A Preview of MPI 3.0: The Shape of Things to Come

20 Years of Excellence in Computational Science
OLCF
Oak Ridge Leadership Computing Facility
1992–2012

Manjunath Gorentla Venkata
manjugv@ornl.gov

Joshua Hursey
hurseyjj@ornl.gov

Overview of Seminar Series

- **Monday, June 25 - 3-4 pm**
  - MPI Process (brief)
  - Timeline to 3.0
  - MPI 3.0 Fortran Bindings
  - MPI 2.2
- **Tuesday, June 26 - 3-4 pm**
  - Collectives in MPI 3.0:
    - Neighborhood
    - Nonblocking
  - Communicator Creation:
    - Noncollective
    - Nonblocking duplication
- **Thursday, June 28 - 3-4 pm**
  - MPI Matched Probe/Recv
  - RMA / One-sided enhancements
  - Tool Interfaces
  - MPI <next>
    - Fault Tolerance
    - Hybrid, collectives, …

MPI Topology and Collectives Support

- Topology Functions (MPI 2.1)
  - Create a Graph or Cartesian topology and query it, nothing else
  - Each rank specifies full graph
- Scalable Graph topology (MPI-2.2)
  - Each rank specifies a subset of the Graph

MPI Topology and Collectives Support

- Neighborhood Collectives (MPI-3.0)
  - Communication functions on the neighbors of the topology (Cartesian, Graph, Distributed Graph)
  - All processes in the communicator call the collective, but communication only along the edges of process topology (neighbors)

- Topology and Neighborhood Collectives

  Users can define a communication topology and perform communication between neighbors in this topology

Need for Neighborhood Collectives

- Many applications and libraries exhibit sparse communication patterns
  - Example: Weather prediction applications, PETSc
- Many architectures support sparse communication efficiently
  - Cray XE/XK node has six neighbors
- Implementation complexity can be reduced if sparse communication is abstracted by libraries

Hoefler et al. : Implemention and Performance Analysis of Non-Blocking Collective Operations for MPI

MPI_NEIGHBOR_ALLGATHER

MPI_Neighbor_allgather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)

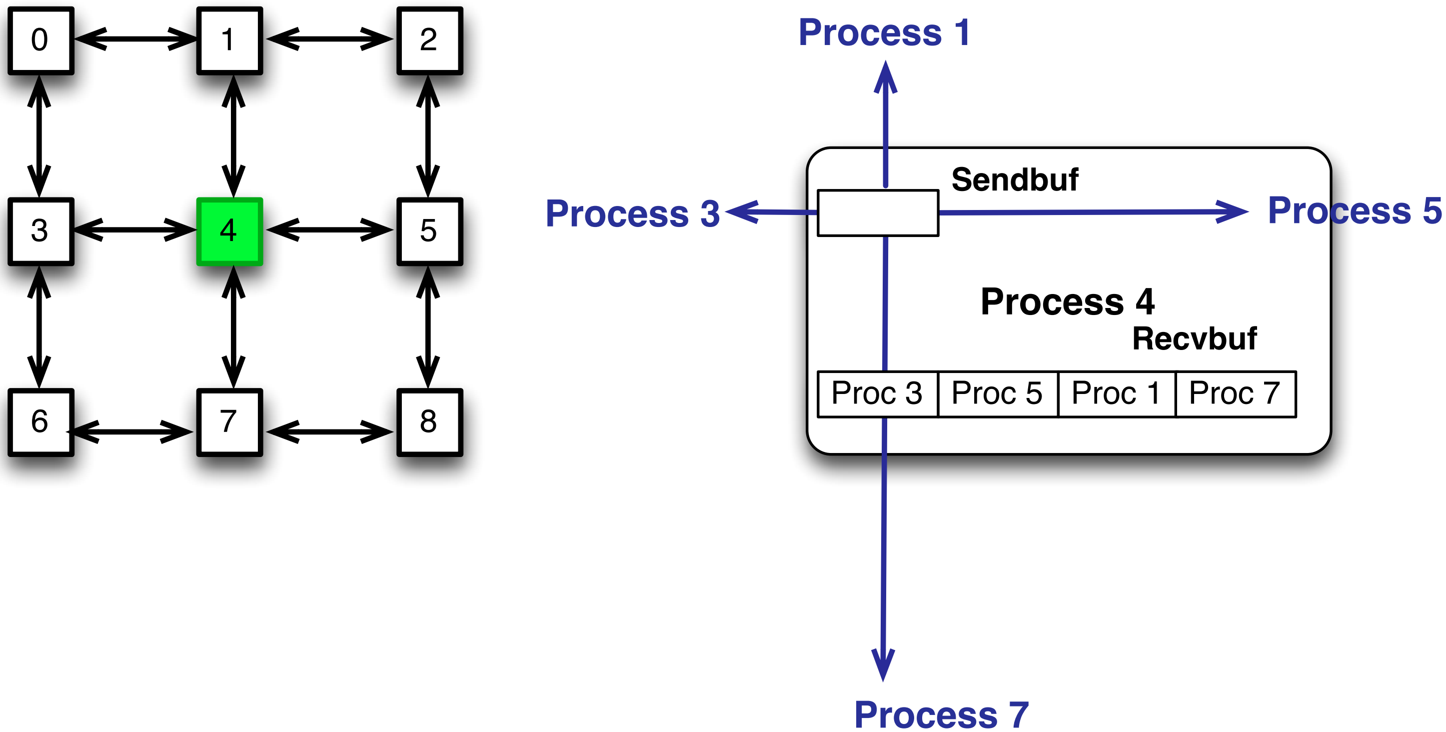
- Send same data element to all neighbor processes
- Receive a distinct data element from each of the neighbor
- Signature of sendtype and recvtype must be same at the corresponding processes
- Order determined by MPI_(Dist)Graph_Neighors
- V version of the call is valid

