Practical Introduction to Message-Passing Interface (MPI)

October 1st, 2015

By: Pier-Luc St-Onge
Partners and Sponsors
Setup for the workshop

1. Get a user ID and password paper (provided in class):
   
   ##:
   
   ********************

2. Access to local computer (replace ## and ___ with appropriate values, “___” is provided in class):
   
   a. User name: csuser##
   b. Password: ___@[S##

3. SSH connection to Guillimin (replace ********************):
   
   a. Host name: guillimin.hpc.mcgill.ca
   b. User name: class##
   c. Password: ********************
Import Examples and Exercises

● **On Guillimin:**

```bash
cp -a /software/workshop/cq-formation-intro-mpi ~/.git

cd cq-formation-intro-mpi
```

● **From GitHub:**

```bash
module add git # If on Guillimin

git clone -b mcgill \\https://github.com/calculquebec/cq-formation-intro-mpi.git

cd cq-formation-intro-mpi
```
Outline

● General Concepts
● A First Code
● MPI Data Types
● Point-to-point Communications
● Synchronization Between Processes
● Collective Communications
● Conclusion
General Concepts
Parallel Algorithm - Why?

- When do you need to parallelize your algorithm?
  - The algorithm takes **too much time** on a single processor or even on a full compute node.
  - The algorithm uses **too much memory**
    - The algorithm may fit in **cache memory**, but only if the data is split on multiple processors.
  - The algorithm does **a lot of read and write operations** (I/O) on the network storage
    - The data does not fit on a single node
    - Bandwidth and latency of a single node
Vocabulary and Concepts

● **Serial tasks**
  ○ Any task that cannot be split in two simultaneous sequences of actions
  ○ Examples: starting a process, reading a file, any communication between two processes

● **Parallel tasks**
  ○ Data parallelism: same action applied on different data. Could be serial tasks done in parallel.
  ○ Process parallelism: one action on one set of data, but action is split in multiple processes or threads.
    ■ Data partitioning: rectangles or blocks
Serial Code Parallelization

- **Implicit** Parallelization | minimum work for you
  - Threaded **libraries** (MKL, ACML, GOTO, etc.)
  - **Compiler directives** (OpenMP)
  - Good for desktops and **shared memory** machines

- **Explicit** Parallelization | work is required!
  - You tell what should be done on what CPU
  - Solution for **distributed clusters** (shared nothing!)

- **Hybrid** Parallelization | work is required!
  - Mix of implicit and explicit parallelization
    - Vectorization and parallel CPU instructions
  - Good for accelerators (CUDA, OpenCL, etc.)
Distributed Memory Model

Process 1

A(10)

Different variables!

Process 2

A(10)

Network
In Practice on a Cluster

- The scheduler provides a list of worker node names
- The job spawns processes on each worker node
  - How to manage all processes?
- Processes must communicate together
  - How to manage communications? By using “sockets” each time? No!
Solution - MPI or Message Passing Interface!

- Model for distributed memory approach
  - Each process is identified by unique integer
  - Programmer manages memory by placing data in a particular process
  - Programmer sends data between processes (point-to-point communications)
  - Programmer performs collective operations on sets of processes

- MPI is a specification for a standardized library: subroutines linked with your code
Different MPI Implementations

Different MPI Modules

- OpenMPI, MVAPICH2, Intel MPI, …
  - They come with a different compilation wrapper
    - Compilation arguments may differ
  - Execution arguments may also differ
    - mpiexec or mpirun, -n or -np

- When working on a cluster, please use provided MVAPICH2 or OpenMPI libraries: they have been compiled with Infiniband support (20-40Gbps).

- MPI library must match compiler used (Intel, PGI, or GCC) both at compile and at run time.
  - {GNU, Intel, PGI} * {OpenMPI, MVAPICH2}
A First Code
Basic Features of MPI Program

- **Include** basic definitions
  - C:  `#include <mpi.h>`
  - Fortran: `INCLUDE 'mpif.h'` (or `USE mpi`)
- **Initialize** MPI environment
- **Get** information about processes
- **Send** information between **two specific** processes (point-to-point communications)
- **Send** information between **groups of processes** (collective communications)
- **Terminate** MPI environment
Six Basic MPI Functions

