LA-UR-16-24311
Approved for public release; distribution is unlimited.

Title: Distributed Computing (MPI)

Author(s): Garrett, Charles Kristopher

Intended for: Parallel Summer Computing Research Internship lecture

Issued: 2016-06-17

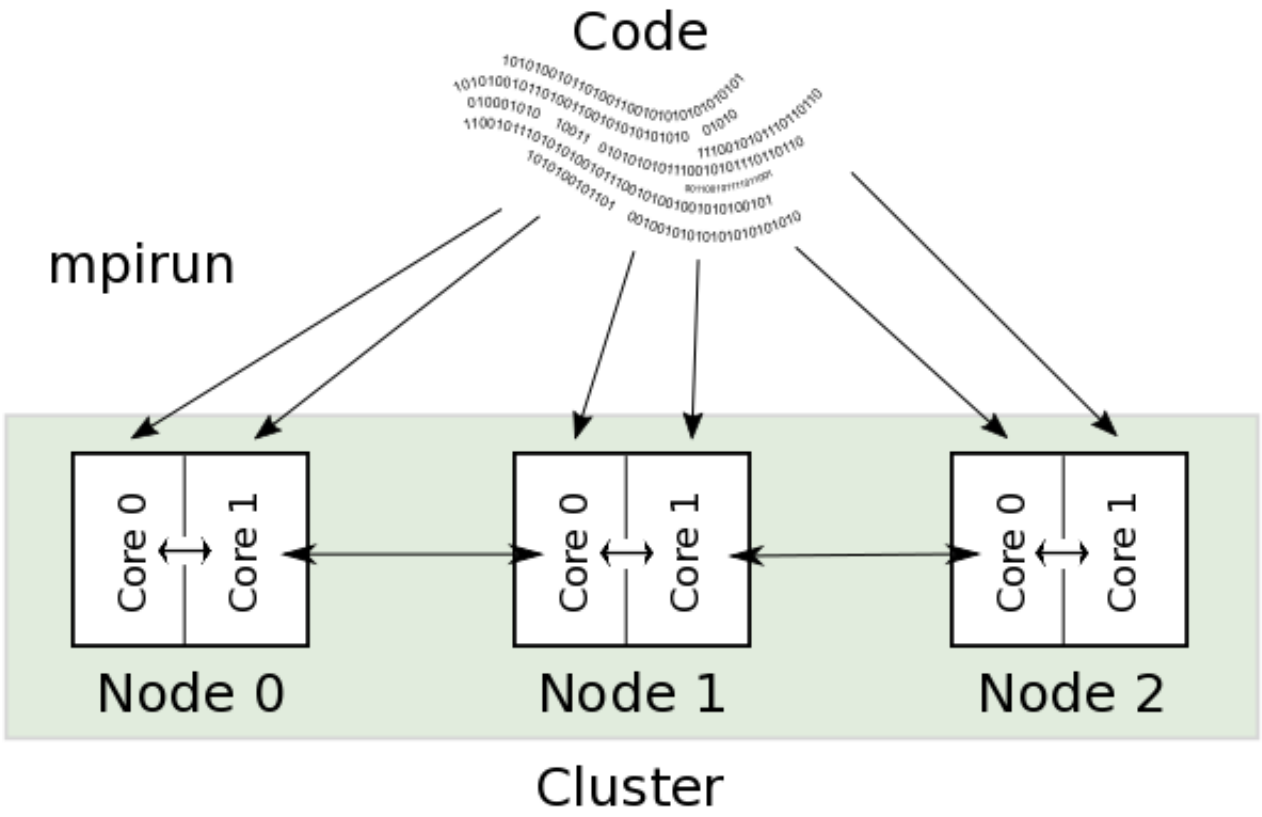
Distributed Computing (MPI)

Kris Garrett

June 2016

Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

How MPI Works



How MPI Works

- **MPI = Message Passing Interface**
- **Executable is run in multiple processes**
- **Each process communicates with each other**
  - Processes may be on the same computer
  - Processes may be on multiple nodes of a cluster
  - Multiple processes may be placed on a node to utilize multi-core processors
- **C and Fortran library APIs given by the standard**
- **Other 3rd party bindings exist (Python, C++, etc)**
- **Will concentrate on C library bindings here**

First Program

```
int main(int argc, char **argv)
{
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    printf("%d of %d\n", rank, size);
    MPI_Finalize();
    return 0;
}
```

Output
1 of 4
3 of 4
0 of 4
2 of 4

First Program

- **Wrapper is used to compile MPI application (OpenMPI)**
  - mpic++ main.cpp
- **Wrapper is used to run application**
  - mpirun -n 4 ./a.out
  - For this example, 4 copies of a.out are run
  - Each copy has an associated index called a rank
- **MPI uses the concept of a communicator**
  - Default MPI_COMM_WORLD for all MPI ranks
  - Can create subsets of ranks
    - Useful for libraries

Point to Point Communication

- **MPI_Send and MPI_Recv used for communication between 2 ranks**
- **Parameters include**
  - Data to send (MPI_Send)
  - Buffer to copy received data (MPI_Recv)
  - Rank to send to or receive from
  - Communicator and integer tag
    - Both must match in a send/recv

MPI_Send

```
int MPI_Send(const void *buf, int count,
             MPI_Datatype datatype, int dest,
             int tag, MPI_Comm comm)
```

- **buf – Buffer of data to send**
- **count – Number of items to send**
- **datatype – Built-in (MPI_INT, MPI_BYTE, MPI_DOUBLE, ...) or your own**
- **dest – Destination MPI rank**
- **tag – Identifier for the data**
- **comm – MPI communicator (MPI_COMM_WORLD or your own)**

Example

Send array of double values to rank+1

```
double sendDoubles[2];
double recvDoubles[2];

if (rank < RANK_MAX) {
    MPI_Send(sendDoubles, 2, MPI_DOUBLE, rank+1, 0,
             MPI_COMM_WORLD);
}
if (rank > 0) {
    MPI_Recv(recvDoubles, 2, MPI_DOUBLE, rank-1, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```

Example

- **MPI_Send is hard to predict**
  - After return from MPI_Send, you can reuse the data buffer
  - But you don't know if the data has been sent when MPI_Send returns
    - Small messages: MPI returns before data is received by destination
    - Large messages: MPI returns after data is received by destination
- **Can use MPI_Ssend to ensure message has been received by destination when function returns**
- **But, if MPI_Ssend works, MPI_Send should work and MPI_Send could yield higher performance**

Example

- **If MPI_Send does not return until data is received**
  - All ranks except N-1 begin a send
  - Only rank N-1 gets to MPI_Recv statement
  - Rank N-1 receives data and rank N-2 finishes send
  - Rank N-2 receives data and rank N-3 finishes send
  - This continues ***sequentially!!!***
- **This code may not parallelize**

Example

- **Several solutions exist**
- **One solution is to use Isend/Irecv**
  - These use nonblocking calls
  - Data buffer cannot be used after Isend/Irecv return
    - *The data in the buffer isn't used yet*
  - Use MPI_Wait, Mpi_Waitall, MPI_Waitany to know when Isend/Irecv is done
    - Must have an MPI_Wait for each Isend/Irecv.
    - *Not having one creates a memory leak*
- **Best practice**
  - Post MPI_Irecv before MPI_Isend

Isend/Irecv

```
int MPI_Isend(const void *buf, int count,
              MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm,
              MPI_Request *request)

int MPI_Irecv(void *buf, int count,
              MPI_Datatype datatype, int source,
              int tag, MPI_Comm comm,
              MPI_Request *request)
```

- **request – Used by MPI_Wait**

Example

Send array of double values to rank+1

