This talk will...

- Focus on the **basics** of parallel programming
  - Will not inundate you with the details of memory hierarchies, hardware architectures, network topology, etc.
  - Advanced topics may be the impetus for future workshops

- Use examples found on Oscar in:
  - /users/mhowison/MPI
  - (You can copy these folders to your home directory)

- Assume that you have some proficiency with:
  - Linux command line and a text editor
  - C programming language
MPI Basics

- A framework for *distributed-memory* parallelism:
  - Multiple *tasks* run concurrently across separate nodes
  - Each task has its own private memory (no shared variables!)
    - Memory is shared by *passing messages* among nodes
    - Messaging requires a high-performance interconnect
- MPI is implemented as a library with wrappers for compiling:
  - `mpicc -o mpi_program source.c`
- Make calls to the MPI library as you would with other APIs
  - An MPI program starts with `MPI_Init()` and ends with `MPI_Finalize()`
- Run an MPI program using the `mpirun` wrapper
  - `mpirun -n [number of tasks] ./mpi_program`

“hello” example
Communicators

- MPI tasks can be grouped into sets called “communicators”
- All available MPI tasks are automatically placed in the MPI_COMM_WORLD communicator
- Can synchronize communicators with barriers:
  - MPI_BARRIER(communicator)
- Advanced topic: constructing more complicated hierarchies, e.g. to mirror underlying hardware
Point-to-point

- Move data from one MPI task to another (similar to a TCP connection)
- In regular MPI (no fancy constructs), this is always *two-sided*
  - The sender has to call MPI_Send()
  - The receiver has to call MPI_Recv()
  - These calls are *blocking*: your program waits until the transaction is complete before continuing
- Advanced topics:
  - Non-blocking send/receive for asynchronous communication
  - One-sided messaging to decrease latency

“pingpong” example
Tags

- The send/receive functions accept a user-defined integer “tag”
- Can be used to screen messages, so that a receive only accepts certain classes of messages
- OR accept all messages with MPI_ANY_TAG
Data types

- For portability, you must tell MPI what kind of data you are transmitting
- Most basic types are predefined (e.g. MPI_DOUBLE, MPI_INT, MPI_CHAR, etc.)
- Advanced topic: “derived” data types
  - You can aggregate basic types into, for example, vectors or arrays
  - You can create custom types for structs
“Collective” calls

- Some common messaging paradigms are already implemented/optimized:
  - Broadcasting a message from one task to all tasks
    - MPI_Bcast
  - Computing (or “reducing”) a value across all tasks
    - MPI_Reduce with MPI_SUM, MPI_MIN, MPI_MAX, etc.
  - From one task, sending a unique message to every other task
    - MPI_Scatter
  - From each task, sending a unique message to one task
    - MPI_Gather
  - Sending a unique message from each task to every other task
    - MPI_Alltoall

“coin” example
**MPI_Bcast**

Broadcasts a message to all other processes of that group

```c
count = 1;
source = 1;
broadcast originates in task 1
MPI_Bcast(&msg, count, MPI_INT, source, MPI_COMM_WORLD);
```

<table>
<thead>
<tr>
<th>task 0</th>
<th>task 1</th>
<th>task 2</th>
<th>task 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

msg (before)

|        | 7      | 7      | 7      | 7      |

msg (after)

https://computing.llnl.gov/tutorials/mpi/#Collective_Communication_Routines
MPI_Scatter

Sends data from one task to all other tasks in a group

```c
sendcnt = 1;
recvcnt = 1;
src = 1;
task 1 contains the message to be scattered
MPI_Scatter(sendbuf, sendcnt, MPI_INT,
            recvbuf, recvcnt, MPI_INT,
            src, MPI_COMM_WORLD);
```

![Diagram showing MPI_Scatter](https://computing.llnl.gov/tutorials/mpi/#Collective_Communication_Routines)
MPI_Gather

Gathers together values from a group of processes

sendcnt = 1;
rcvcnt = 1;
src = 1;
messages will be gathered in task 1
MPI_Gather(sendbuf, sendcnt, MPI_INT,
recvbuf, rcvcnt, MPI_INT,
src, MPI_COMM_WORLD);

task 0  task 1  task 2  task 3

1 2 3 4

sendbuf (before)

1
2
3
4
recvbuf (after)

https://computing.llnl.gov/tutorials/mpi/#Collective_Communication_Routines
MPI_Allgather

Gathers together values from a group of processes and distributes to all

```
sendcnt = 1;
recvcnt = 1;
MPI_Allgather(sendbuf, sendcnt, MPI_INT,
recvbuf, recvcnt, MPI_INT,
MPI_COMM_WORLD);
```

<table>
<thead>
<tr>
<th>task 0</th>
<th>task 1</th>
<th>task 2</th>
<th>task 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- sendbuf (before)
- recvbuf (after)

https://computing.llnl.gov/tutorials/mpi/#Collective_Communication_Routines
MPI_Reduce

Perform and associate reduction operation across all tasks in the group and place the result in one task

```c
count = 1;
dest = 1;
result will be placed in task 1
MPI_Reduce(sendbuf, recvbuf, count, MPI_INT, MPI_SUM, 
dest, MPI_COMM_WORLD);
```

task 0  task 1  task 2  task 3

1  2  3  4  → sendbuf (before)

10  10  10  10  → recvbuf (after)

https://computing.llnl.gov/tutorials/mpi/#Collective_Communication_Routines
MPI_Allreduce

Perform and associate reduction operation across all tasks in the group and place the result in all tasks

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

<table>
<thead>
<tr>
<th>task 0</th>
<th>task 1</th>
<th>task 2</th>
<th>task 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

sendbuf (before)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

recvbuf (after)

https://computing.llnl.gov/tutorials/mpi/#Collective_Communication_Routines
MPI_Alltoall

Sends data from all to all processes. Each process performs a scatter operation.

```
sendcnt = 1;
recvcnt = 1;
MPI_Alltoall(sendbuf, sendcnt, MPI_INT,
recvbuf, recvcnt, MPI_INT,
MPI_COMM_WORLD);
```

```
<table>
<thead>
<tr>
<th>task 0</th>
<th>task 1</th>
<th>task 2</th>
<th>task 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
</tr>
</tbody>
</table>
```

sendbuf (be)

```
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

recvbuf (e)

https://computing.llnl.gov/tutorials/mpi/#Collective_Communication_Routines
MPI_Scan

Computes the scan (partial reductions) of data on a collection of processes

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

<table>
<thead>
<tr>
<th>task 0</th>
<th>task 1</th>
<th>task 2</th>
<th>task 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

sendbuf (before)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

recvbuf (after)

https://computing.llnl.gov/tutorials/mpi/#Collective_Communication_Routines
mpi4py

- http://mpi4py.scipy.org/
- Adds MPI support to Python scripts
- Multiple python interpreters are launched:
  mpirun -n 16 python MyMPIscript.py
- Can access the MPI interface from within python:
  ```python
  from mpi4py import MPI
  comm = MPI.COMM_WORLD
  size = comm.Get_size()
  rank = comm.Get_rank()
  ```
Exercises

- Write a program that prints out a message from each rank *in order*
  - Using a barrier
  - OR using point-to-point messages to send a “token” message through the ranks
- Choose a collective operation and implement it yourself with point-to-point messaging
Additional resources

- MPI Specification: http://www.mpi-forum.org/docs/docs.html
- NERSC Tutorial (Fortran and C/C++): http://www.nersc.gov/nusers/help/tutorials/mpi/intro/
- LLNL Tutorial (Fortran and C/C++): https://computing.llnl.gov/tutorials/mpi/