Tutorial 2: MPI

CS486 - Principles of Distributed Computing

Papageorgiou Spyros

What is MPI?

- An Interface Specification
- MPI = **M**essage **P**assing **I**nterface
- Provides a standard -> various implementations
- Offers access to message passing primitives
  - Point to point (send, receive)
  - Collective (broadcast)

- Enable use of MPI features by including **mpi.h**
- Utilities required to run MPI programs: **mpicc** and **mpirun**
- **mpicc** used for compilation
- **mpirun** is used to start MPI processes
    - Useful command line:
      mpirun -np *6* --hostfile *my_hosts test_exec*
    - This will start 6 MPI processes in a round-robin fashion across the hosts specified in the *my_hosts* file. All processes will run *test_exec*.

Basic MPI calls

- MPI_Init(int *argc, char ***argv)
  - Initializes MPI execution environment
  - Must be called before **any** other MPI functions
  - Called only once in a program
  - Can be used to pass cli arguments to other processes
  - Usually called in the form of MPI_Init(NULL, NULL);
- MPI_Finalize( )
  - Terminates MPI execution environment
  - No other MPI routines should be called after MPI_Finalize is called
- MPI_Get_processor_name(processor_name, &name_len)
  - Get the machine name in a buffer

Basic MPI calls

- MPI Communicators
  - MPIs way to organize MPI processes in “channels” of communication
  - Send messages to other processes in the same communicator group
  - By default all MPI processes belong to the same communicator: **MPI_COMM_WORLD**
  - Probably no need to use another communicator for Project 2...

Basic MPI Calls

- MPI_Comm_size(comm, &size)
  - Returns the total number of MPI processes in the specified communicator
  - E.g. MPI_Comm_size(MPI_COMM_WORLD, &num_procs)
  - Number will match the one we provided using the **np** flag when **mpirun** was invoked
- MPI_Comm_rank(comm, &rank)
  - Returns the MPI rank of the calling processes within the communicator
  - Rank ranges from 0 to <np - 1> and is unique for each process within the communicator

Communication Routines

- MPI supports two types of communication operations: **point-to-point** and **collective**
- Point-to-point
  - Involve message passing between only two MPI processes
  - One MPI process performs a **send** operation
  - The other MPI process performs a matching **receive** operation
- Collective
  - Must involve **all** processes within the scope of a communicator (e.g. MPI_COMM_WORLD)
  - Routines involve broadcasts, scatter/gather, barriers etc.

Basic MPI Calls: send

- MPI_Send(&buffer, count, datatype, destination, tag, communicator)
  - Basic blocking send operation (returns after send completes)
  - **buffer:** buffer containing data
  - **count:** number of **datatype** elements being sent
  - **datatype:** type of data being sent. MPI predefines its elementary data types (see tutorial)
  - **destination:** MPI rank of the process intended to receive the message
  - **tag:** non-negative integer to uniquely identify a message. Can use wildcard **MPI_ANY_TAG**.
  - **communicator:** indicates the communication context

Basic MPI Calls: receive

- MPI_Recv(&buffer, count, datatype, source, tag, communicator, &status)
  - Receive a message and block until the requested data is available in the buffer
  - Most fields similar to MPI_Send
  - **source:** indicates originating process of message (it's MPI rank). May be set to wildcard **MPI_ANY_SOURCE**.
  - **status:** indicates the source and tag of the message. Useful if **MPI_ANY_SOURCE/ MPI_ANY_TAG** were used. (status.MPI_SOURCE & status.MPI_TAG)

Basic MPI Calls: Barrier & Broadcast

- MPI_Barrier(communicator)
  - Synchronization operation
  - Creates a barrier synchronization in a group
  - Tasks block when reaching the MPI_Barrier call until all tasks in the group reach the same MPI_Barrier call.
- MPI_Bcast(&buffer, count, datatype, root, communicator)
  - Data movement operation
  - Broadcasts message from process with rank **root** to all other processes in communicator
  - **NOTE:** all processes must call MPI_Bcast => **receiving processes too**!

Creating MPI types

- MPI_Type_contiguous (count, oldtype, &newtype)
  - Simplest method to produce a new data type
  - Simply makes **count** copies of an existing data type
  - e.g.

```
MPI_Datatype event_t;
MPI_Type_contiguous(SIZE, MPI_INT, &event_t);
MPI_Type_commit(&event_t);
// use event as a data type during sends
MPI_Type_free(&event_t);
```