- MPI tasks
- Simple Example
- Collective communications
- Point-to-Point Communications
The same code is run on many processors

Each MPI task can use many OpenMP threads. So, a task is not necessarily mapped on a core or a processor, but often it does.

After initialization each MPI task gets its unique id (rank)

All MPI tasks are equal. For programming purposes it is convenient to name the rank=0 task as root and use it differently.

Exchange of data between tasks is done by library calls.

Semantics for Fortran and C are very similar.

Root (and some other tasks) can be allocated to different compute nodes with larger memory.

Submit a PBS script to a queue. The script gives details of your job and has a line `mpiexec -np NNN mycode.exe` (or mpirun) where -np NNN specifies the number of MPI tasks.
#include "mpi.h"
#include <stdio.h>

int main(argc,argv)
    int argc;
    char *argv[]; 
    int numtasks, rank, rc;

rc = MPI_Init(&argc,&argv);
if (rc != MPI_SUCCESS) {
    printf("Error starting MPI program. Terminating.\n");
    MPI_Abort(MPI_COMM_WORLD, rc);
}

MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
printf("Number of tasks= %d My rank= %d\n", numtasks,rank);

/**************** do some work ********/

MPI_Finalize();
	
program simple
    include 'mpif.h'

    integer numtasks, rank, ierr, rc

    call MPI_INIT(ierr)
    if (ierr .ne. MPI_SUCCESS) then
        print *, 'Error starting MPI program. Terminating,'
        call MPI_ABORT(MPI_COMM_WORLD, rc, ierr)
    end if

    call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, numtasks, ierr)
    print *, 'Number of tasks=',numtasks,' My rank=',rank

C ***** do some work *****

call MPI_FINALIZE(ierr)
end
Collective Communications

- Collective communication must involve all processes
- Types of communications:
  - Synchronization
  - Data transfer: broadcast, scatter gather, all-to-all
  - Collective computation: (reductions) - one member of the group collects data from the other members and performs an operation (min, max, add, multiply, etc.) on that data.

One task reads input from screen and distributes it to all others

https://computing.llnl.gov/tutorials/mpi/#Getting_Started
Simple Example: matrix transposition

Three-dimensional matrix $A(Nrow,Nrow,Nrow)$ is split such that each task $k$ has its one page $G(:,:) = A(:, :, k)$

After transposition $Gb(:, :) = A(k ; :, :)$
MPI_AlltoALL

https://computing.llnl.gov/tutorials/mpi/#Getting_Started

sendbuf = starting address of send buffer
recvbuf = address of receive buffer
sebdcnt = number of elements send to each process
recvcnt = number of elements received

j-th block from process i is received by process j and placed in the i-th block of recvbuf

<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 (before)

 recvbuf (after)
MPI_Scatter: root distributes data

```c
MPI_Comm comm;
    int gsize,*sendbuf;
    int root, rbuf[100];
...
MPI_Comm_size( comm, &gsize);
sendbuf = (int *)malloc(gsize*100*sizeof(int));
...
MPI_Scatter( sendbuf, 100, MPI_INT, rbuf, 100, MPI_INT, root, comm);
```

![Diagram](http://www.mpi-forum.org/docs/mpi-11-html/node72.html)
Root gathers data from other tasks

```c
MPI_Comm_size( comm, &gsize);
rbuf = (int *)malloc(gsize*stride*sizeof(int));
displs = (int *)malloc(gsize*sizeof(int));
rcounts = (int *)malloc(gsize*sizeof(int));
for (i=0; i<gsize; ++i) {
    displs[i] = i*stride;
    rcounts[i] = 100;
}
MPI_Gatherv( sendarray, 100, MPI_INT, rbuf, rcounts, displs, MPI_INT,
            root, comm);
```

Note that the program is erroneous if \textit{stride} < 100.

\begin{itemize}
  \item \textit{all processes}
  \item \textit{at root}
\end{itemize}
Point-to-Point communications

MPI_SEND: send message to task \texttt{dest}

MPI_RECV: receive message from task \texttt{dest}

C synopsis

```c
#include <mpi.h>
int MPI_Send(void* buf, int count, MPI_Datatype datatype, 
             int dest, int tag, MPI_Comm comm);
```

C++ synopsis

```cpp
#include mpi.h
void MPI::Comm::Send(const void* buf, int count, const MPI::Datatype& datatype, 
                     int dest, int tag) const;
```

FORTRAN synopsis

```fortran
#include 'mpif.h' or use mpi
MPI_SEND(CHOICE BUF,INTEGER COUNT,INTEGER DATATYPE,INTEGER DEST, 
          INTEGER TAG,INTEGER COMM,INTEGER IERROR)
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>The initial address of the send buffer (choice) (IN)</td>
</tr>
<tr>
<td>count</td>
<td>The number of elements in the send buffer (non-negative integer) (IN)</td>
</tr>
<tr>
<td>datatype</td>
<td>The data type of each send buffer element (handle) (IN)</td>
</tr>
<tr>
<td>dest</td>
<td>The data type of each send buffer element (handle) (IN)</td>
</tr>
<tr>
<td>tag</td>
<td>The rank of the destination task in \textit{comm}(integer) (IN)</td>
</tr>
<tr>
<td>comm</td>
<td>The communicator (handle) (IN)</td>
</tr>
<tr>
<td>IERROR</td>
<td>The FORTRAN return code. It is always the last argument.</td>
</tr>
</tbody>
</table>
Example of Point-to-Point communications

```c
if (my_rank == 0) {
    fputs(greeting, stdout);
    for (partner = 1; partner < size; partner++) {
        MPI_Recv(greeting, sizeof(greeting), MPI_BYTE, partner, 1, MPI_COMM_WORLD, &stat);
        fputs(greeting, stdout);
    }
} else {
    MPI_Send(greeting, strlen(greeting)+1, MPI_BYTE, 0, 1, MPI_COMM_WORLD);
}
```