● You need to know these six functions:
  ○ `MPI_Init`: initialize MPI environment
  ○ `MPI_Comm_size`: number of processes in a group
  ○ `MPI_Comm_rank`: unique integer value identifying each process in a group (0 <= rank < size)
  ○ `MPI_Send`: send data to another process
  ○ `MPI_Recv`: receive data from another process
  ○ `MPI_Finalize`: close MPI environment

● `MPI_COMM_WORLD` is a default communicator (defined in `mpi.h`), refers to the group of all processes in the job

● Each statement executes **independently** in each process
Example: A Smiley from N Processors

- **smiley.c**

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

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( "%3d/%-3d   :-D
", rank, size );
    MPI_Finalize();
    return 0;
}
```

- **smiley.f90**

```fortran
PROGRAM smiley
IMPLICIT NONE
INCLUDE 'mpif.h'
INTEGER ierr, rank, size
CALL MPI_Init(ierr)
CALL MPI_Comm_rank (MPI_COMM_WORLD, &rank, ierr)
CALL MPI_Comm_size (MPI_COMM_WORLD, &size, ierr)
WRITE(*,*) rank, '/', size, '  :-D'
CALL MPI_Finalize(ierr)
END PROGRAM smiley
```
Compiling your MPI Code

● **Not** defined by the standard

● More or less similar for all implementations:
  ○ Need to specify include directory and MPI library
  ○ But usually a compiler wrapper (*mpicc*, *mpif90*) does it for you automatically

● On the Guillimin cluster:

  ```
  module add ifort_icc openmpi
  mpicc smiley.c -o smiley
  mpif90 smiley.f90 -o smiley
  ```
Running your MPI Code

- Not defined by the standard
- A launching program (mpirun, mpiexec, mpirun_rsh, ...) is used to start your MPI program
- Particular choice of launcher depends on MPI implementation and on the machine used
- A hosts file is used to specify on which nodes to run MPI processes (--hostfile nodes.txt), but will run on localhost by default or will use the host file provided by the batch system
- On a worker node of Guillimin:
  
  mpiexec -n 4 ./smiley
Exercises - Part 1

1. If not done already, log in to Guillimin:
   
   ssh class##@guillimin.hpc.mcgill.ca

2. Check for loaded software modules:
   
   module list

3. See all available modules:
   
   module av

4. Load necessary modules:
   
   module add ifort_icc openmpi

5. Check loaded modules again

6. Verify that you have access to the correct mpi package:
   
   which mpicc                # which mpif90
   /software/CentOS-6/tools/openmpi-1.6.3-intel/bin/mpicc
Exercises - Part 2

1. Make sure to get workshop files
   a. See page 4: from /software or from GitHub
      
      ```
      cd ~/cq-formation-intro-mpi
      ```

2. Compile code:
   
   ```
   mpicc smiley.c -o smiley
   mpif90 smiley.f90 -o smiley
   ```

3. Edit `smily.pbs`: reserve 4 processors (`ppn=4`) for 5 minutes (00:05:00). Edit:
   
   ```
   mpiexec -n 4 ./smiley > smiley.out
   ```
Exercises - Part 3

1. Verify your `smiley.pbs`:
   ```bash
   #!/bin/bash
   #PBS -l nodes=1:ppn=4
   #PBS -l walltime=00:05:00
   #PBS -V
   #PBS -N smiley
   cd $PBS_O_WORKDIR
   mpiexec -n 4 ./smiley > smiley.out
   ```

2. Submit your job:
   ```bash
   qsub smiley.pbs
   ```

3. Check the job status:
   ```bash
   qstat -u $USER
   ```

4. Check the output (`smiley.out`)
Exercises - Part 4

● Copy `smiley.{c,f90}` to `smileys.{c,f90}`

● Modify `smileys.c` or `smileys.f90` such that each process will print a different smiley:
  ○ If rank is 0, print `:-|`
  ○ If rank is 1, print `:-)`
  ○ If rank is 2, print `:-D`
  ○ If rank is 3, print `:-P`

● Compile the new code

● Copy `smiley.pbs` to `smileys.pbs` and modify the copied version accordingly
  ○ Submit a new job and check the output
Another Example - hello.{c,f90}

