MPI and Modern Network Hardware

Rich Graham
September 2017
The views expressed here reflect my personal views, and not Mellanox’s point of view.
What is MPI?

- Library interface for
  - Moving data
  - Operating on data that is moved
  - Supports commonly used data patterns
  - Provide ancillary services needed to support such capabilities
  - The “glue” of a parallel job

- User-level interface to control such data movement

- Not: programming model
Challenges
Characteristics of a Good User-Level Communication API

- High-level and portable
- Can stand the test of time
- Easy to use, without having to know the details of an underlying hardware or software platform
- Interface objects are implementation neutral
- Provides the ability to pass information to and from the communication library
Goals for a High Performance Communication Library Interface

- Performance
- Performance
- Performance

- Obtains good performance over a wide range of hardware configurations
- Obtains good performance for a wide range of user applications
- Is not aware of application context, unless some sort of hints are passed to it
Characteristics of Emerging Hardware Systems

- Many communication end-points
- Heterogeneous architectures
- Computation can occur at the edges (CPU, Storage) and the interior (Network)
- Rich single process environment
  - Multiple compute engines
  - Large latency differences between memories (some memory may not even be directly addressable)
  - Different compute engines may have exclusive access to some memories (GPU memory, Scratch-Pad memory)

- With all of this need to allow an application to express their communication needs in a simple manner
- Want to be able to use these capabilities in an effective manner
Barriers
Computation on the Fly
Current states: opportunistic
- Collective communication provides a hook for a small number of such operations
  - In the switches
  - In the HCA
  - Network-level coordination
  - ?
- Simple objects can be mapped outside of host memory and opiated on via MPI API’s
  - Atomic updates
- Split address and computational capabilities, such as GPUs
  - Challenge to express communication in a manner that reflects the “split” nature of the MPI process (or is this even the right way to think about it)

Need to think of communication and computation as part of a single operation to be optimized

Need to be able to express
- Communication and work to do on this data
Limited Context for Communication
Current state:
• Some ability to pass information to the library via
  - Info Objects
• No ability to express
  - Object durability, such as
    ▪ One-use data type
    ▪ Collective operations that are used very rarely
  - Capabilities to be used
    ▪ Collective operations
  - Nature of an application
    ▪ Highly unbalanced

Can we do better?
• Is persistence a good example?
Data Path Opacity
Data Buffers

- **MPI_ALLOC_MEM** \(\text{(size, info, baseptr)}\)
  - **IN** size size of memory segment in bytes (non-negative integer)
  - **IN** info info argument (handle)
  - **OUT** baseptr pointer to beginning of memory segment allocated

  - Issue: no output (opaque) meta-data to describe the region

- **User created buffers** (malloc, heap, mmap, …)
  - Issue: no way to associate meta-data with these regions
Point-to-point Send Function

- **MPI_ISEND(buf, count, datatype, dest, tag, comm, request)**
  - IN buf initial address of send buffer (choice)
  - IN count number of elements in send buffer (non-negative integer)
  - IN datatype datatype of each send buffer element (handle)
  - IN dest rank of destination (integer)
  - IN tag message tag (integer)
  - IN comm communicator (handle)
  - OUT request communication request (handle)

  - Issue: no way to pass in/out “buf” meta-data
Collective Function

- **MPI_ALLREDUCE**(sendbuf, recvbuf, count, datatype, op, comm)
  - IN sendbuf starting address of send buffer (choice)
  - OUT recvbuf starting address of receive buffer (choice)
  - IN count number of elements in send buffer (non-negative integer)
  - IN datatype data type of elements of send buffer (handle)
  - IN op operation (handle)
  - IN comm communicator (handle)

- Issue: no way to pass in/out meta-data information on buffers
Consequences

- Communication libraries do create meta-data for tracking communication
- Buffer meta-data is looked up for each access in the data path

Possible solutions:
  - Rely on hardware-level On-Demand-Paging to setup memory for communication, if not ready (first access can be very expensive)
  - Enhance interfaces to allow for opaque meta-data to be passed between MPI functions
Limited Pattern Expresiveness
Communication Patterns

- **Supported**
  - Point-to-point
  - Collective

- **Missing**
  - Send: one to several (not full communicator)
  - Send: several one-to-ones
  - Receive: One/some of several
  - Receive: Eureka
  - ????
Thank You