```
double sendDoubles[2];
double recvDoubles[2];
MPI_Request requests[2] = {MPI_REQUEST_NULL,
                           MPI_REQUEST_NULL};
if (rank < RANK_MAX) {
    MPI_Isend(sendDoubles, 2, MPI_DOUBLE, rank+1, 0,
              MPI_COMM_WORLD, &requests[0]);
}
if (rank > 0) {
    MPI_Irecv(recvDoubles, 2, MPI_DOUBLE, rank-1, 0,
              MPI_COMM_WORLD, &requests[1]);
}
MPI_Waitall(2, requests, MPI_STATUSES_IGNORE);
```

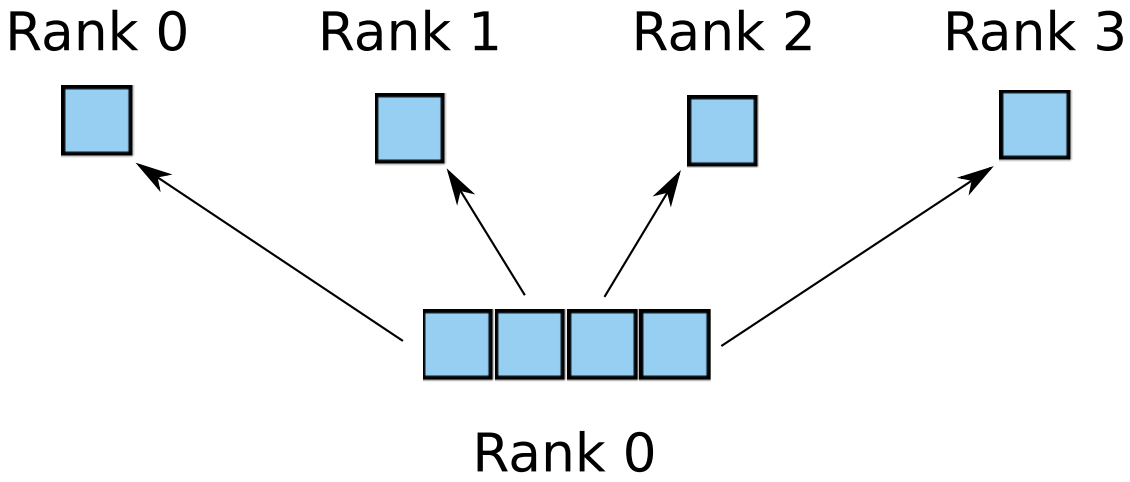
Broadcast and Collective Routines

- **These functions include all ranks in a communicator**
- **Simplest is MPI_Barrier**
  - Make all ranks wait until they hit the barrier
  - Be careful not to put this in a branching statement (like an if statement)
  - All ranks in communicator must call this before code moves forward
  - Can be useful for debugging

MPI_Scatter

Elements in array go to different ranks

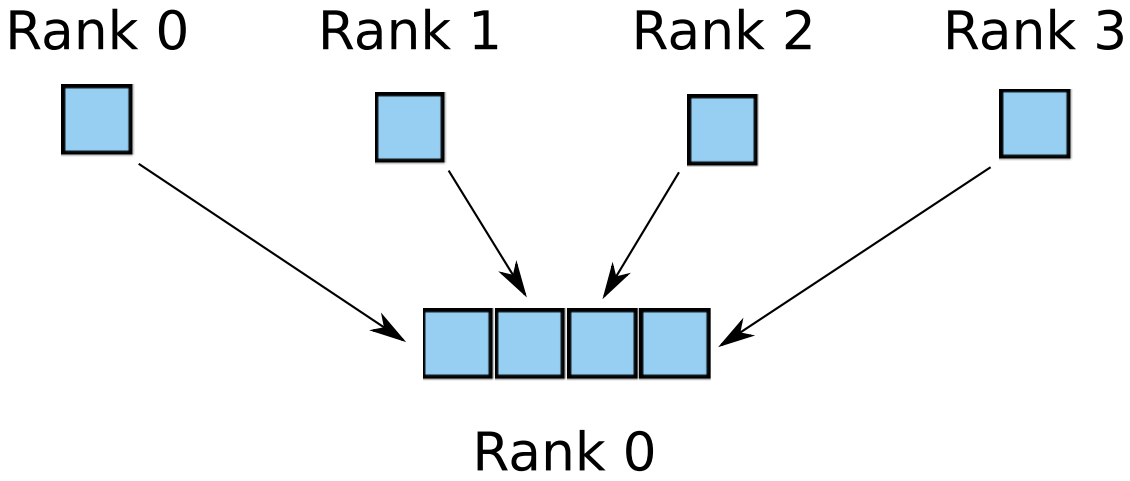
(Bcast: 1 element sent to all ranks)



MPI_Gather

Element from each rank goes into one array