```
#include <math.h>
// [...] 
float a, b;
if (rank == 0) {
    a = sqrt(2.0);
    b = 0.0;
}
if (rank == 1) {
    a = 0.0;
    b = sqrt(3.0);
}
printf("On proc %d: a, b = \t%f\t%f\n", rank, a, b);
```
MPI Data Types

Portable data types for heterogeneous compute nodes
# MPI Basic Data Types (C)

<table>
<thead>
<tr>
<th>MPI Data type</th>
<th>C Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_CHAR</td>
<td>char</td>
</tr>
<tr>
<td>MPI_SHORT</td>
<td>short</td>
</tr>
<tr>
<td>MPI_INT</td>
<td>int</td>
</tr>
<tr>
<td>MPI_LONG</td>
<td>long int</td>
</tr>
<tr>
<td>MPI_UNSIGNED_CHAR</td>
<td>unsigned char</td>
</tr>
<tr>
<td>MPI_UNSIGNED_SHORT</td>
<td>unsigned short</td>
</tr>
<tr>
<td>MPI_UNSIGNED</td>
<td>unsigned int</td>
</tr>
<tr>
<td>MPI_UNSIGNED_LONG</td>
<td>unsigned long int</td>
</tr>
<tr>
<td>MPI_FLOAT</td>
<td>float</td>
</tr>
<tr>
<td>MPI_DOUBLE</td>
<td>double</td>
</tr>
<tr>
<td>MPI_LONG_DOUBLE</td>
<td>long double</td>
</tr>
</tbody>
</table>
### MPI Basic Datatypes (Fortran)

<table>
<thead>
<tr>
<th>MPI Data type</th>
<th>Fortran Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>MPI_INTEGER8</td>
<td>INTEGER(selected_int_kind(18))</td>
</tr>
<tr>
<td>MPI_REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>MPI_DOUBLE_PRECISION</td>
<td>DOUBLE_PRECISION</td>
</tr>
<tr>
<td>MPI_COMPLEX</td>
<td>COMPLEX</td>
</tr>
<tr>
<td>MPI_DOUBLE_COMPLEX</td>
<td>DOUBLE_COMPLEX</td>
</tr>
<tr>
<td>MPI_LOGICAL</td>
<td>LOGICAL</td>
</tr>
<tr>
<td>MPI_CHARACTER</td>
<td>CHARACTER(1)</td>
</tr>
</tbody>
</table>
# MPI Advanced Datatypes

<table>
<thead>
<tr>
<th>MPI Data type</th>
<th>C or Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MPI_PACKED</code></td>
<td>Data packed with <code>MPI_Pack</code></td>
</tr>
<tr>
<td><code>MPI_BYTES</code></td>
<td>8 binary digits</td>
</tr>
</tbody>
</table>
Point-to-point Communications

qsub -I -l nodes=1:ppn=2 -l walltime=5:00:00
Six Basic MPI Functions

- **MPI_Init**: initialize MPI environment
- **MPI_Comm_size**: number of processes in a group
- **MPI_Comm_rank**: unique integer value identifying each process in a group (0 <= rank < size)
- **MPI_Send**: send data to another process
- **MPI_Recv**: receive data from another process
- **MPI_Finalize**: close MPI environment
MPI_Send / MPI_Recv

- Passing message between two different MPI processes (point-to-point communication)
- If one process sends, another initiates the matching receive
- The exchange data types are predefined for portability
- MPI_Send / MPI_Recv is **blocking**!
  (There are also non-blocking versions)
MPI: Sending a message

C: MPI_Send(&data, count, data_type, dest, tag, comm)

Fortran: MPI_Send(data, count, data_type, dest, tag, comm, ierr)

- **data**: variable to send
- **count**: number of data elements to send
- **data_type**: type of data to send
- **dest**: rank of the receiving process
- **tag**: the label of the message
- **comm**: communicator - set of involved processes
- **ierr**: error code (return value for C)
MPI: Receiving a message

C: MPI_Recv(&data, count, data_type, source, tag, comm, &status)

Fortran: MPI_Recv(data, count, data_type, source, tag, comm, status, ierr)

- **source**: rank of the sending process (or can be set to MPI_ANY_SOURCE)
- **tag**: must match the label used by sender (or can be set to MPI_ANY_TAG)
- **status**: a C structure (MPI_Status) or an integer array with information in case of an error (source, tag, actual number of bytes received)
- MPI_Send and MPI_Recv are **blocking**!
Example 1