Thanks to Torsten Hoefler (UIUC) and Martin Schulz (LLNL)

Neighborhood Collectives (Cartesian Communicator)

- Communication between nearest neighbors
  - All processes in the communicator are required to call the collective
  - Number of sources and destinations are equal to 2 * num dimensions
  - The order of neighbors in buffers is in dimension order, and in each dimension first negative neighbor, and then positive neighbor

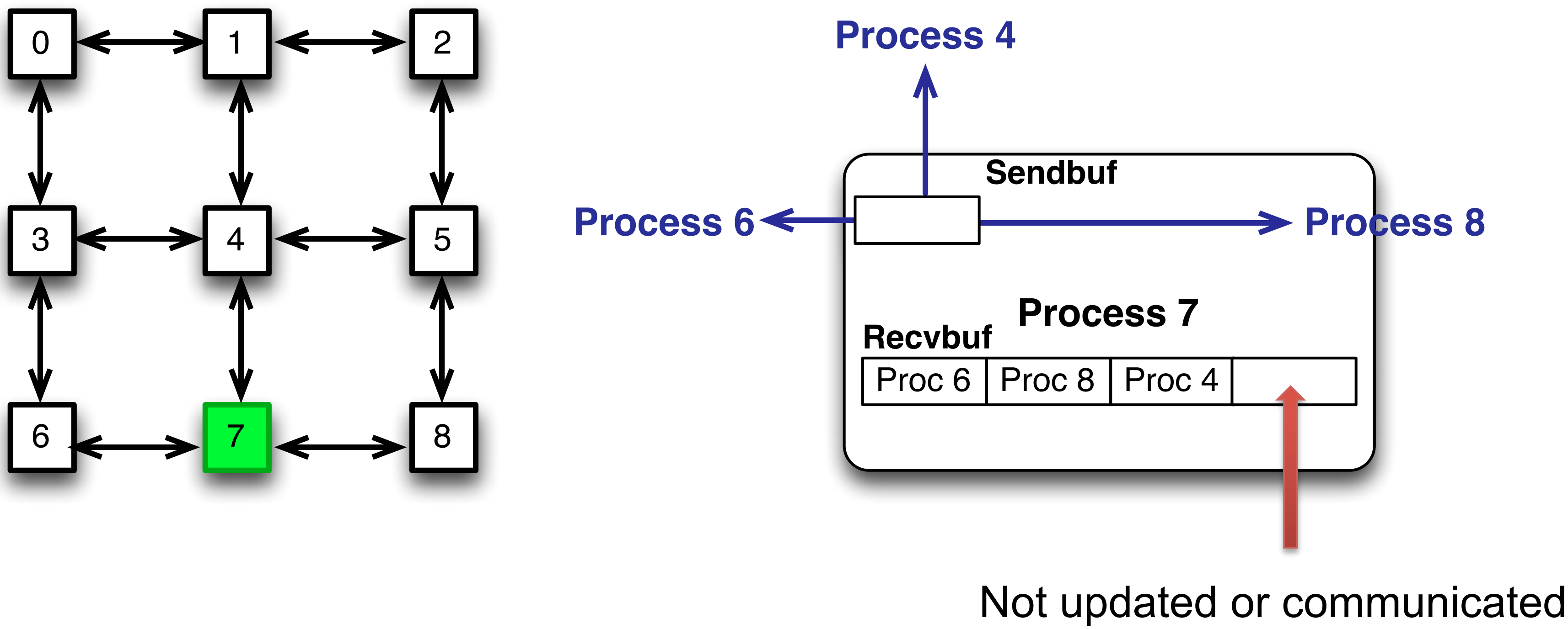
MPI_NEIGHBOR_ALLGATHER (Cartesian Communicator)



- Buffer order: In dimension order, first negative, and then positive

Thanks to Torsten Hoefler (UIUC) and Martin Schulz (LLNL)

