

MPI

https://github.com/t-brown/RMACC_2016

August 11, 2016

Timothy Brown



Overview

Background

MPI

Communicator

Data Types

Communications

Point to Point Communications

Collective Communications

Parallelism

Parallelism can be achieved across many levels

Nodes

MPI



Threads

OpenMP



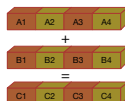
Instructions

ILP

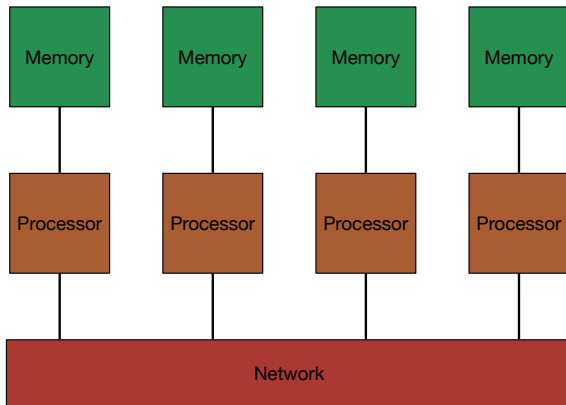
I1: add R1, R2, R3
I2: sub R4, R1, R5
I3: xor R10, R2, R11

Data

SIMD



Distributed Memory Model



- ▶ All processors see a different view of data.
- ▶ Processors interact and synchronize by passing messages.

Message Passing

- ▶ Most natural and efficient paradigm for distributed-memory systems.
- ▶ Two-sided, send and receive communication between processes.
- ▶ Efficiently portable to shared-memory or almost any other parallel architecture: “assembly language of parallel computing” due to universality and detailed, low-level control of parallelism.

- ▶ Provides natural synchronization among processes (through blocking receives, for example), so explicit synchronization of memory access is unnecessary.
- ▶ Sometimes deemed tedious and low-level, but thinking about locality promotes:
 - ▶ good performance
 - ▶ scalability
 - ▶ portability

Programming

- ▶ MPI (Message Passing Interface).
- ▶ Message passing standard, universally adopted library of communication routines callable from C, C++, Fortran, Java, (Python).
- ▶ 125+ functions—I will introduce a small subset of functions.

MPI

- ▶ MPI has been developed in three major stages, MPI 1, MPI 2 and MPI 3.
 - ▶ MPI 1 – 1994
 - ▶ MPI 2 – 1996
 - ▶ MPI 3 – 2012
- ▶ MPI Forum
<http://www.mpi-forum.org/docs/docs.html>
- ▶ MPI Standard
<http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf>
- ▶ *Using MPI* and *Using Advanced MPI*
<http://www.mcs.anl.gov/research/projects/mpi/usingmpi/>
- ▶ Online MPI tutorial
<http://mpitutorial.com/beginner-mpi-tutorial/>

MPI 1

- ▶ Features of MPI-1 include:
 - ▶ Point-to-point communication.
 - ▶ Collective communication process.
 - ▶ Groups and communication domains.
 - ▶ Virtual process topologies.
 - ▶ Environmental management and inquiry.
 - ▶ Profiling interface bindings for Fortran and C.

MPI 2

- ▶ Additional features of MPI-2 include:
 - ▶ Dynamic process management input/output.
 - ▶ One-sided operations for remote memory access (update or interrogate).
 - ▶ Memory access bindings for C++.

MPI 3

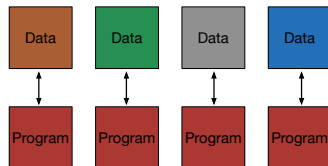
- ▶ Updates, corrections and clarifications of MPI-3 include:
 - ▶ Non-blocking collectives.
 - ▶ New one-sided communication operations.
 - ▶ Fortran 2008 bindings.

