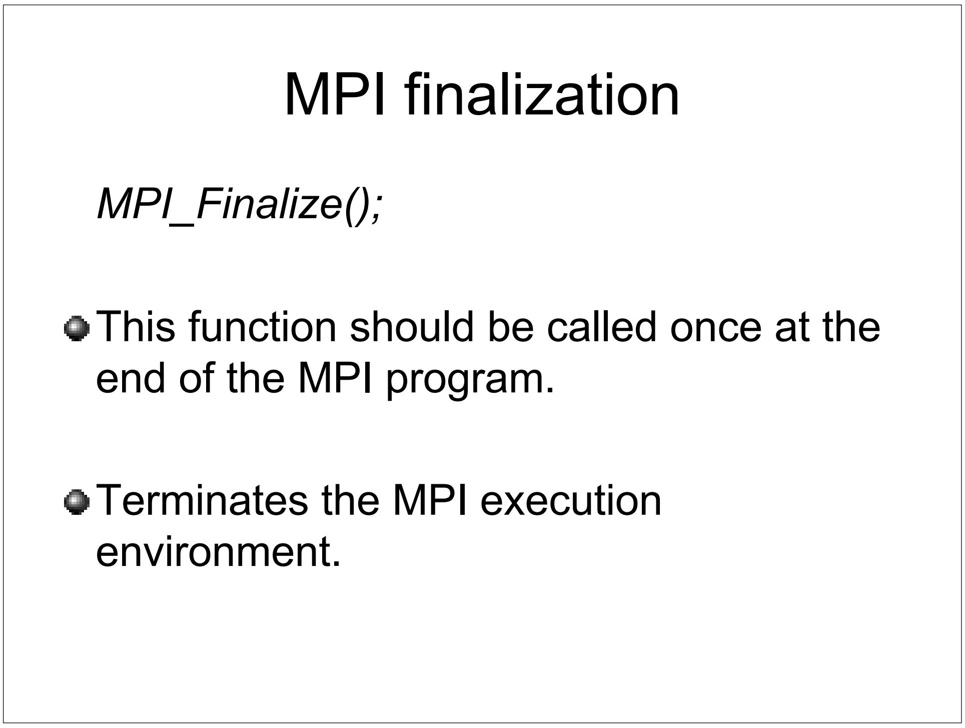
Status, as mentioned on the previous slide, is a has that contains several fields:

- MPI_TAG: The tag on which a message was received.
- MPI_ERROR: An error code, if any.
- count: The number of elements coming.
- MPI_SOURCE: The source of the message.

Broadcasting Messages

MPI_Bcast(\$from, count, datatype, root, communicator)

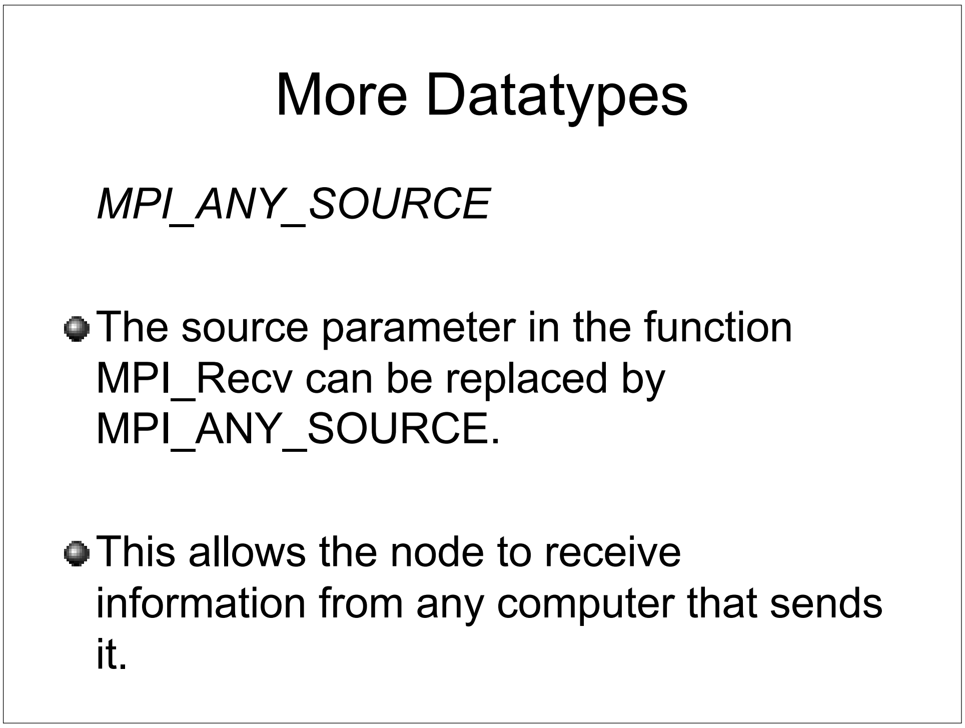
- This function call broadcasts a message to all nodes in the cluster.
- The count parameter specifies the number of data elements to be sent.
- The root parameter specifies the head node.