// examples/sendRecvEx1.c

int rank, size, buffer = -1, tag = 10;
MPI_Status status;

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

if (size >= 2 && rank == 0) {
    buffer = 33;
    MPI_Send( &buffer, 1, MPI_INT, 1, tag, MPI_COMM_WORLD );
}
if (size >= 2 && rank == 1) {
    MPI_Recv( &buffer, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &status );
    printf("Rank %d	buffer= %d
", rank, buffer);
    if (buffer != 33) printf("fail\n");
}

MPI_Finalize();
Exercise: Sending a Matrix

- Goal: sending a matrix 4x4 from process 0 to process 1
- Edit file `send_matrix.{c,f90}
- Compile your modified code
- Run it with 2 processes
Synchronization Between Processes
Processes Waiting for Communications

- When using blocking communications, unbalanced workload may cause processes to be waiting
- Worst case: everyone waiting after everyone
  - Must avoid cases of deadlocks
  - Some bugs may lead to non-matching send/receive
MPI_Send / MPI_Recv

Example 2

/////////////////////////////////////////////////////////////////////
// Should not work //
// Why?              //
/////////////////////////////////////////////////////////////////////

if (size >= 2 && rank == 0) {
    MPI_Send( &buffer1, 1, MPI_INT, 1, 10, MPI_COMM_WORLD );
    MPI_Recv( &buffer2, 1, MPI_INT, 1, 20, MPI_COMM_WORLD, &status );
}

if (size >= 2 && rank == 1) {
    MPI_Send( &buffer2, 1, MPI_INT, 0, 20, MPI_COMM_WORLD );
    MPI_Recv( &buffer1, 1, MPI_INT, 0, 10, MPI_COMM_WORLD, &status );
}
// Exchange Send/Recv order //
// on one processor        //

if (size >= 2 && rank == 0) {
    MPI_Send( &buffer1, 1, MPI_INT, 1, 10, MPI_COMM_WORLD );
    MPI_Recv( &buffer2, 1, MPI_INT, 1, 20, MPI_COMM_WORLD, &status );
}

if (size >= 2 && rank == 1) {
    MPI_Recv( &buffer1, 1, MPI_INT, 0, 10, MPI_COMM_WORLD, &status );
    MPI_Send( &buffer2, 1, MPI_INT, 0, 20, MPI_COMM_WORLD );
}
MPI_Send / MPI_Recv

Example 2 - Solution B

(/)///////////////////
// Use non-blocking send: Isend //
(/)///////////////////

MPI_Request request;

if (size >= 2 && rank == 0) {
    MPI_Isend( &buffer1, 1, MPI_INT, 1, 10, MPI_COMM_WORLD, &request);
    MPI_Recv( &buffer2, 1, MPI_INT, 1, 20, MPI_COMM_WORLD, &status );
}

if (size >= 2 && rank == 1) {
    MPI_Isend( &buffer2, 1, MPI_INT, 0, 20, MPI_COMM_WORLD, &request);
    MPI_Recv( &buffer1, 1, MPI_INT, 0, 10, MPI_COMM_WORLD, &status );
}

MPI_Wait( &request, &status ); // Wait until send is complete
Non-blocking Communications

- **MPI_Isend()** starts the transfer and returns control
  - **Advantage**: between the start and the end of transfer the code can do other things
  - **Warning**: during a transfer, the buffer cannot be reused or deallocated
  - **MPI_Request**: additional structure for the communication

- **You need to check for completion!**
  - **MPI_Wait(&request, &status)**: the code waits for the completion of the transfer
Exercise: Exchanging Vectors

- Goal: exchanging a small vector of data
- Edit file `exchange.{c,f90}`
- Compile your modified code
- Run it with 2 processes
Collective Communications

One step beyond
Collective Communications

● Involve ALL processes in the communicator (MPI_COMM_WORLD)
● **MPI_Bcast**: Same data sent from “root” process to all the others
● **MPI_Reduce**: “root” process collects data from the others and performs an operation (min, max, add, multiply, etc ...)
● **MPI_Scatter**: distributes variables from the “root” process to each of the others
● **MPI_Gather**: collects variables from all processes to the “root” one
MPI_Bcast Example