Implementations

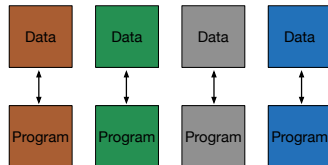
- ▶ MPICH <ftp://ftp.mcs.anl.gov/pub/mpi>
- ▶ OpenMPI <http://www.open-mpi.org/>
- ▶ Intel MPI
<https://software.intel.com/en-us/intel-mpi-library>
- ▶ SGI
- ▶ Cray
- ▶ IBM

Programming Models

- ▶ Single Program Multiple Data (SPMD)
 - ▶ Same program runs on each process.



- ▶ Multiple Programs Multiple Data (MPMD)
 - ▶ Different programs runs on each process.



Compiling

Most MPI implementations have wrapper scripts for the compiler.

_____ C _____
`mpicc -o a.out a.c`

_____ Fortran _____
`mpifc -o a.out a.f90`

If not, you will need to specify the include path, library path and link to the library.

Execution

Once your program has compiled you can run it:

- ▶ Through a batch system (SLURM for example).

```
login01 ~$ srun -N 1 --ntasks-per-node=12 ./a.out
```

- ▶ Through `mpiexec`.

```
node0001 ~$ mpiexec -n 12 ./a.out
```

Program Structure

C

```
#include <mpi.h>

int ierr = 0;
ierr = MPI_Init(&argc,
               &argv);
...
ierr = MPI_Finalize();
```

Fortran

```
use mpi

integer :: ierr
ierr = 0
call MPI_Init(ierr)
...
call MPI_Finalize(ierr)
```

- ▶ C returns error codes as function values.
- ▶ Fortran uses the last argument (ierr).

Note: All MPI calls *should* have man-pages.

Communicator

- ▶ A collection of processors working on some part of a parallel job.
- ▶ Used as a parameter for most MPI calls.
- ▶ Processors within a communicator are assigned ranks 0 to $n - 1$.
- ▶ `MPI_COMM_WORLD` includes all of the processors in your job.
- ▶ Can create subsets of `MPI_COMM_WORLD`

```
program hello
  use mpi
  implicit none

  integer :: ierr
  integer :: id, nprocs
  ierr = 0
  call MPI_Init(ierr)
  call MPI_Comm_rank(MPI_COMM_WORLD, id, ierr)
  call MPI_Comm_size(MPI_COMM_WORLD, nprocs, ierr)
  call MPI_Finalize(ierr)

end program hello
```

- ▶ Determine the process rank (id).
- ▶ Total number of processes (nprocs).

Data Types

| Fortran | Optional |
|----------------------|-------------|
| MPI_CHARACTER | |
| MPI_COMPLEX | 8, 16, 32 |
| MPI_DOUBLE_COMPLEX | |
| MPI_DOUBLE_PRECISION | |
| MPI_INTEGER | 1, 2, 4, 8 |
| MPI_LOGICAL | 1, 2, 4, 8 |
| MPI_REAL | 2, 4, 8, 16 |

| C |
|------------------------|
| MPI_CHAR |
| MPI_DOUBLE |
| MPI_FLOAT |
| MPI_INT |
| MPI_LONG_DOUBLE |
| MPI_LONG_LONG |
| MPI_LONG_LONG_INT |
| MPI_SHORT |
| MPI_SIGNED_CHAR |
| MPI_UNSIGNED |
| MPI_UNSIGNED_CHAR |
| MPI_UNSIGNED_LONG |
| MPI_UNSIGNED_LONG_LONG |
| MPI_UNSIGNED_SHORT |
| MPI_WCHAR |

Communications

- ▶ Bytes transferred from one processor to another.
- ▶ Specify destination, data buffer, and message ID (called a tag).
- ▶ Synchronous send: send call does not return until the message is sent.
- ▶ Asynchronous send: send call returns immediately, send occurs during other calculation ideally.
- ▶ Synchronous receive: receive call does not return until the message has been received (may involve a significant wait).
- ▶ Asynchronous receive: receive call returns immediately. When received data is needed, call a wait subroutine.
- ▶ Asynchronous communication used in attempt to overlap communication with computation.

