Introduction to MPI

Introduction to MPI

Presenter: Mea Trahan

Contact: Daniel.Trahan@colorado.edu

Website: www.colorado.edu/rc

Sign in: http://tinyurl.com/curc-names

Slides: https://github.com/ResearchComputing/MPI-Spring-2020

RMACC Forum

- Post your HPC questions to a growing community of users!
- https://ask.cyberinfrastructure.org/c/rmacc/65

Overview

- What is MPI
- Message Passing
- Compiling and Linking MPI
- Group Commands

Note about examples…

- To avoid any issues with compilers or MPI distributions, we will be running today's examples on Summit.
- Log into summit using the temporary accounts provided:

```
ssh <username>@tlogin1.rc.colorado.edu
```

- Type in temporary password when prompted.
- Navigate to a compile node and run the following command.

```
ssh scompile
module load gcc/8.2.0 openmpi
```

RMACC Summit reference

- `nano` – Simple text editor
- `vim` – Less simple text editor

- `module load <software>` – Load software into the environment
- `module avail` – Lists all software available to be loaded

- `sbatch <script>` - Submits a job script to the cluster
- `srun <application>` - Runs an application on the cluster

Why Parallel?

- Modern CPUs are capped in clock speed…
- Instead of increasing clock speed, add more transistors!
- Multiple processors on a single chip.
- Parallelism required to utilize full power!

MPI

- MPI or Message Passing Interface is a portable programming standard that allows for the communication of processes distributed either on a single or multiple compute nodes.
- Operates on processes communication through messages passing data or instructions.
- Not a single library, but a programming standard.
  - Different implementations of the standard require recompilation.
  - Same base function calls across every implementation.

MPI Implemtations

- MPICH
  - Reference MPI implementation where features of the MPI standard are implemented first.
  - High performance but built for certain network fabrics.
- Intel MPI
  - Distribution optimized with the intel compiler in mind.
  - Shares an ABI with MPICH and MPICH derivatives
- OpenMPI (Not to be confused with OpenMP!)
  - Popular implementation of MPI
  - Lower performance but usually easier to use.

Minimum MPI application

- To start let's create a minimum MPI application.
  - Need 2 Primary functions from the MPI library:
- `MPI_Init()` - Sets up the MPI environment.
- `MPI_Finalize()` - Cleans up the MPI environment.

C/C++

```
MPI_Init(&argc, &argv);
```

```
MPI_Finalize();
```

Fortran

```
MPI_INIT(ierr);
```

```
MPI_FINALIZE(ierr);
```

Some Useful MPI Variables and Functions

- `MPI_Status Status` - MPI Status Variable required to receive messages.
- `MPI_comm_size()` - Returns the total amount of ranks handled by the MPI communicator to a designated variable.
- `MPI_comm_rank()` - Returns the "rank" of the process to a designated variable.
  - Ranks start from 0

C/C++

```
MPI_Comm_size(Comm, &size);
```

```
MPI_Comm_rank(Comm, &rank);
```

Fortran

```
MPI_COMM_SIZE(comm, &size, ierr);
```

```
MPI_COMM_RANK(comm, &rank, ierr);
```

MPI Wrappers

- Most MPI distributions come with a compiler wrapper to automatically link required MPI libraries to what you need to compile.
- Not a separate compiler!
- Different for every implementation of MPI…

	IntelMPI w/ Intel Compiler	OpenMPI w/ GCC
C	`mpiicc`	`mpicc`
C++	`mpiicpc`	`mpic++`
Fortran	`mpiifort`	`mpifort`

MPI Hello World

- Now that we know the bare minimums of what goes into an MPI application lets go ahead and make our first MPI application.
- Lets go ahead and make hello-mpi.c
  - Prints out “Hello world from process x of n”

MPI Hello World

C/C++

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char* argv[]){

    int corenum, totcores;
    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &totcores);
    MPI_Comm_rank(MPI_COMM_WORLD, &corenum);

    printf("Hello World from process: %d of %d\n", corenum, totcores);

    MPI_Finalize();
    return 0;

}
```

Compiling an MPI application

- Once we have completed our MPI code, we can then compile it just as we would with any C application but using the wrapper script instead.
- For this tutorial we will use GCC 8.2.0 w/ OpenMPI
- Wrapper provides additional flags with the rest being passed to the compiler.

C/C++

```
mpiicc hello-mpi.c -o hello-mpi.exe
```

Fortran

```
mpiifort hello-mpi.f90 -o hello-mpi.exe
```

Running your MPI application (1)

- Finally we can run an MPI application
- Various commands to run an MPI application
  - `mpiexec` - Standardized MPI run script available on most modern MPI implementations.
  - `srun` – Slurm managed run script. Allocates resources from slurm, then runs the application.
  - `mpirun` – Implementation dependent run script.
- We will be using `mpiexec` primarily as our startup script.

Running your MPI application (2)

- `mpiexec` - General MPI run script.
  - Flags: -n for procs, -f for a nodelist (usually not needed with slurm)
  - Requires jobscript!

mpiexec

```
mpiexec –n 4 hello-mpi.exe
```

- `srun` - Slurm handled run script.
  - Slurm flags: -n for procs, -N for nodes, -t for time, and -p for partition

srun