```c
int rank, size, root = 0;
float a[2];
//--...

if (rank == root) {
    a[0] = 2.0f;
    a[1] = 4.0f;
}

MPI_Bcast( a, 2, MPI_FLOAT, root, MPI_COMM_WORLD );

//-- Print result
```
MPI_Reduce
Example

int rank, size, root = 0;
float a[2], res[2];
// ...
a[0] = 2 * rank + 0;
a[1] = 2 * rank + 1;

MPI_Reduce( a, res, 2,
            MPI_FLOAT, MPI_SUM,
            root,
            MPI_COMM_WORLD );

if (rank == root) {
    // Print result
}

P₀
a₁,b₁

P₁
a₂,b₂

P₂
a₃,b₃

P₃
a₄,b₄

Sₐ = a₁ + a₂ + a₃ + a₄
Sₛ = b₁ + b₂ + b₃ + b₄
Exercise: Approximation of Pi

\[ \pi = 4 \times \arctan(1) \]

\[ \approx 4 \times \left( \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots \right) \]

- **Edit** `pi_collect.{c,f90}`
  - Use `MPI_Bcast` to broadcast the number of iterations
  - Use `MPI_Reduce` to compute the final sum
- **Run:**
  
  ```bash
  mpiexec -n 2 ./pi_collect < pi_collect.in
  ```
Exercise: MPI_Wtime

- Edit `pi_collect.{c,f90}` in order to measure elapsed time for computing Pi
  - Only process 0 will compute elapsed time

```c
double t1, t2;
if (rank == 0) {
    t1 = MPI_Wtime();
}
// Computing Pi...
if (rank == 0) {
    t2 = MPI_Wtime();
    printf("Time = %.16f sec\n", t2 - t1);
}
```
**MPI_Scatter and MPI_Gather**

- **MPI_Scatter**: one-to-all communication - different data sent from root process to all others in the communicator, following the rank order.

- **MPI_Gather**: data collected by the root process. Is the opposite of Scatter.
Example

**MPI_Scatter, MPI_Gather**

```c
// Scatter
int i, sc = 2; // sendcount
float a[16], b[2];

if (rank == root) {
    for (i = 0; i < 16; i++) {
        a[i] = i;
    }
}

MPI_Scatter (a, sc, MPI_FLOAT,
             b, sc, MPI_FLOAT,
             root,
             MPI_COMM_WORLD);

// Print rank, b[0] and b[1]
```

```c
// Gather
int i, sc = 2; // sendcount
float a[16], b[2];

b[0] = rank;
b[1] = rank;

MPI_Gather(b, sc, MPI_FLOAT,
           a, sc, MPI_FLOAT,
           root,
           MPI_COMM_WORLD);

if (rank == root) {
    // Print a[0] through a[15]
}
```
Exercise
Dot Product of Two Vectors

• Edit `dot_prod.{c,f90}`
  ○ Dot product = $A^T B = a_1 b_1 + a_2 b_2 + ...$
  ○ 2*8400 integers: $dp = 1*8400 + 2*8399 + ... + 8400*1$
  ○ The root process initializes vectors A and B
  ○ Use `MPI_Scatter` to split A and B
  ○ Use `MPI_Reduce` to compute the final result

• (Optional) Replace `MPI_Reduce` with `MPI_Gather` and let the root process compute the final result
Conclusion
MPI routines we know ...

- **MPI environment**
  - MPI_Init, MPI_Finalize

- **Information on processes**
  - MPI_Comm_rank, MPI_Comm_size

- **Point-to-point communications**
  - MPI_Send, MPI_Recv
  - MPI_Isend, MPI_Wait

- **Collective communications**
  - MPI_Bcast, MPI_Reduce
  - MPI_Scatter, MPI_Gather
Further readings

● The standard itself, news, development:
  ○ http://www.mpi-forum.org

● Online reference book:

● Calcul Quebec's wiki:
  ○ https://wiki.calculquebec.ca/w/MPI/en

● Detailed MPI tutorials:
  ○ http://people.ds.cam.ac.uk/nmm1/MPI/
  ○ http://www.mcs.anl.gov/research/projects/mpi/tutorial/
Questions?

● Calcul Quebec support team:
  ○ support@calculquebec.ca

● Specific site support teams:
  ○ briaree@calculquebec.ca
  ○ colosse@calculquebec.ca
  ○ guillimin@calculquebec.ca
  ○ mammouth@calculquebec.ca