(Allgather: every rank gets the whole array)



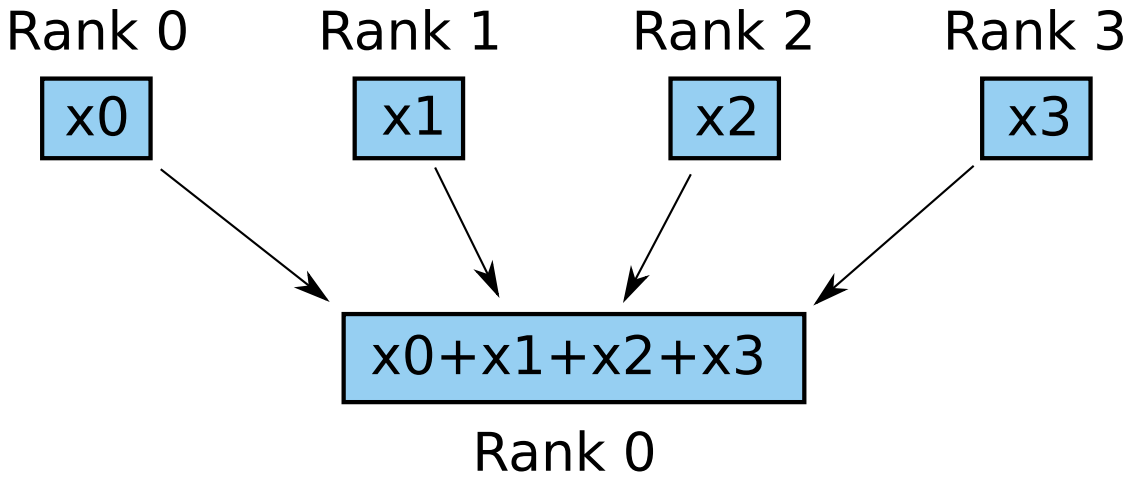
MPI_Reduce

Binary operation of element sent to one rank

Built-in operations: max, min, sum, product, …

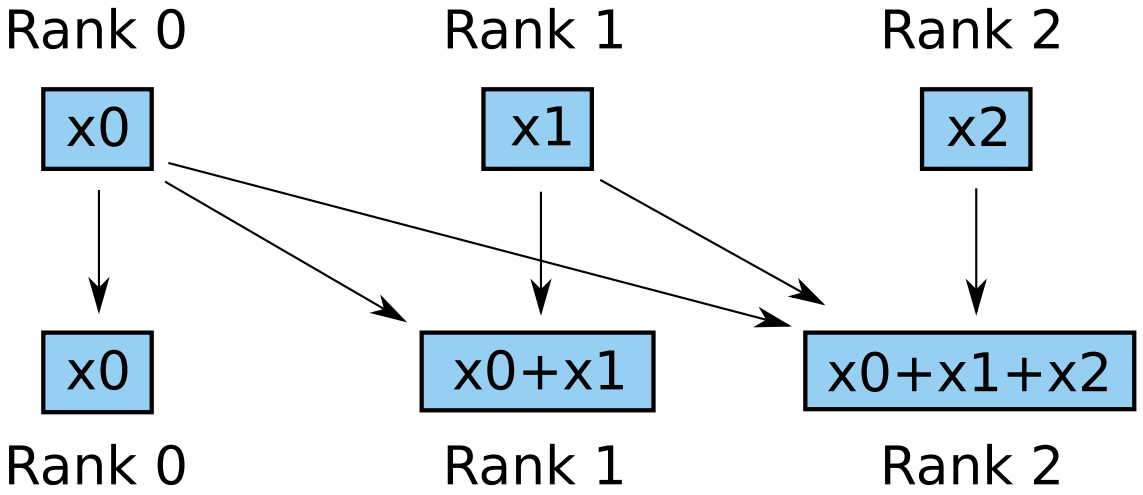
Can define your own binary operation

(Allreduce: All ranks get answer)



MPI_Scan

Binary operation on elements from each rank



MPI_Init_thread

- **Use for threading with MPI (such as OpenMP, pthreads, ...)**
- **Four threading types**
  - MPI_THREAD_SINGLE – No threading
  - MPI_THREAD_FUNNELED – All MPI calls made by master thread
  - MPI_THREAD_SERIALIZED – Only one thread makes an MPI call at a time, but calls can come from different threads
  - MPI_THREAD_MULTIPLE – Different threads may call MPI routines at the same time
- **Best practice: create a communicator for each thread**

Error Handling

- In C: MPI functions return an error status
- In Fortran: MPI functions have an extra argument, ierr
- Should return MPI_Success every time