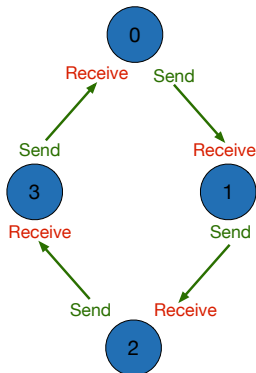
Point to Point Communications

- ▶ `MPI_Send()`
 - ▶ Does not return until the message data and envelope have been buffered in matching receive buffer or temporary system buffer.
 - ▶ Can complete as soon as the message was buffered, even if no matching receive has been executed by the receiver.
 - ▶ MPI buffers or not, depending on availability of space.
 - ▶ **Non-local**: successful completion of the send operation may depend on the occurrence of a matching receive.
- ▶ `MPI_Recv()`
 - ▶ The opposite of `MPI_Send()`.
 - ▶ Does not return until the message data has been copied into the destination buffer.
 - ▶ Stalls progress of program **but**:
 - ▶ Blocking sends and receives enforce process synchronization.
 - ▶ Enforces consistency of data.

Sending Data In A Ring

Sending n elements to the $rank + 1$ receive n elements from the $rank - 1$.

- ▶ Store them in an array of size $nprocs \times n$.
- ▶ Sum up and print the local results.



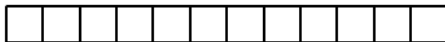
Ring Data

Nprocs = 4, N = 3

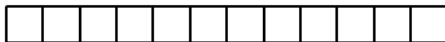
Rank

Data

0



1



2



3



Ring Data

Nprocs = 4, N = 3

Rank

Data

0

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

1

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

2

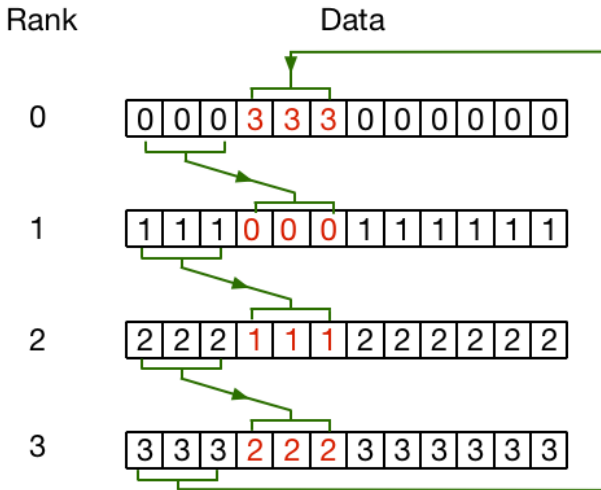
| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

3

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

Ring Data

Nprocs = 4, N = 3

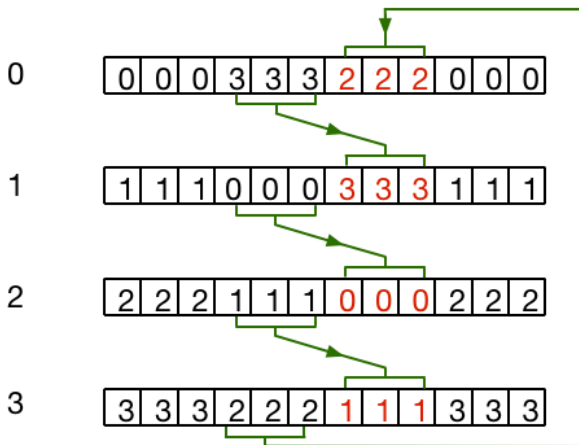


Ring Data

Nprocs = 4, N = 3

Rank

Data

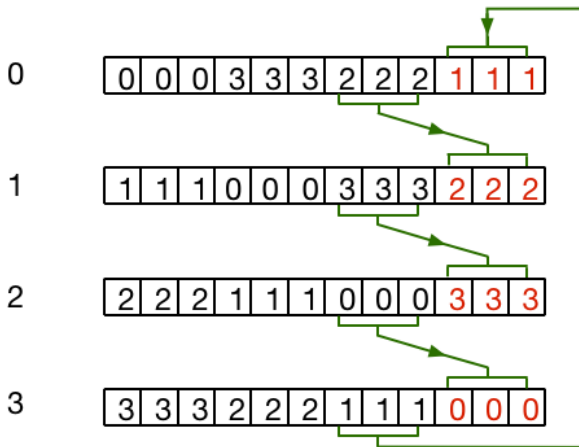


Ring Data

Nprocs = 4, N = 3

Rank

Data



Ring Data

Nprocs = 4, N = 3

Rank

Data

0

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|

1

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|

2

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 2 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|

3

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 3 | 3 | 2 | 2 | 2 | 1 | 1 | 1 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|

For Success

- ▶ Sender must specify a valid destination rank.
- ▶ Receiver must specify a valid source rank.
- ▶ The communicator must be the same.
- ▶ Tags must match.
- ▶ Message data types must match.
- ▶ Receiver's buffer must be large enough.

Ring Program

The program `ring.c` has been written using blocking `MPI_Send/Recv()`.

Compile and run the program with different array sizes:

- ▶ $N = 10$
- ▶ $N = 100000$

Deadlocks

A process waiting for a condition that will never become true.

- ▶ Easy to write send/receive code that deadlocks
 - ▶ Two processes: both receive before send.
 - ▶ Send tag does not match receive tag.
 - ▶ Process sends message to wrong destination process.

Non-Blocking Send & Receive

Non-Blocking Send `MPI_Isend()` and Receive `MPI_Irecv()`.

- ▶ Same syntax as `MPI_Send()`/`MPI_Recv()` with the addition of a request handle.
- ▶ Request handle (integer in Fortran; `MPI_Request` in C) is used to check for completeness of the send.
- ▶ These calls returns immediately.
- ▶ Data in the buffer may not be accessed until the user has completed the send/receive operation.
- ▶ The send is completed by a successful call to `MPI_Test()` or a call to `MPI_Wait()`.

Non-Blocking Wait

Non-Blocking Wait for ISend completion `MPI_Wait()`

`MPI_Wait(request, status, ierr)`

- ▶ Request is the handle returned by the non-blocking send or receive call.
- ▶ Upon return, status holds source, tag, and error code information.
- ▶ This call does not return until the non-blocking call referenced by request has completed.
- ▶ Upon return, the request handle is freed.
- ▶ If request was returned by a call to `MPI_Isend()`, return of this call indicates nothing about the destination process.

Wait for any specified send or receive to complete.

`MPI_Waitany(count, requests, index, status, ierr)`

- ▶ Requests is an array of handles returned by nonblocking send or receive calls.
- ▶ Count is the number of requests.
- ▶ This call does not return until a non-blocking call referenced by one of the requests has completed.
- ▶ Upon return, index holds the index into the array of requests of the call that completed.
- ▶ Upon return, status holds source, tag, and error code information for the call that completed.
- ▶ Upon return, the request handle stored in `requests[index]` is freed.

Wait for all specified send or receive to complete.

`MPI_Waitall(count, requests, status, ierr)`

- ▶ `Requests` is an array of handles returned by nonblocking send or receive calls.
- ▶ `Count` is the number of requests.
- ▶ This call does not return until all non-blocking call referenced by one of the `requests` has completed.
- ▶ Upon return, `status` holds source, tag, and error code information for all the call that completed.
- ▶ Upon return, the request handle stored in `requests` is freed.

Tests for the completion of a specific send or receive.

`MPI_Test(request, flag, status, ierr)`

- ▶ Request is a handle returned by a non-blocking send or receive call.
- ▶ Upon return, `flag` will have been set to true if the associated non-blocking call has completed. Otherwise it is set to false.
- ▶ If `flag` returns true, the request handle is freed and `status` contains source, tag, and error code information.
- ▶ If request was returned by a call to `MPI_Isend()`, return with `flag` set to true indicates nothing about the destination process.

Non-Blocking Ring

A fixed ring program called `iring.f90` uses no-blocking `MPI_Isend/Irecv()`.

Collective Communications

- ▶ Synchronization points:
- ▶ One to all:
 - ▶ Broadcast data to all ranks.
 - ▶ Scatter (all and parts).
- ▶ All to one:
 - ▶ Gather from a group.
 - ▶ Gather varying amounts from all.
 - ▶ Reduce values on all.
- ▶ All to all
 - ▶ Gather and distribute to all.
 - ▶ Gather and distribute various amounts to all.
 - ▶ Combines values from all processes and distribute.

Synchronization

- Synchronization between MPI processes in a group.

`MPI_Barrier(comm, ierr)`

`comm` Communicator.

`ierr` Error status.

Broadcast

Broadcasts a message from the process with rank root to all other processes of the group.

`MPI_Bcast(buf, count, data_type, root, comm, ierr)`

buf Beginning address of data.

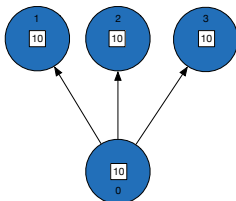
count Length of the source array (in elements).

data_type MPI type of data.

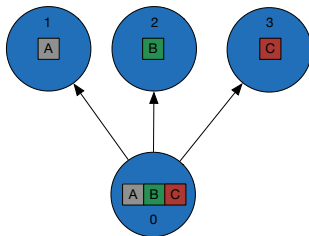
root Rank of broadcast root.

comm Communicator.

ierr Error status.

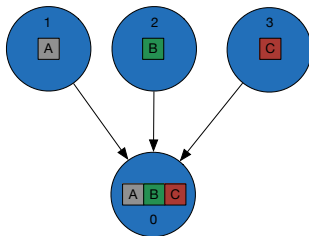


Scatter



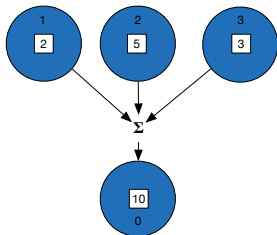
```
MPI_Scatter(sendbuf, sendcount, sendtype, &  
            recvbuf, recvcnt, recvtpe, &  
            root, comm, ierr)
```

Gather



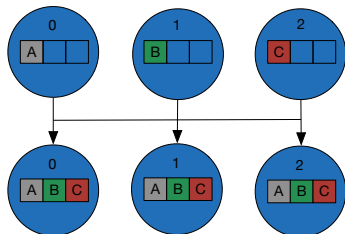
```
MPI_Gather(sendbuf, sendcount, sendtype, &  
          recvbuf, recvcount, recvtype, &  
          root, comm, ierr)
```

Reduction



```
MPI_Reduce(sendbuf, recvbuf, count, &  
           datatype, op, root, comm, &  
           ierr)
```

All Gather



```
MPI_Allgather(sendbuf, sendcount, sendtype, &  
              recvbuf, recvcount, recvtype, &  
              root, comm, request, ierr)
```

All Reduce Ring

A version of the ring program called `rring.c` uses `MPI_Allreduce`.

Questions?

Online Survey

<Timothy.P.Brown@noaa.gov>

License

© University of Colorado Boulder, 2016

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>

When attributing this work, please use the following text:
“MPI”, CIRES, University of Colorado Boulder, 2016. Available under a Creative Commons Attribution 4.0 International License.

