Design and Implementation of a Portals Collective Communication Library

Jim Schutt **Ron Brightwell**

Sandia National Laboratories

Center for Computation, Computers, Information, and Mathematics

Cray User Group Meeting

May 10, 2007

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

Outline

- Motivation
- Implementation
- Initial performance results
- Future work

Motivation

- High-performance, scalable collectives for Red Storm
- Need collectives for more than just MPI
  - SHMEM
  - ARMCI
  - Open MPI
- Extending the Portals API for native collectives
  - Collective communication at the network level
  - Building blocks versus complete functionality
- Research several areas
  - Topology/route-aware collectives
  - Non-blocking collectives
  - One-sided collective operations
- Provide a mechanism for collective communication research

Approach

- Start with Puma collective library from ASCI Red
  - Based on InterComm library from van de Geijn
- Port from Portals 2.0 to Portals 3.3

Goals

- Implement asynchronous collective calls
- Allow for arbitrary groups of possibly independent processes
- Minimize use of tuning parameters that affect robustness
- Match or exceed performance and scalability of existing collective implementations

Algorithms

- All algorithms based on a minimum spanning tree
- Broadcast recursively halves network
  - Reduce uses inverse operation
- Scatter recursively halves network and data
  - Gather uses inverse operation
- Barrier interleaves 0-length reduce and broadcast operations
- Allows for implementing Allreduce and Allgather with the same message pattern
  - Scaling will likely be sub-optimal for larger messages

Protocols

- Each rank
  - Keeps a receive heap per group dedicated to eager sends
  - Conservatively tracks peers' eager send heap use
  - Uses eager send for any message for which peer has sufficient send heap space
  - Can use both eager and rendezvous in the same collective
- Some sequences operations have send-only or receive-only ranks
  - May race ahead and deplete resources
  - We detect such sequences and limit periodically by forcing a rendezvous operation
  - Largely a micro-benchmark issue

Implementation

- Single Portal event queue for all operations for all groups
- Eager send heap is double-buffered
- Role of each message completely encoded in match bits
- Each operation creates a state object to track progress
- Asynchronous operations
  - Return handle identifying state object that is used to poll for completion
  - Currently have to poll in order to make progress

Lifetime of a Collective Operation

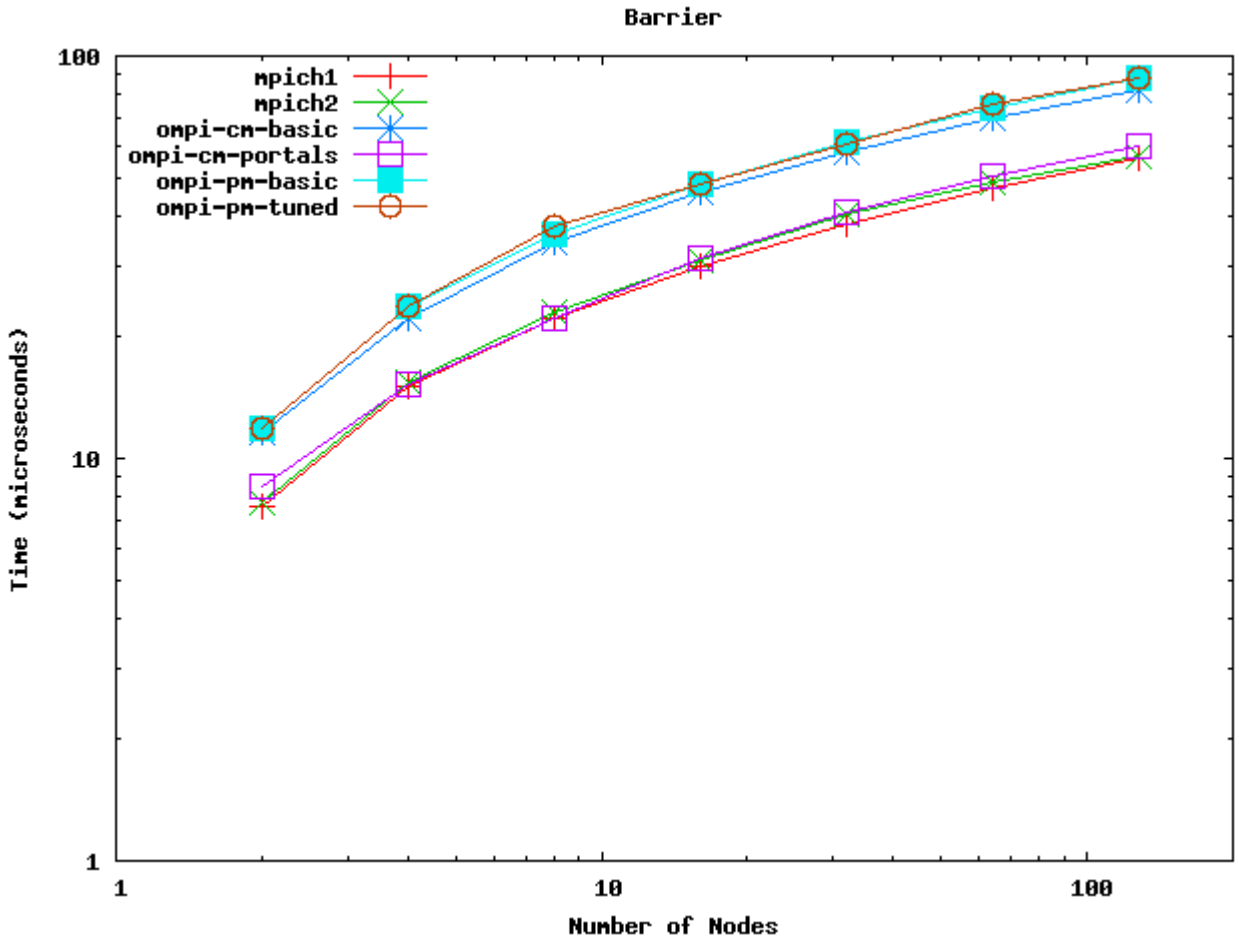
- Create a state object
  - Compute list of messages
  - Assign eager/rendezvous protocol to each message
- Process messages
  - Send
    - If eager, do send
    - If rendezvous, poll event queue until RTR arrives
  - Receive
    - If eager, poll event queue for message
    - if rendezvous, send RTR to peer, then poll event queue
    - Perform reduce operation if necessary
- If asynchronous, return whenever event queue is empty
- Cache events not related to current operation