```
int mpiError = MPI_Send(...);
```

MPI 3 and One Sided Communication

- **Original MPI requires all ranks involved in a communication to call a function**
- **MPI 3 standard allows 'putting' and 'getting' data in memory windows on other ranks**
  - No corresponding MPI function call is necessary on the other rank
- **For hardware supporting this paradigm, large scaling results can be better**

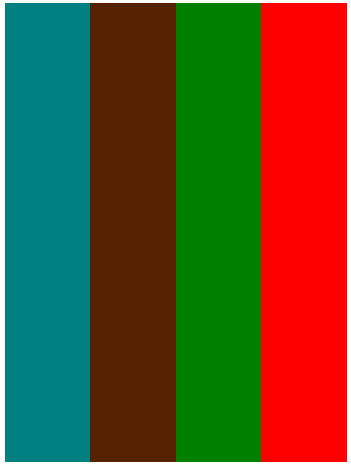
Parallel IO

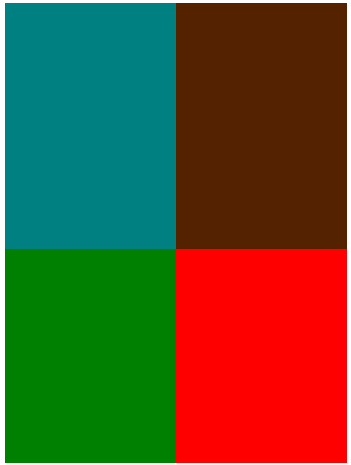
- **Three strategies**
  - Every rank writes its own file; rank 0 writes a master file
    - Ok for small parallelization
    - Bad for large parallelization
  - Use MPI-IO (or a parallel library) to write one file
  - Hybrid approach
    - Chunks of MPI ranks send data to 1 rank in the chunk
    - Each chunk writes its own file; rank 0 writes a master file

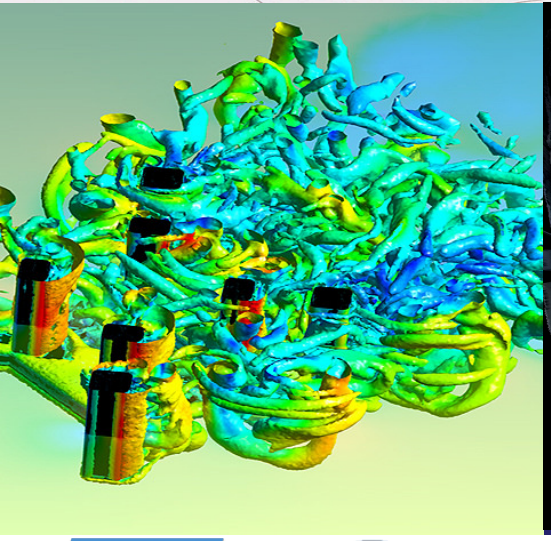
Resources

- **MPI Tutorial**
  - http://mpitutorial.com/
  - https://computing.llnl.gov/tutorials/mpi/
- **MPI IO overview from TACC**
  - https://www.tacc.utexas.edu/documents/13601/900558/MPI-IO-Final.pdf

How to parallelize a 2D Domain?







Los Alamos
NATIONAL LABORATORY

EST.1943

Delivering science and technology
to protect our nation
and promote world stability

Parallel Computing SRI

ISTI
INFORMATION SCIENCE & TECHNOLOGY INSTITUTE

NSEC

NNSA
National Nuclear Security Administration

Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA