MPI Cheat Sheet

The majority of this information is taken with permission from Henry Neeman's excellent MPI slides. These are available online from the OU Supercomputing Center for Education and Research (http://www.oscer.ou.edu /sc08workshopou.php).

Setup and Tear Down

C/C++	Fortran
`MPI_Init(&argc, &argv);`	`CALL MPI_Init(mpi_error_code)`
`MPI_Finalize();`	`CALL MPI_Finalize(mpi_error_code)`

- `MPI_Init` starts up the MPI runtime environment at the beginning of a run.
- `MPI_Finalize` shuts down the MPI runtime environment at the end of a run.

Gathering Information

C/C++	Fortran
`MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);`	`CALL MPI_Comm_Rank(my_rank, mpi_error_code)`
`MPI_Comm_size (MPI_COMM_WORLD, &num_procs);`	`CALL MPI_Comm_size(num_procs, mpi_error_code)`
`MPI_Get_processor_name (&name, &result_length)`	`CALL MPI_Get_processor_name(name, & length, mpi_error_code)`

- `MPI_Comm_size` gets the number of processes in a run, Np (typically called just after MPI_Init).
- `MPI_Comm_rank` gets the process ID that the current process uses, which is between 0 and Np-1 inclusive (typically called just after MPI_Init).
- `MPI_Get_processor_name` is not often used in real code. We used it to prove to ourselves that "Hello, World!" was running on different machines. It returns the name of the machine that the code is running on.

Communication (Message Passing)

C/C++	Fortran
`MPI_Send (message, strlen(message)+1, MPI_CHAR, destination, tag, MPI_COMM_WORLD);`	`CALL MPI_Send(message, string_len(message), & MPI_CHARACTER, destination, tag & MPI_COMM_WORLD, mpi_error_code)`

```
MPI_Recv (message,
max_message_length+1,
     MPI_CHAR, source, tag,
     MPI_COMM_WORLD, &status)
```

```
 CALL MPI_Recv(message,
maximum_message_length,
&     MPI_CHARACTER, source, tag,
&     MPI_COMM_WORLD, status, mpi_error_code);
```

```
MPI_Bcast(array, length, MPI_INTEGER,
     source, MPI_COMM_WORLD);
```

```
 CALL MPI_Bcast(array, length, MPI_INTEGER,
&     source, MPI_COMM_WORLD, mpi_error_code)
```

```
MPI_Reduce(&value, &value_sum, count,
     MPI_INT, MPI_SUM, server,
     MPI_COMM_WORLD);
```

```
 CALL MPI_Reduce(value, value_sum, count,
&     MPI_INT, MPI_SUM, server,
&     MPI_COMM_WORLD, mpi_error_code)
```

```
MPI_Allreduce(&value, &value_sum,
count,
     MPI_INT, MPI_SUM, MPI_COMM_WORLD);
```

```
 CALL MPI_Allreduce(value, value_sum, count,
&     MPI_INT, MPI_SUM, MPI_COMM_WORLD,
&     mpi_error_code)
```

- `MPI_Send` sends a message from the current process to some other process (the ***destination***).
- `MPI_Recv` receives a message on the current process from some other process (the ***source***).
- `MPI_Bcast` broadcasts a message from one process to all of the others.
- `MPI_Reduce` performs a ***reduction*** (e.g., sum, maximum) of a variable on all processes, sending the result to a *single* process.
- `MPI_Allreduce` performs a reduction of a variable on all processes, and sends result to *all* processes (and therefore takes longer)

MPI Data Types

C/C++		Fortran	
char	**MPI_CHAR**	**CHARACTER**	**MPI_CHARACTER**
int	**MPI_INT**	**INTEGER**	**MPI_INTEGER**
float	**MPI_FLOAT**	**REAL**	**MPI_REAL**
double	**MPI_DOUBLE**	**DOUBLE PRECISION**	**MPI_DOUBLE_PRECISION**

Recommended Resources

- Peter Pacheco, Parallel Programming With MPI, 1st edition, Morgan Kaufmann, 1996.
- Documentation from the makers of MPI is available online at http://www-unix.mcs.anl.gov/mpi/www/