MPI finalization

MPI_Finalize();

- This function should be called once at the end of the MPI program.
- Terminates the MPI execution environment.



More Datatypes

MPI_ANY_SOURCE

- The source parameter in the function MPI_Recv can be replaced by MPI_ANY_SOURCE.
- This allows the node to receive information from any computer that sends it.

More Datatypes

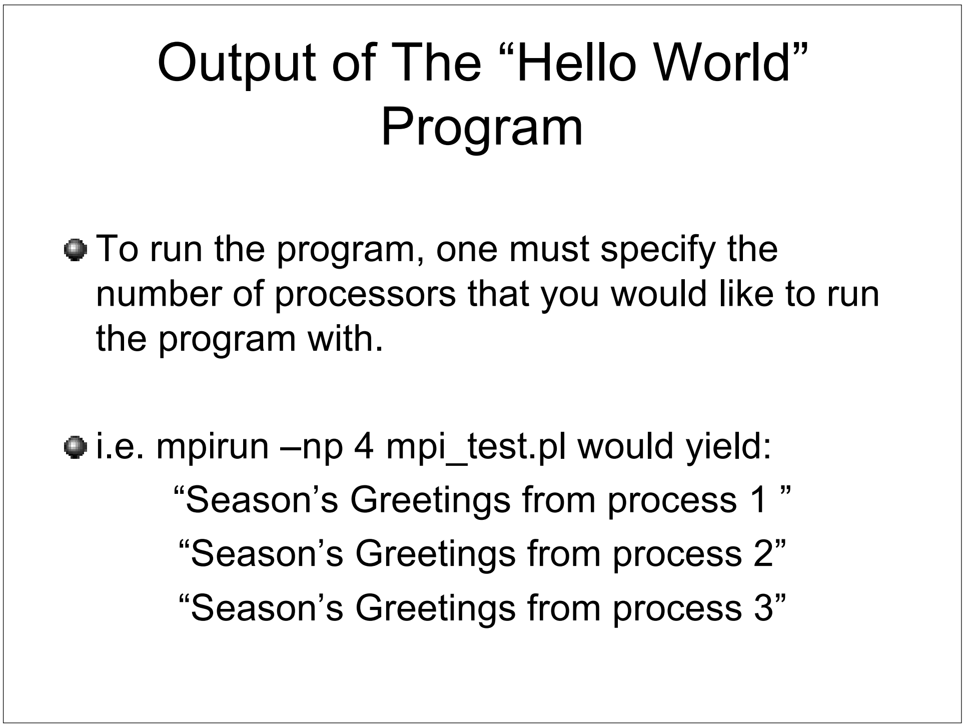
MPI_ANY_TAG

- Like MPI_ANY_SOURCE, this datatype can be placed in the tag parameter for MPI_Recv or MPI_Iprobe.
- This allows the node to receive information from another node that sends it using any tag.

Basic “Hello World” Program

- Perl:

```
#!/usr/bin/perl

use Parallel::MPI qw(:all);

MPI_Init();

my ($rank, $size);

$rank = MPI_Comm_rank(MPI_COMM_WORLD);
$size = MPI_Comm_size(MPI_COMM_WORLD);

$tag = 1137;

if( $rank != 0 ) {
    my $send = “Season’s Greetings from process $rank!";
    MPI_Send(\$send, length($send), MPI_CHAR, 0, $tag,
MPI_COMM_WORLD );
} else {
    my ($x, $recv);
    for( $x = 1; $x < $size; $x++ ) {
         MPI_Recv( \$recv, 35, MPI_CHAR, $x, $tag,
MPI_COMM_WORLD );
         print "Received $recv \n";
    }
}

MPI_Finalize();
```



Output of The “Hello World” Program

- To run the program, one must specify the number of processors that you would like to run the program with.
- i.e. mpirun –np 4 mpi_test.pl would yield:

```
“Season’s Greetings from process 1 ”
 “Season’s Greetings from process 2”
 “Season’s Greetings from process 3”
```