MPI_NEIGHBOR_ALLGATHER (Cartesian Communicator)

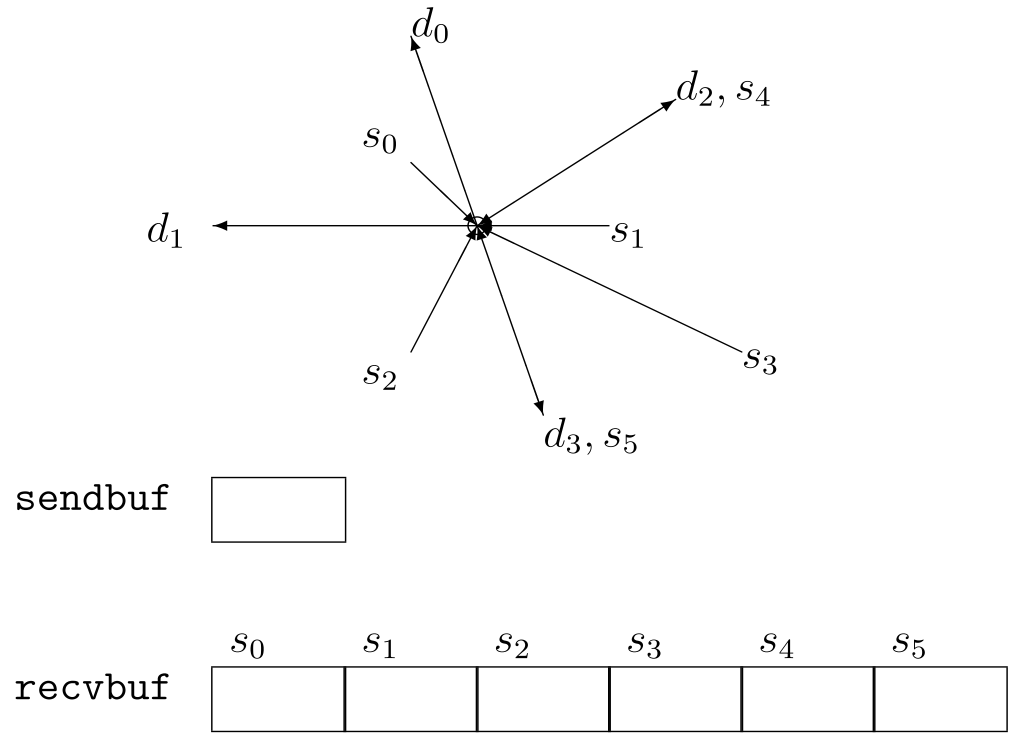


Neighborhood Collectives
(Dist Graph or Graph Communicator)

- Communication between arbitrary neighbors
  - All processes should call the collective
  - Order is determined by MPI_{Dist}Graph_Neighbors call

Equivalent to regular collectives, when each process creates graph treating all processes in the communicator as neighbors

MPI_NEIGHBOR_ALLGATHER (Dist Graph Communicator)



- Between two processes, it sends and receives the same amount of data
- MPI_IN_PLACE is not meaningful

Thanks to Torsten Hoefler (UIUC)

MPI_NEIGHBOR_ALLTOALL

MPI_Neighbor_alltoall(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)

- Send a distinct data element to all neighbor process
- Receive a distinct data element from each of the neighbor
- Type signature of sendtype and recvtype must be same at the corresponding processes
- Order determined by MPI_(Dist)Graph_Neighors
- V and W versions of the call is valid

Thanks to Torsten Hoefler (UIUC) and Martin Schulz (LLNL)

Neighborhood Collectives Summary

- Scalable Graph Topology Creation
- Neighborhood Collectives
  - MPI_Neighbor_Allgather{v}
  - MPI_Neighbor_Alltoall{v,w}
- Neighborhood Collectives (Cartesian Communicator)
- Neighborhood Collectives (Graph Communicator)

Nonblocking Collectives

- Collectives: A global synchronization, data communication, or a reduction operation
- Blocking Collectives: Returns when completed
- Nonblocking Collectives: Splits the invocation and completion of an operation
  - Properties
    - Synchronization decoupled from invocation
    - Enables asynchronous progress (not guaranteed)
    - Multiple outstanding operations
    - Out of order completion

Thanks to Torsten Hoefler (UIUC) and Martin Schulz (LLNL)

Nonblocking Collective Routines in MPI 3.0

MPI_IBARRIER
MPI_IBCAST
MPI_IGATHER
MPI_IGATHERV
MPI_ISCATTER
MPI_ISCATTERV
MPI_IALLGATHER
MPI_IALLGATHERV
MPI_IALLTOALL
MPI_IALLTOALLV
MPI_IREDUCE_LOCAL
MPI_IALLTOALLW
MPI_IREDUCE
MPI_IALLREDUCE
MPI_IREDUCE SCATTER
MPI_ISCAN
MPI_IEXSCAN
MPI_INEIGHBOR_ALLGATHER
MPI_INEIGHBOR_ALLGATHERV
MPI_INEIGHBOR_ALLTOALL
MPI_INEIGHBOR_ALLTOALLV
MPI_IREDUCE_SCATTER_BLOCK

