Early Experiences with MPICH over the Intel KNC Architecture

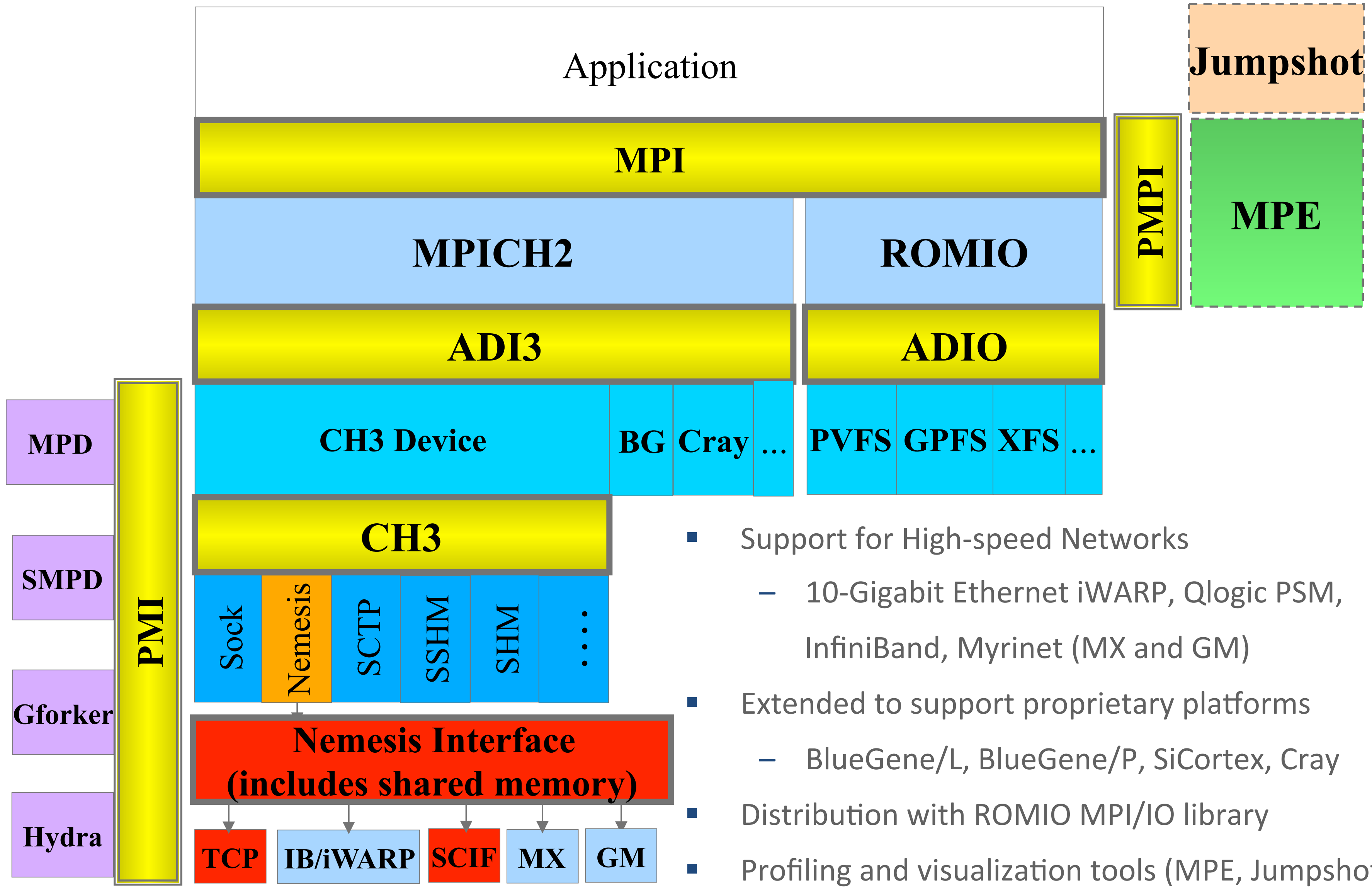
Pavan Balaji

Argonne National Laboratory

balaji@mcs.anl.gov

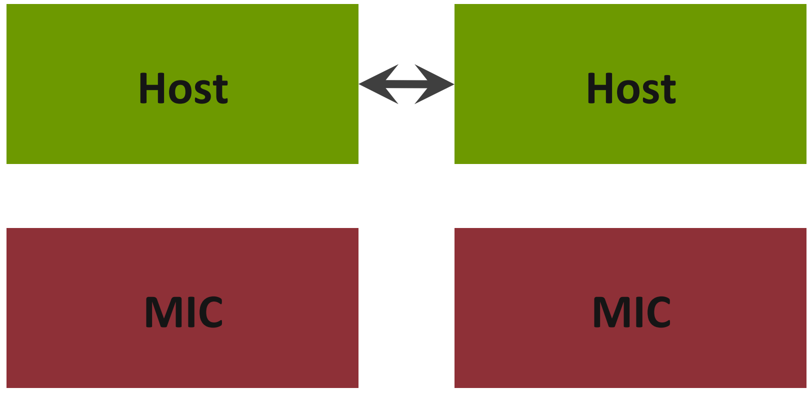
MPICH Architecture

- Implementation of MPI-2.2 (and soon MPI-3.0)
- Portable architecture that works on virtually every supercomputer
  - 6 of the top 10 supercomputers use MPICH2 derivatives exclusively
  - Remaining 4 use MPICH2 derivatives in conjunction with other MPI implementations
- Argonne develops the vanilla MPICH code (currently 1.4.x/1.5.x)
  - MPICH philosophy: fork and tune!
    - Partners provide platform-specific tuned versions of MPICH
  - Intel MPI (partner since the early stages of the MPICH project)
    - Recent code contributor – we are working on integrating several changes into the next MPICH2 release
  - IBM MPI
    - Contributor for many years – is working on integrating MPICH2 as the unified MPI implementation for all their platforms
  - Cray MPI, Microsoft, MVAPICH, many others …

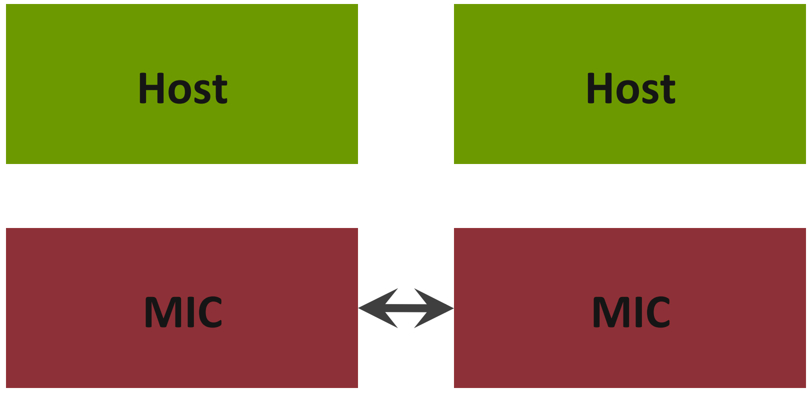


- Support for High-speed Networks
  - 10-Gigabit Ethernet iWARP, Qlogic PSM, InfiniBand, Myrinet (MX and GM)
- Extended to support proprietary platforms
  - BlueGene/L, BlueGene/P, SiCortex, Cray
- Distribution with ROMIO MPI/IO library
- Profiling and visualization tools (MPE, Jumpshot)

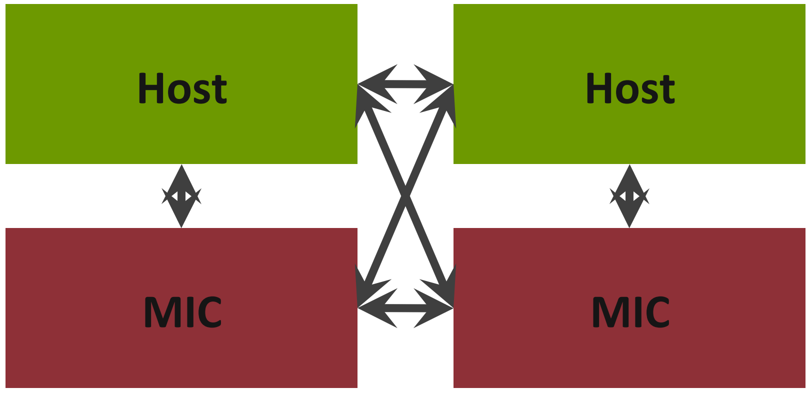
Models for using MIC



Offloaded



Many-core hosted



Symmetric

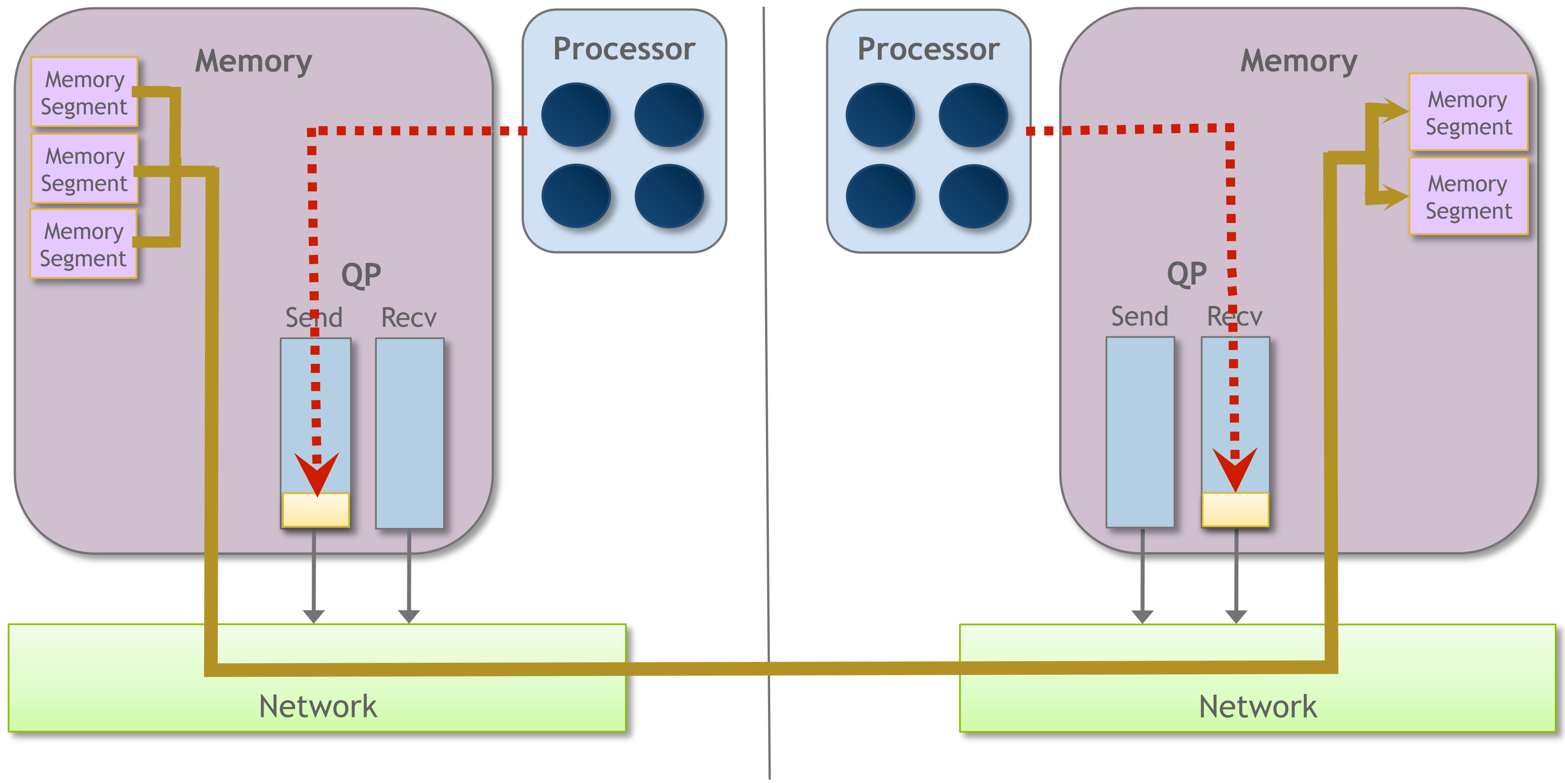
MPICH on KNC

- *Shared code development between Argonne and Intel*
- We plan on having support for three levels of capabilities
  - Shared memory support (supported in MPICH >= 1.5a2)
    - Only useful for communicating on the MIC
    - Gives us the best case performance that can be achieved
    - Read/write memory barriers are full memory barriers
  - TCP/IP sockets based MIC-to-host and MIC-to-MIC communication
    - Can be used within the node and outside (using the Virtual Ethernet driver from Intel)
    - Already working (except Fortran bindings; should be easy to add)
  - SCIF based MIC-to-host and MIC-to-MIC communication
    - Early implementation in place
      - Not best performing (or stable)
    - In discussion with Intel to see if some things can be improved

MPICH over SCIF

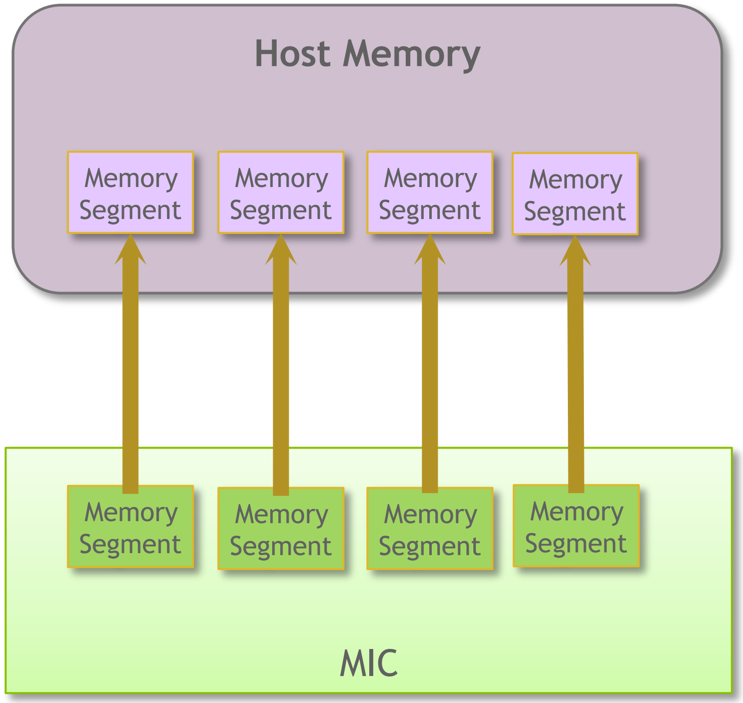
- Current code utilizes both SCIF send/recv and SCIF put/get
  - SCIF send/recv internally performs an extra copy
    - Used for small messages
    - MPI needs an additional copy as well for matching purposes
      - This leads to two copies on the receiver and 1-2 copies on the sender (total 3-4 copies)
  - Larger messages use DMA transfer, but …
    - The MIC only seems to support one-sided communication
      - The MIC tells the DMA engine where to move data to
      - SCIF send/recv internally utilizes a one-sided model for moving data
        - » This is a scalability concern

Traditional Networks vs. MIC-to-Host Communication

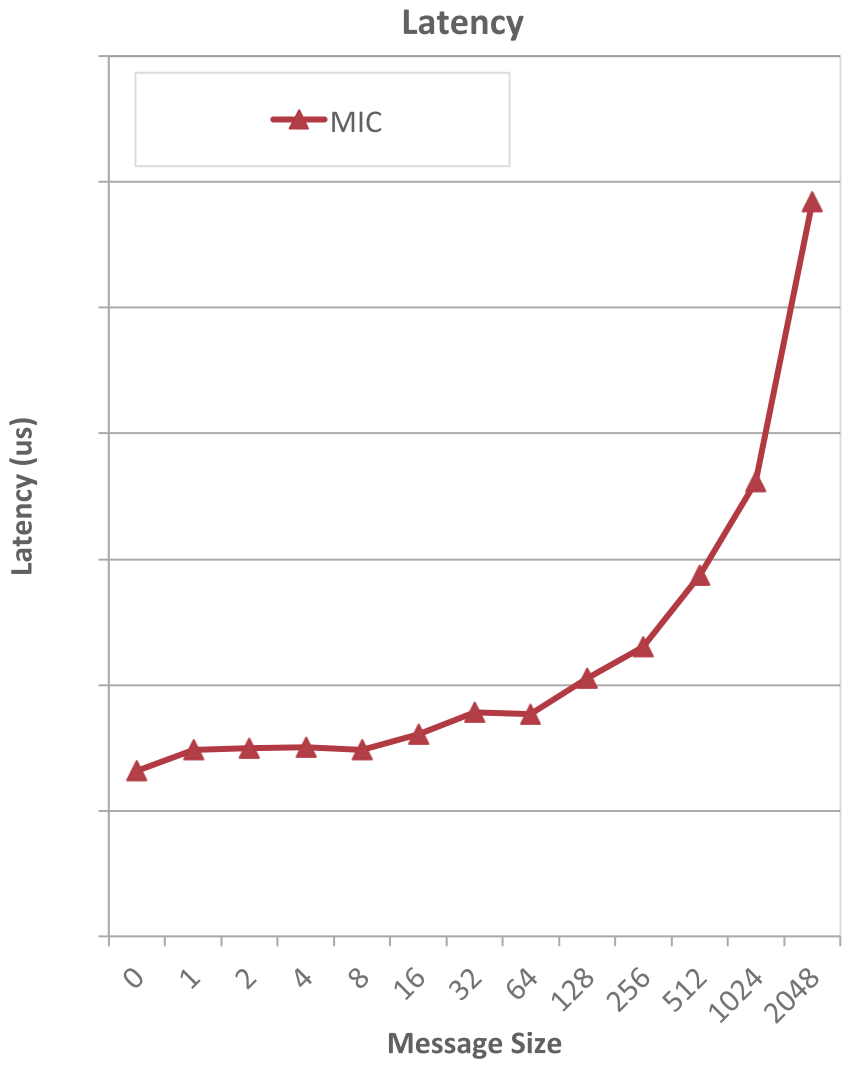


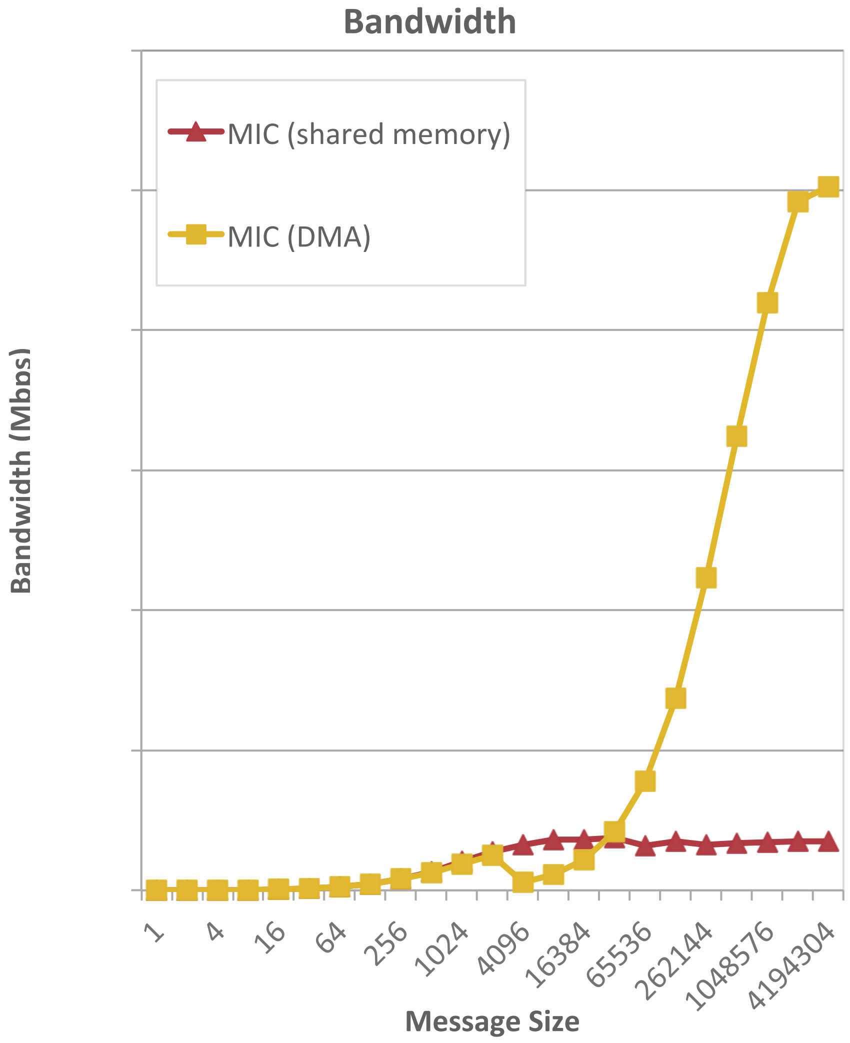
MIC-to-Host Communication Model

- All communication has to be one-sided because there is only one DMA engine (there are no separate sender and receiver DMA engines)
- For one-sided communication, the origin process (residing on the MIC in this figure) has to know the buffer to put data into at the target process (residing on the host in this figure)
- SCIF send/recv works around this by allocating a persistent set of internal buffers and performing PUTs within these set of buffers

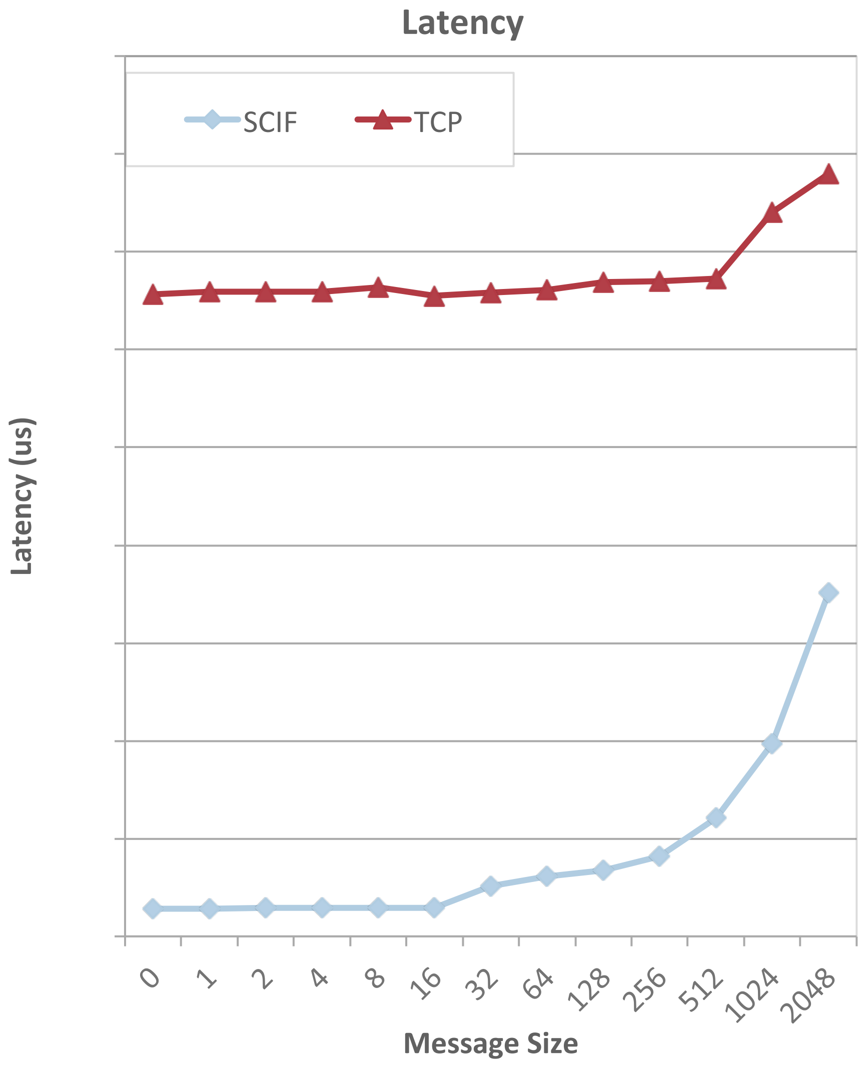


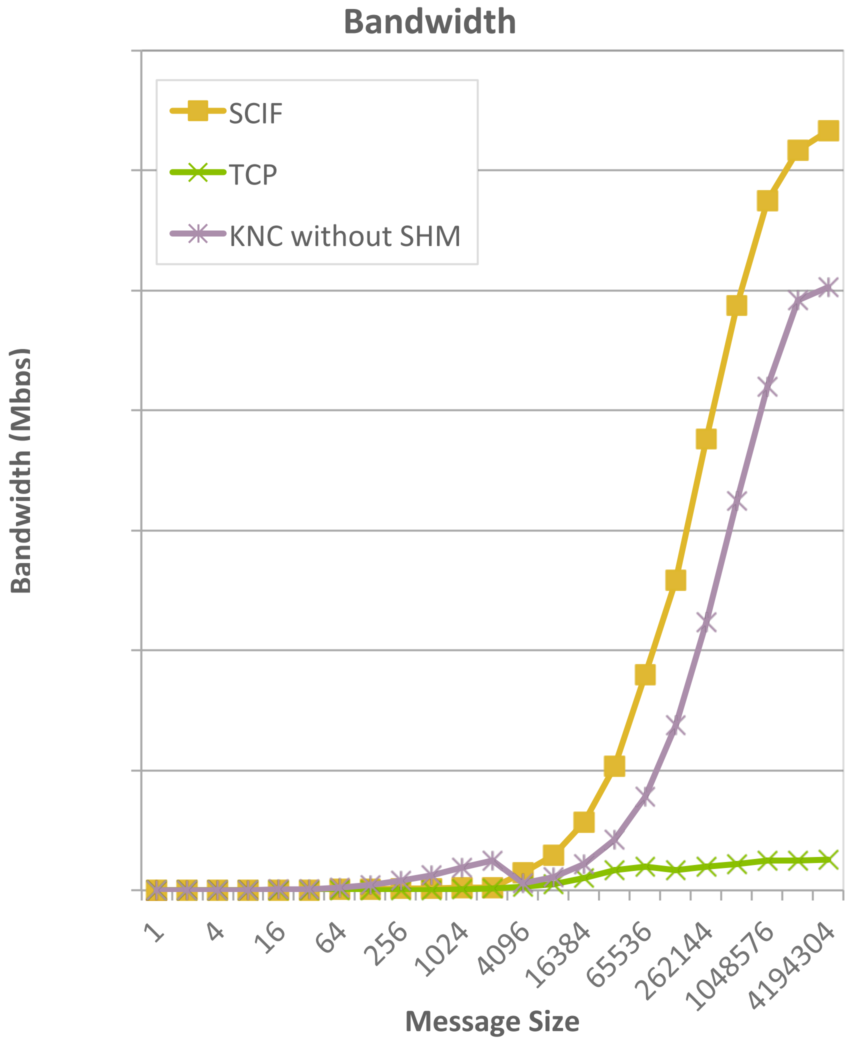
Shared Memory Communication



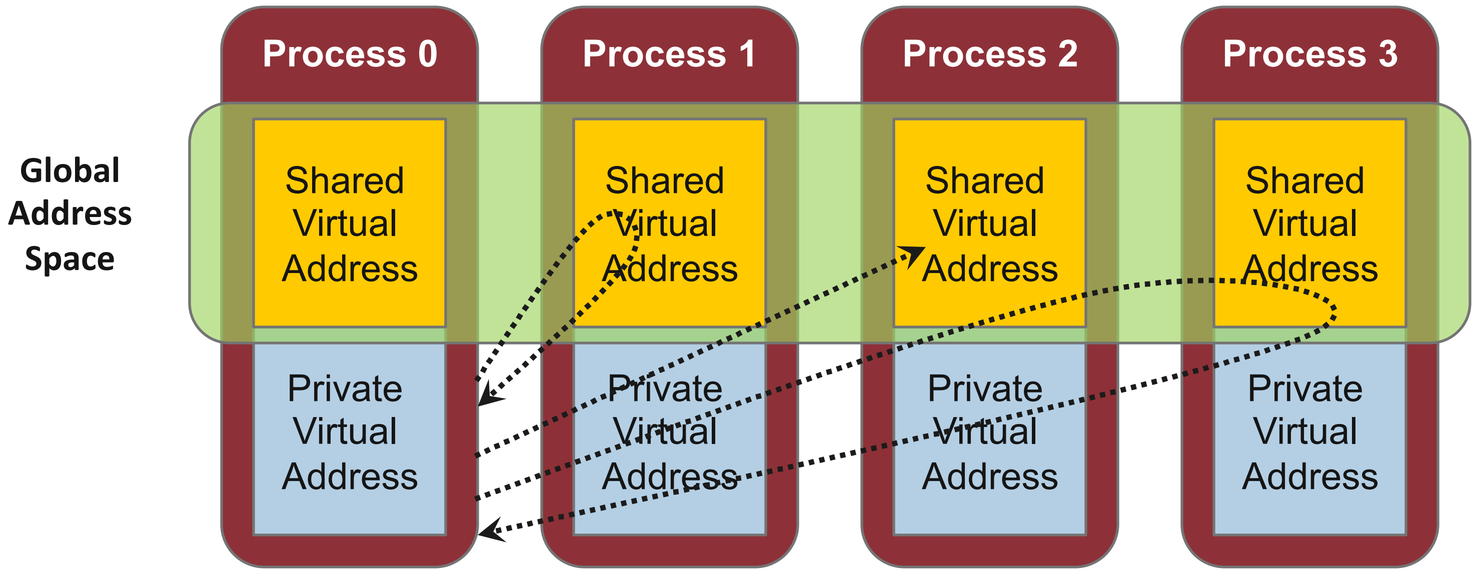


MIC-to-Host Communication





A Short Live Demo



- A more asynchronous application model for MICs
  - Model is that every process exposes a “public buffer” that other processes have access to through PUT/GET
  - Each process gets a part of the global public data, computes on it and puts it back
  - Application model is inherently asynchronous – faster processes do more work, slower processes do lesser work
- Demo will show the computation completed on each process