```
srun –n 4 –N 2 –t 00:01:00 hello-mpi.exe
```

Some MPI Nuance

- Applications run as MPI processes are run entirely in parallel.
  - No parallel ‘sections’ unless explicitly defined.
- MPI processes are managed by an additional process manager.
  - Process manager implementation dependent.
- MPI applications can be run on multiple nodes and/or multiple cores.
- MPI applications can be mixed with other parallelization tools like OpenMP.

Message Passing

- Now that we've learned how to run a basic MPI application lets look at how we can communicate with processes.
- Message passing is a way for a process to invoke actions on another process.
- In MPI, message passing is usually done from a single process to another.
  - Blocking
  - Non-Blocking

Blocking Messages

- A Blocking Message is a message that holds the process until the desired buffer is either safe to write to or ready to use.
- Most common type of Message Passing
- Done with 2 commands:
  - MPI_Send – Sends a message containing some buffer to be received by another process in the communicator
  - MPI_Recv – Receives a message containing some buffer from another process in the communicator
  - Commands both require a variety of parameters.

MPI_Send and MPI_Recv (1)

C/C++

```
MPI_Send(&buf, count, mpi-datatype, 1, 10, mpi-com)
MPI_Recv(&buf, count, mpi-datatype, 3, 10, mpi-com, &status)
```

- `&buf` – the variable being sent/written to
- `count` – the amount of variables being sent (if array)
- `mpi-datatype` – type of the variable being sent
- `dest/src` – rank of process being sent to/received from
- `tag` – unique identifier of message
- `mpi-com` – MPI communicator
- `&status` – MPI status variable

MPI_Send and MPI_Recv (2)

Fortran

```
MPI_SEND(buf, count, mpi-datatype, src, tag, mpi-com, ierr)
MPI_Recv(buf, count, mpi-datatype, src, tag, mpi-com, status, ierr)
```

- `buf` – the variable being sent/written to
- `count` – the amount of variables being sent (if array)
- `mpi-datatype` – type of the variable being sent
- `dest/src` – rank of process being sent to/received from
- `tag` – unique identifier of message
- `mpi-com` – MPI communicator
- `status` – MPI status variable
- `ierr` - Error Handling Variable

MPI Datatypes

- Since MPI is a standard meant to fit a variety of machines, MPI has predefined types that must be provided to send and receive calls.
- Some of these datatypes are listed below. For a full list visit: https://www.mpich.org/static/docs/v3.2.x/www3/Constants.html

	C/C++	Fortran
integer	MPI_INT	MPI_INTEGER
double	MPI_DOUBLE	MPI_DOUBLE_PRECISION
boolean	MPI_C_BOOL	MPI_LOGICAL
char	MPI_CHAR	

Example: Message Passing

- Let's create an application that uses 4 processes to pass the process's rank ID to the process next in rank.
- Should be passed to a new variable.
- Once the messages have been passed then simply print out the initial variable and the passed variable on every rank.

Non-Blocking Messages

- MPI also offers non-blocking alternatives to standard blocking message passing.
- Usually done when a process can do work while sending or receiving a message to increase speed.
  - `MPI_Isend(...)`
  - `MPI_Irecv(...)`
- Same parameters as blocking send and receives but with an additional “request” parameter at the end.
  - Used with testing functions to check if non-blocking messages have completed.

Collective Operations

- MPI provides Collective operations as well!
  - Must be called on all processes within the communicator
- 3 Categories of Collective operations:
  - Synchronization
  - Data Movement
  - Reductions
- Both blocking and non-blocking variants

Synchronization

- The primary function that is used to synchronize MPI process is the `MPI_Barrier(...)` function.
  - Blocks tasks within a communicator until they have all reached the barrier.
  - Takes in a communicator as a parameter.

C/C++

```
MPI_Barrier(mpi-com)
```

Fortran

```
MPI_BARRIER(mpi-com, ierr)
```

Data Movement

- Many functions exist within MPI to distribute and collect data through collective means.
  - `MPI_Bcast(...)` – copies data from one process to all processes within a communicator.
  - `MPI_Scatter(...)` – distributes an array of data to every process within a communicator.
  - `MPI_Gather(...)` – collects data from every process on a communicator into a single process.
  - `MPI_Allgather(...)` – collects data from every process on a communicator into every process on a communicator.

Reductions

- Reductions are collective functions that apply a mathematical operation on data present on every process.
  - `MPI_Reduce(...)` – Apply a mathematical operation on all processes and store on one node.
  - `MPI_Allreduce(...)` – Apply a mathematical operation on all processes and store on all nodes.
- A Variety of Mathematical operations can be applied to these functions.
  - MPI_SUM – Sum reduction
  - MPI_PROD – Product reduction
  - MPI_MAX – Maximum of values
  - MPI_MIN – Minimum of values

Example: Collective Synchronization

- Create an MPI application prints out the ranks of every process in order from 0 to n.
- Use the `MPI_Barrier` Method!

Additional Resources

- https://mpitutorial.com/ - Community Driven MPI tutorial
- https://computing.llnl.gov/tutorials/mpi/ - Laurence Livermore National Laboratory
- https://www.mcs.anl.gov/research/projects/mpi/tutorial/gropp/talk.html - Argonne National Laboratory

Questions and Thank You!

Contact: Daniel.Trahan@colorado.edu

Website: www.colorado.edu/rc

Sign in: http://tinyurl.com/curc-names

Slides: https://github.com/ResearchComputing/MPI-Spring-2020

Survey: http://tinyurl.com/curc-survey18