Nonblocking Collectives Semantics

- Multiple nonblocking collectives can be outstanding and their progress is independent

```
MPI_Request req1, req2;

MPI_Ialltoall(sbuf, scnt, stype, rbuf, rcnt, rtype, comm, &req1);
MPI_Ialltoall(sbuf, scnt, stype, rbuf, rcnt, rtype, comm, &req2);
MPI_Wait(&req2, MPI_STATUS_IGNORE);
MPI_Wait(&req1, MPI_STATUS_IGNORE);
```

https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/109

Nonblocking Collectives Semantics

- Blocking and nonblocking collectives can be interleaved

```
MPI_Request req;

MPI_Ialltoall(sbuf, scnt, stype, rbuf, rcnt, rtype, comm, &req);
MPI_Bcast(rbuf, rcnt, type, 0, comm);
MPI_Wait(&req1, MPI_STATUS_IGNORE);
```

https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/109

Nonblocking Collectives Semantics

- Order of nonblocking collectives on a communicator should be the same

```
switch(rank) {
    case 0:
    MPI_Ibcast(buf, count, type, 0, comm, &req);
    MPI_Barrier(comm);
    MPI_Wait(&req, MPI_STATUS_IGNORE);
    break;

    case 1:
    MPI_Barrier(comm);
    MPI_Ibcast(buf, count, type, 0, comm, &req);
    MPI_Wait(&req, MPI_STATUS_IGNORE);
    break;
}
```

https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/109

Nonblocking Collectives Semantics

- Matching of blocking and nonblocking collectives are invalid

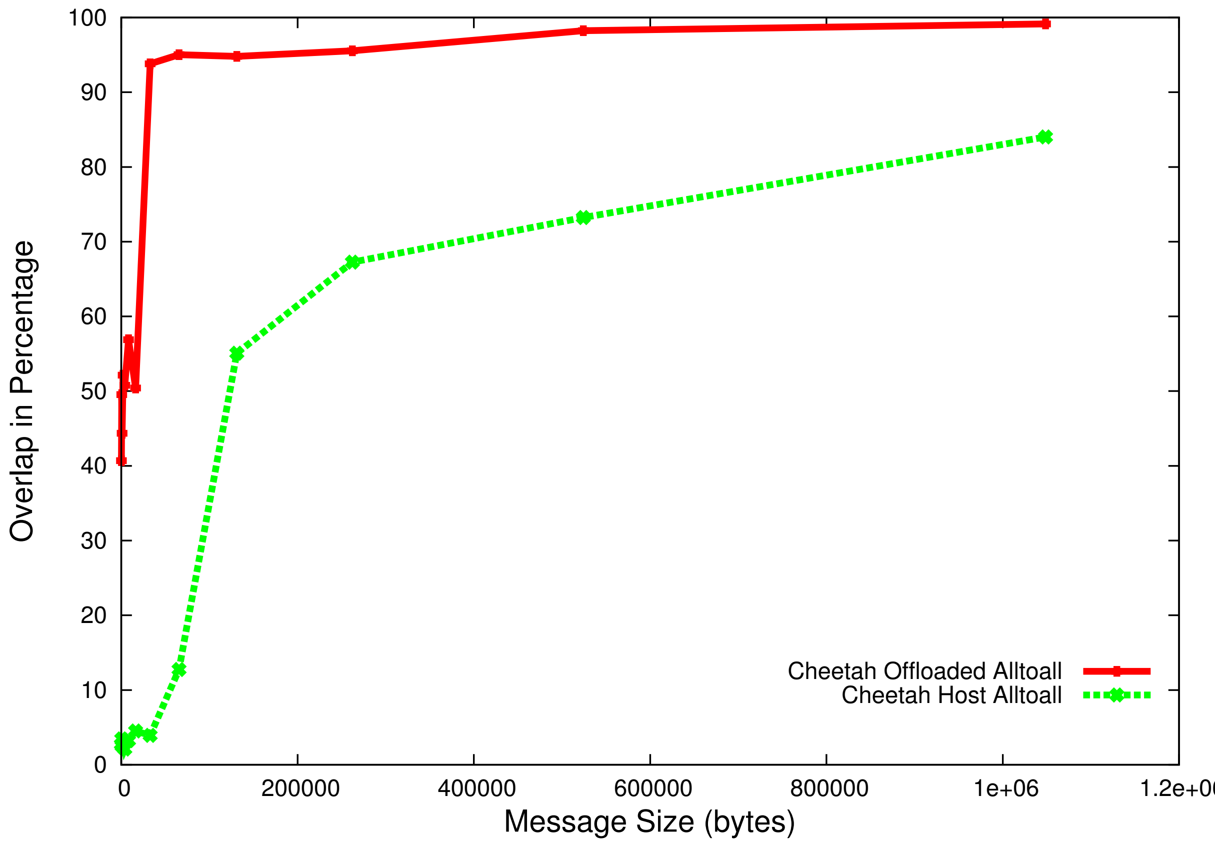
```
switch (rank) {
     case 0:
     MPI_Ibcast(buf, count, type, 0, comm, &req);
     MPI_Wait(&req, MPI_STATUS_IGNORE);
     break;

     case 1:
     MPI_Bcast(buf, count, type, 0, comm);
     break;
}
```

https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/109

Nonblocking Collectives Advantages

- Communication – Computation Overlap
- Noise Resiliency
- Asynchronous Progress
- Multiple Outstanding Operations

Nonblocking Collectives Provides Better Computation-Communication Overlap



- 64-process MPI_Ialltoall and progress examined with MPI_Test
- With network interface offload support one can achieve close to 100% overlap

Gorentla et al. : Exploring the All-To-All Collective Optimization Space with ConnectX CORE-Direct

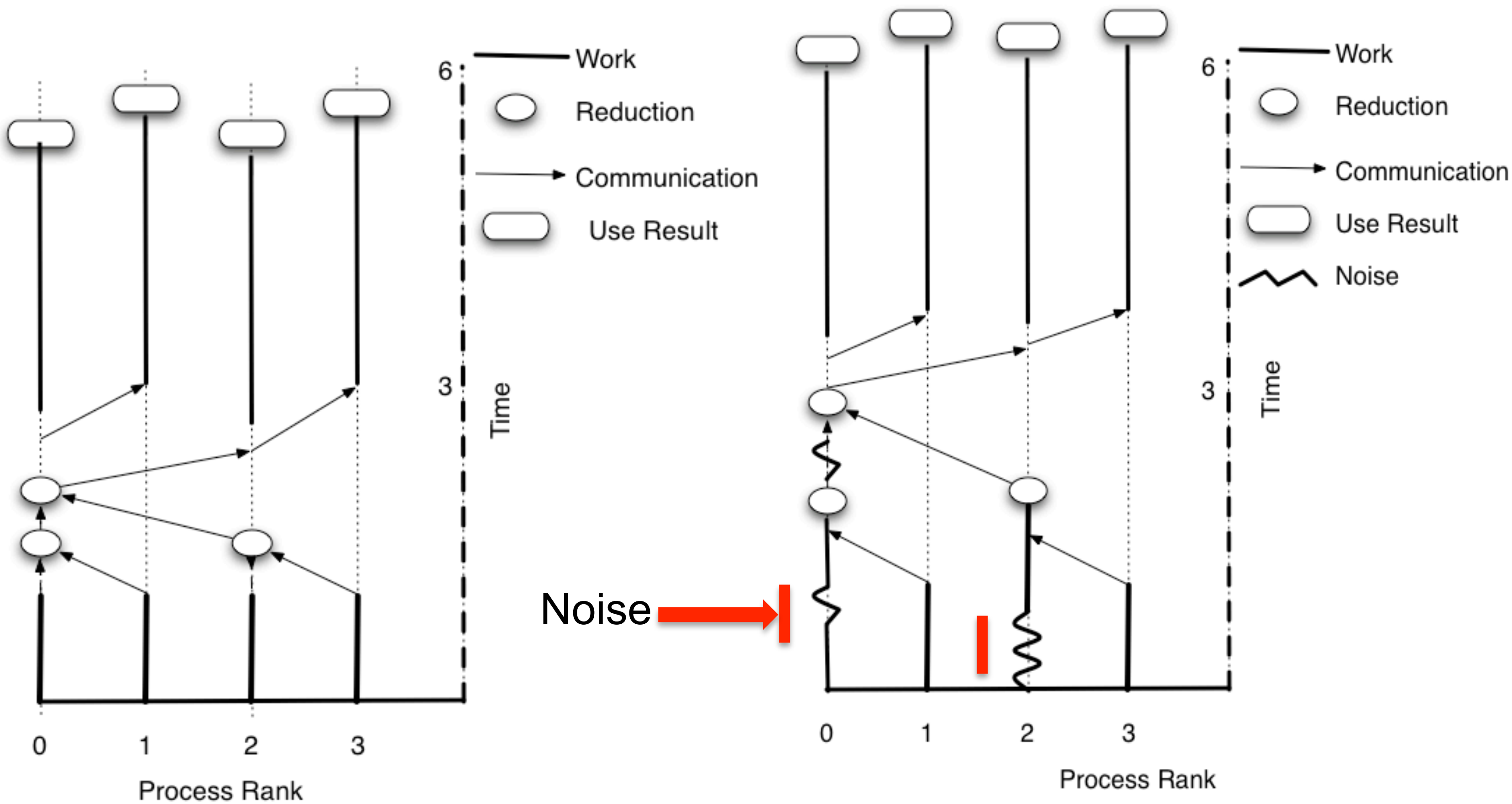
System Noise

- Noise: OS related activity that steals CPU from the application
  - Timer tick
  - Hardware Interrupts
  - Kernel Daemons

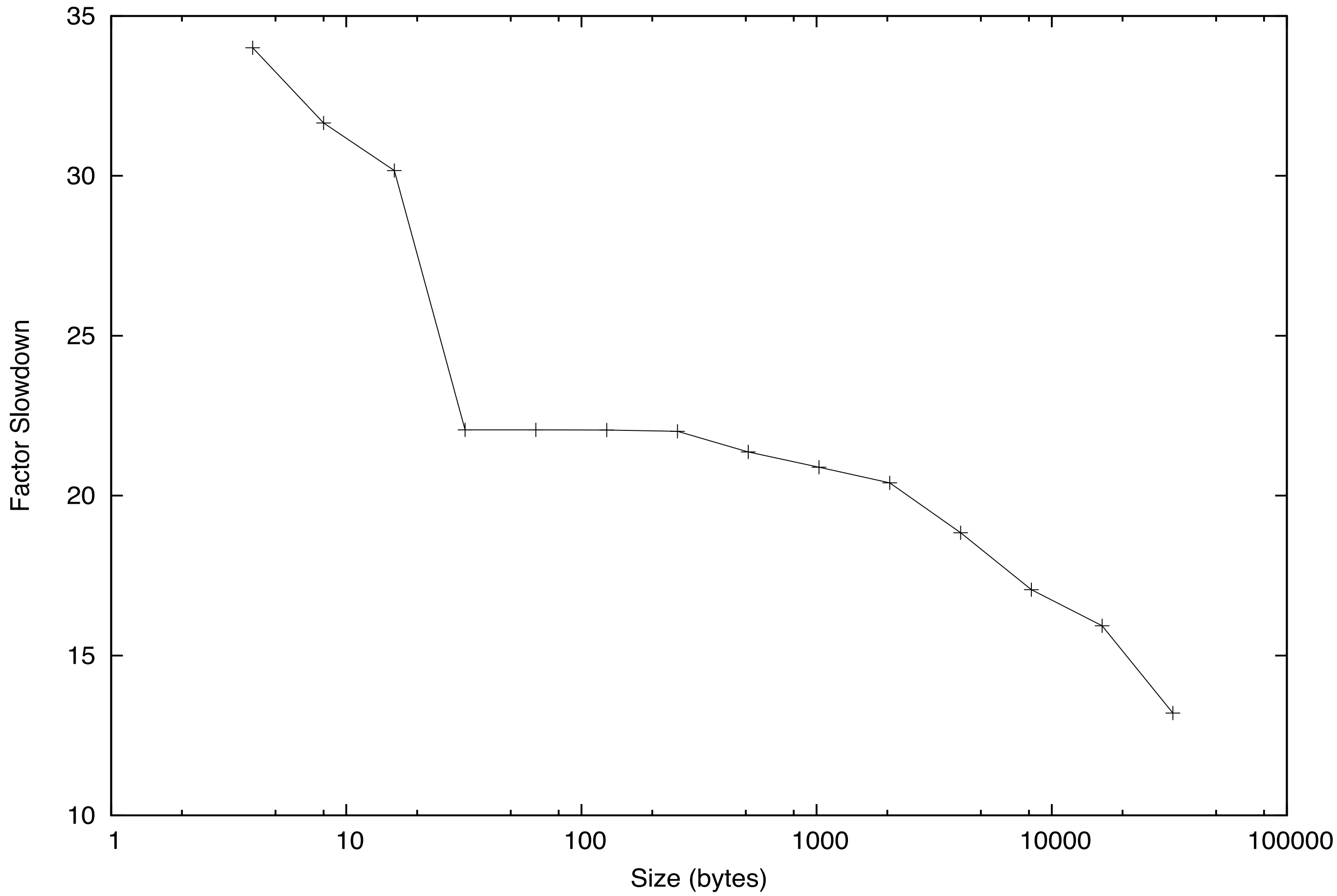
Collective (Global) Performance Cost of System Noise

No Noise

System Noise



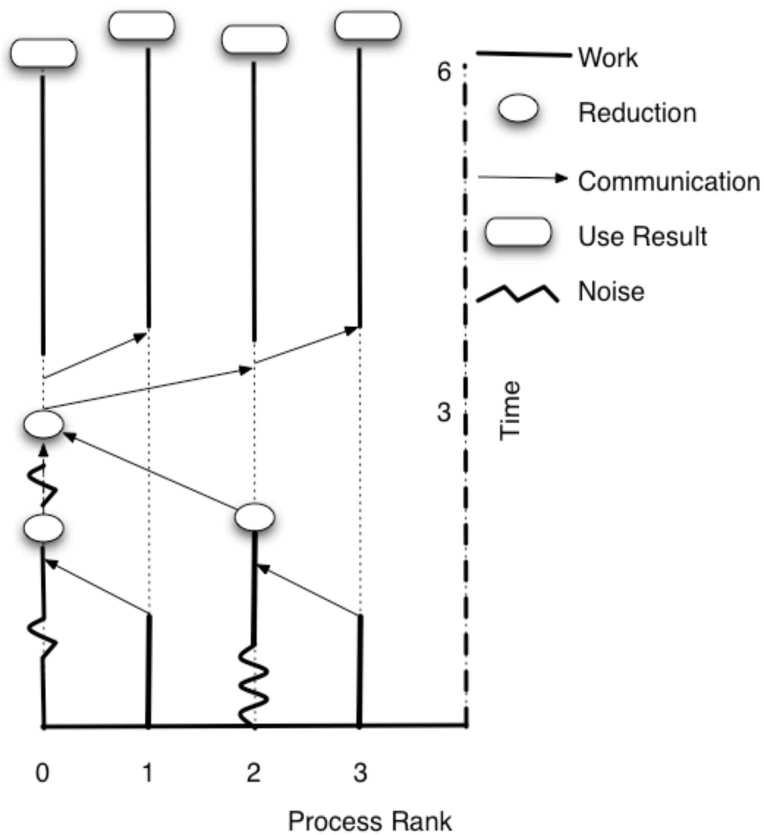
Impact of System Noise on MPI_Allreduce



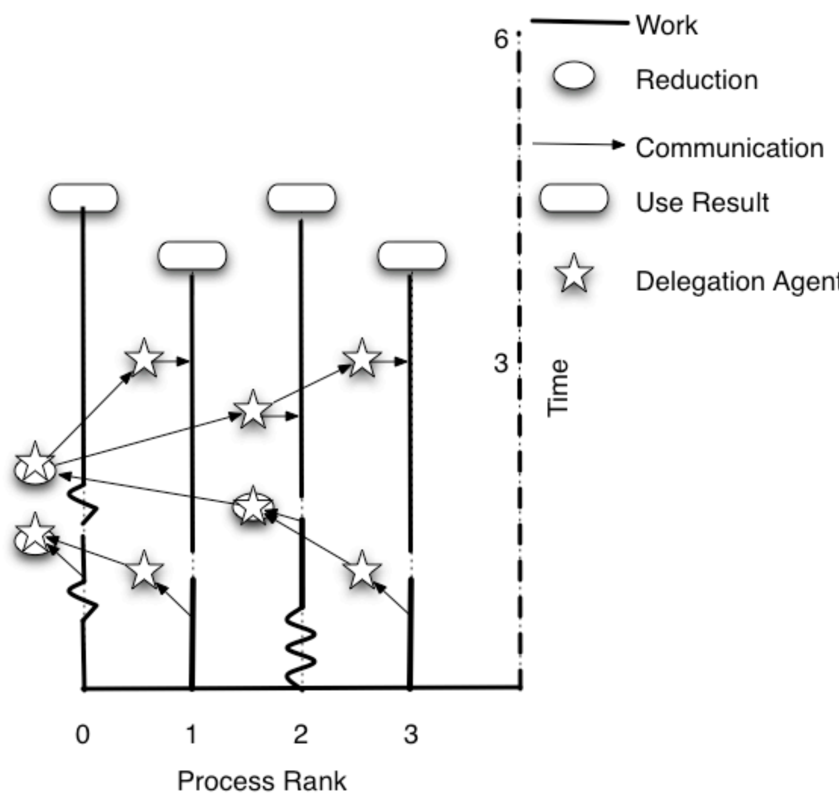
Ferreira et al. : The Impact of System Design Parameters on Application Noise Sensitivity

Nonblocking Collectives Resilient to System Noise Effects

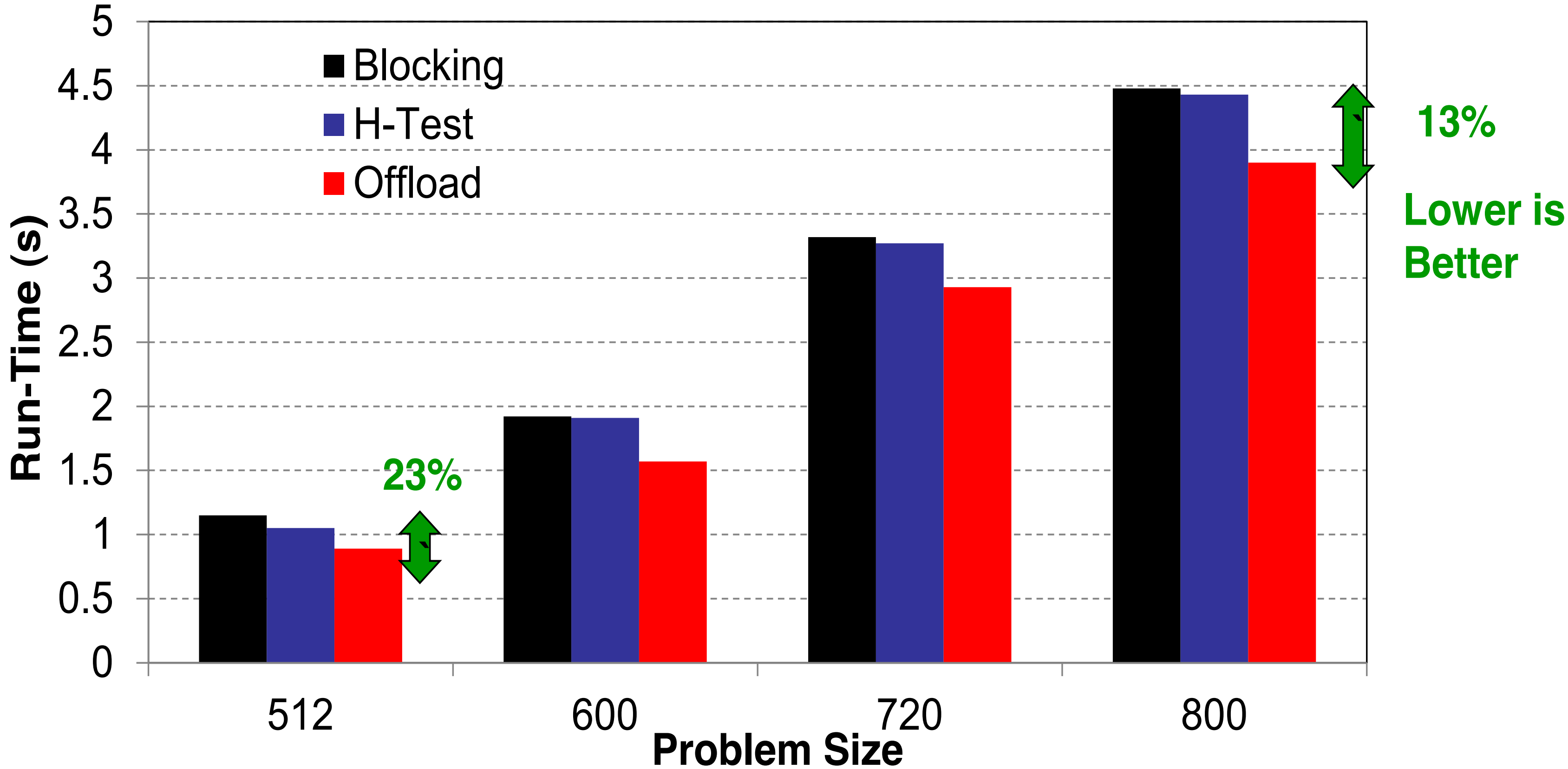
Blocking Collective



Nonblocking Collective



Nonblocking Collectives: Impact on Parallel 3D FFT Kernel Performance



K. Kandalla et al. : High-Performance and Scalable Non-Blocking All-to-All with Collective Offload on InfiniBand Clusters: A Study with Parallel 3D FFT

Nonblocking Collectives Summary

- Nonblocking Collectives Semantics
- Nonblocking Collectives Advantages
  - Communication-Computation Overlap
  - Noise resiliency
- Nonblocking Performance Results

Noncollective Communicator Creation

```
MPI_Group_comm_create(MPI_Comm in, MPI_Group grp, int tag, MPI_Comm *out)
```

- grp is a sub-group of communicator (in)
- No cached information passes from old communicator to the new one

- *Create a communicator with less processes – good for fault tolerance, scalability*

Thanks to Torsten Hoefler (UIUC), Martin Schulz (LLNL), and James Dinan (ANL)

Nonblocking Communicator Duplication Function

MPI_Comm_idup(MPI_Comm comm, MPI_Comm *newcomm, MPI_Request *request)

- Duplicates communicator without blocking
  - Provides a way to overlap communicator creation with other computation
  - Semantics
    - Restrictions and assumptions of nonblocking collectives apply here
    - Error to use newcomm before completion of MPI_Comm_idup creation
    - Attributes changed after MPI_Comm_idup called is not copied to new communicator

Thanks to Torsten Hoefler (UIUC) and Martin Schulz (LLNL)

Implementation Status

	Open MPI	MPICH2
Nonblocking Collectives	Supports Partially (limited release)	Supports
Neighborhood Collectives	No Support	No Support
Nonblocking Communicator Duplicate	No Support	Supports
Noncollective Communicator Create	No Support	Supports

Acknowledgements

- Center for Computational Sciences
- MPI Forum
- Torsten Hoefler (UIUC) and Martin Schulz (LLNL)