Point-to-Point Methods

- Used to distribute large data across processes
- Allows nodes to send data directly between each other
  - “Send” and “Receive” will be two operative ideas
- Luckily, all communication managed by the communicator!
  - We just need to know rank destinations

Local vs global data

- Keep in mind that each processor will have to index from 0, while that index may be in a broader context!

```
int myfirst = ...; // calculate
for(int ilocal=0; ilocal<nlocal;
ilocal++) {
  int iglobal = myfirst+ilocal;
  array[ilocal] = f(iglobal);
}
```

Example: Fourier Transform

- (More on this, mathematically, later)
- Used to transform a “signal” or “wave” from time domain to frequency domain
- If f is a function on interval [0,1] then the Fourier coefficients are given by

$$f_n := \int_0^1 f(t) e^{-t/\pi} dt$$

- So we approximate by:

$$f_n = \sum_{i=1}^{N-1} f(ih) e^{-in/\pi}$$

More Blocking Operations

- Recall that collective operations are blocking
  - Consider reduce, gather, scatter, bcast
  - Slowdown in prime detection lab?
- We sometimes need blocking to be intentional
- Example: three-point averaging
  - For vector x, compute y so that

$$y_i = (x_{i-1} + x_i + x_{i+1})/3$$

Three-point averaging

- Consider the problem when vector x is disjointly distributed
- The first index computed by every processor will be an issue!
  - Needs an index “owned” by another active processor
  - Same happens with the last index

Sending & Receiving

- “ping-pong”: A sends message to B, which receives it and sends a reply:

```
// On A
MPI_Send( /* to: */ B ..... );
MPI_Recv( /* from: */ B ... );
// On B
MPI_Recv( /* from: */ A ... );
MPI_Send( /* to: */ A ..... );
```

MPI_Send Semantics

- int MPI_Send(

  const void* buf, int count, MPI_Datatype datatype,

  int dest, int tag, MPI_Comm comm)
- Notice similarity with rooted collectives
- This maybe non-blocking!!
  - (For small messages)
  - Use MPI_Ssend to block

Receive Semantics

- `int MPI_Recv(`

  ```
  void* buf, int count, MPI_Datatype datatype,
  int source, int tag, MPI_Comm comm, MPI_Status *status)
  ```

- Status will encode other metadata of the data
  - Receiver can use MPI_STATUS_IGNORE a lot of the time

Example: Ping-Pong

- Create two clients that send each other a message back and forth
- Use a counter, if same parity as rank, increment and send back to partner

Simulating Ring Topology

- Create a “token” and pass it between all processes in order of rank
- Keep in mind the “last” process has to send to rank 0
- Add prints at each stage, notice the blocking behavior!

Probing & Dynamic Recv

- Finally use the MPI_Status object!
- It contains:
  - The rank of the sender, MPI_SOURCE field
  - The tag of the message, MPI_TAG field
- We can pass the status to MPI_Get_count to determine the length of the message
  - MPI_Get_count(MPI_Status*, MPI_Datatype, int*)
  - Ordinarily, Recv size specifies a maximum, but may get less

Why??

- In MPI_Recv, we can pass MPI_ANY_TAG and MPI_ANY_SOURCE, to accept any values in that field
  - Remember that otherwise, these fields have to match what we receive, all others will be queued in a message buffer!
- So use the status to store information on what the source and tag is
  - Tags commonly used to differentiate message types, specific for the application

Tag Example

- Use c enums to specify the “purpose” of some data
- enum my_tag_t {
    day_tag, month_tag, year_tag

  }
- Now we can add semantics to MPI_Send!
- MPI_Send(&var, 1, MPI_Int, dest, day_tag, world)

Probing

- MPI_Probe(int source, int tag, MPI_Comm comm, MPI_Status* status)
  - Gets the status before actually loading the message out of the incoming buffer
  - Regular blocking behavior
  - This way, allocation for the receiving data buffer can be done more efficiently

Pairwise Exchange

- We can send and receive data at the same time with MPI_Sendrcv(

  const void *sendbuf, int sendcount,
  MPI_Datatype sendtype, int dest,
  int sendtag, void *recvbuf, int recvcount,
  MPI_Datatype recvtype, int source,
  int recvtag, MPI_Comm comm, MPI_Status *status

)
- Example: “every node send data to the right”

Partial Operations

- “Scanning”
  - Like a reduce, but leaves the first i elements combined, on processor i
  - MPI_Scan
    - Indices are inclusive
  - MPI_Exscan
    - Indices are exclusive
- “Noop” destination, MPI_PROC_NULL can be used to send nowhere, ignoring the send completely

Sorting Algorithm Example

- The “Odd-Even” sort on an array of n elements works as follows:
  - Distribute data across a linear array
  - Repeat n times:
    - Even processors do “compare-and-swap” with right neighbor
    - Odd processors do “compare-and-swap” with right neighbor
- Compare-and-swap puts the larger of two elements to the right of the smaller one
  - One can use MPI_MIN and the other MPI_MAX!

In-place Sendrcv “Swap”

- If send and recv buffer have the same type and size, we can us MPI_Sendrcv_replace to use just one buffer to send and receive the data
- int MPI_Sendrecv_replace(

  void *buf, int count, MPI_Datatype datatype, int dest, int sendtag, int source, int recvtag, MPI_Comm comm, MPI_Status *status)

Exercise

- Adapt the “odd-even” sort algorithm to situation where each processor stores more than one single element
- Consider the following diagram for inspiration: