Intermediate MPI

Mobeen Ludin

Overview

- Point-to-Point Communication
  - In point to point communication, one process sends a message and one process receives it.

Overview (Cont.)

```
if ( I am processor A ) then
   add a bunch of numbers
else if ( I am processor B ) then
   multiply a matrix times a vector
end
```

If I have my own brain, memory, and functional units like (eyes, hands, legs, muscles, etc..) to get work done on my own, then I could be doing work independently of someone else.

Overview (Cont.)

```
if ( I am processor A ) then
  call MPI_Send ( X )
else if ( I am processor B ) then
  call MPI_Recv ( X )
end
```

- Data stored on one computer is completely different from another
- Cant read your mind: if you need something you ask for it explicitly

Collective Communication

- MPI collective operations allow all ranks (processes) in a given communication context (communicator) to talk to each other at the same time.
- All ranks in the communicator must make the same MPI call for the operation to succeed.
- Collective operations are:
  - Provided for convenience
  - Tuned for system performance

MPI Collective Operation: Broadcast

```
MPI_Bcast(void* buffer, int count,
  MPI_Datatype datatype, int root,
  MPI_Comm comm);
```

Often when a process needs to communicate to all other process in the communicator (MPI_COMM_WORLD).

Example:

```
MPI_Bcast(&num_sub_intervals, 1,
  MPI_INT, 0, MPI_COMM_WORLD);
```

MPI Collective Operation: Reduce

```
MPI_Reduce(void* sbuf, void* rbuf, int
  count, MPI_Datatype stype, MPI_Op op,
  int root, MPI_Comm comm);
```

- Global reduction or combine operation
- The partial result in each process in the group is combined in one specified process.

Example:

```
MPI_Reduce(&pi, &pi_val, 1, MPI_DOUBLE,
  MPI_SUM, 0, MPI_COMM_WORLD);
```

Monte Carlo PI estimation

The Exercise is to use MPI collective operations in a program that estimates pi.

In this method, the program generates **N** random points in the unit square.

Count how many points are in the quarter circle (**C**). Then PI is approximately equal to the ratio **(4 * C) / N.**

It's important that each processor use DIFFERENT random numbers. One way to ensure this is to have a single master processor generate all the random numbers, and then divide them up.

More on Algorithm:

http://en.wikipedia.org/wiki/Monte_Carlo_integration

Exercise:

In your MPI directory that you copied is a serial code (**pi_MonteCarlo.c**). Use the MPI collective communication operations to parallelize this code.