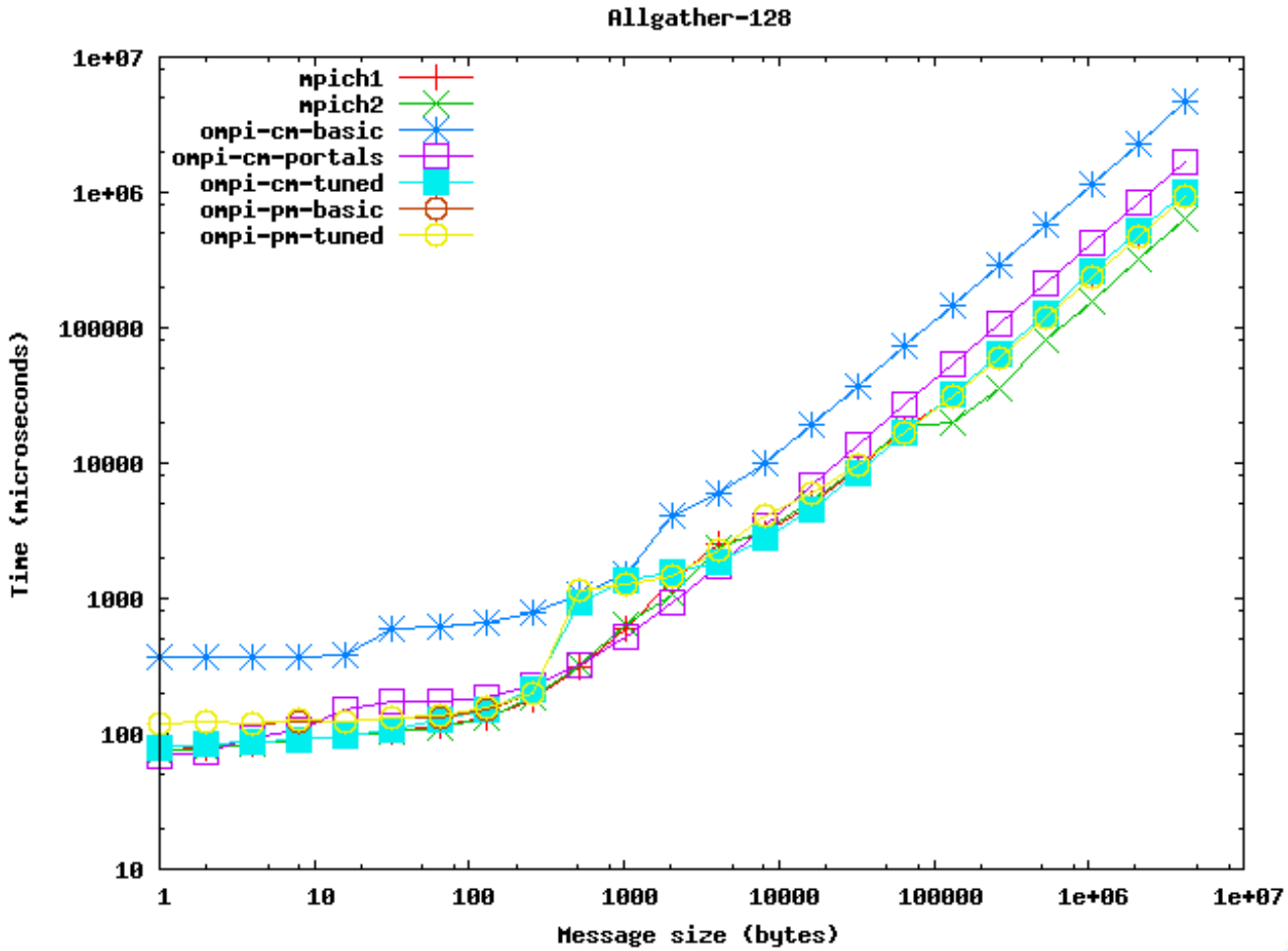
Current Status

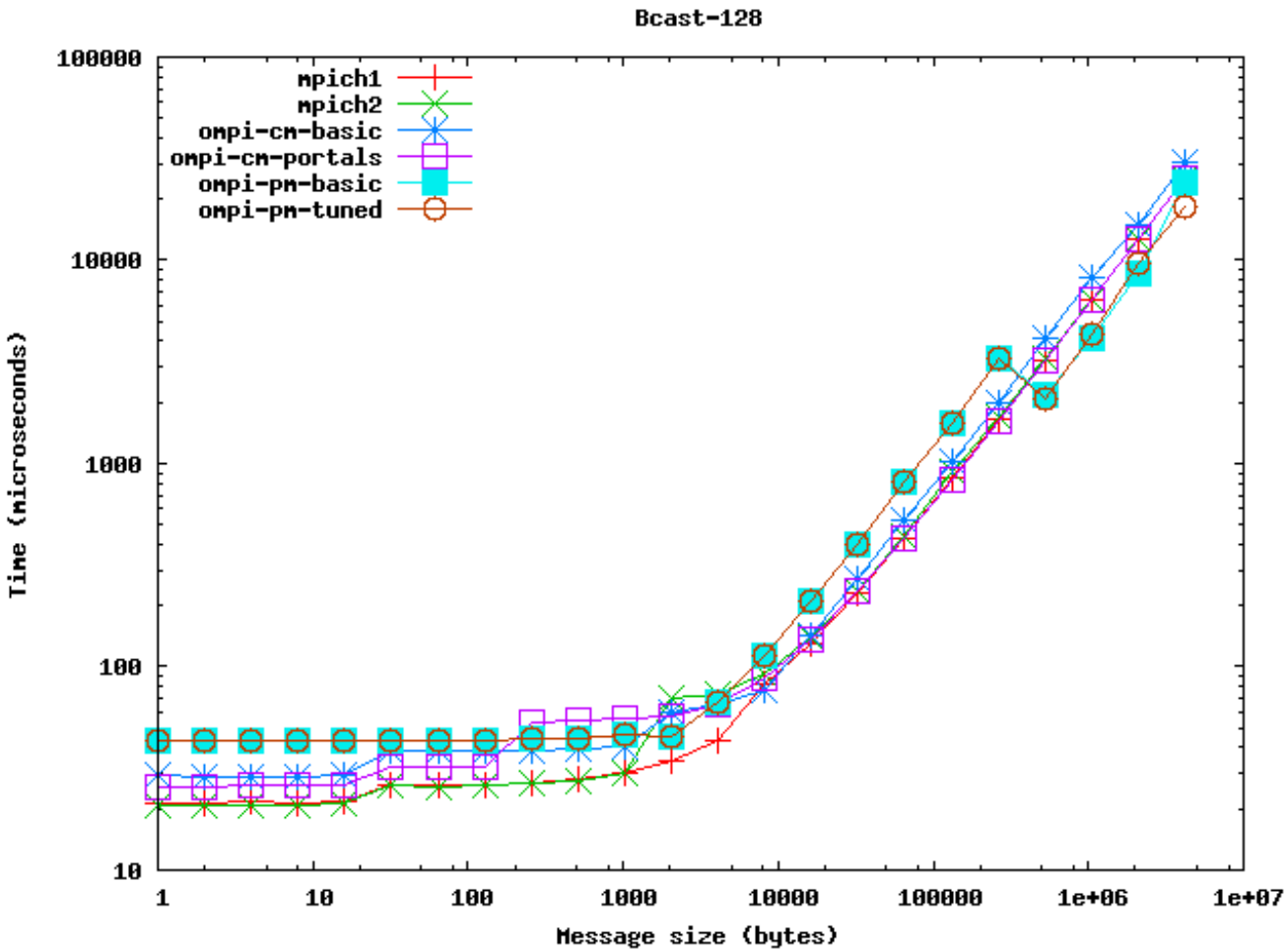
- Implemented Barrier, Bcast, Gather, Scatter, Allgather
- Implemented subset of Reduce and Allreduce
  - Restricted to long integers and MIN and MAX operations
- Integrated into Open MPI
  - Limited to contiguous datatypes
  - Sometimes assumes significant-at-root-only arguments are significant everywhere
  - Doesn't support MPI_IN_PLACE yet
- Tested on 128-node Red Storm development cage

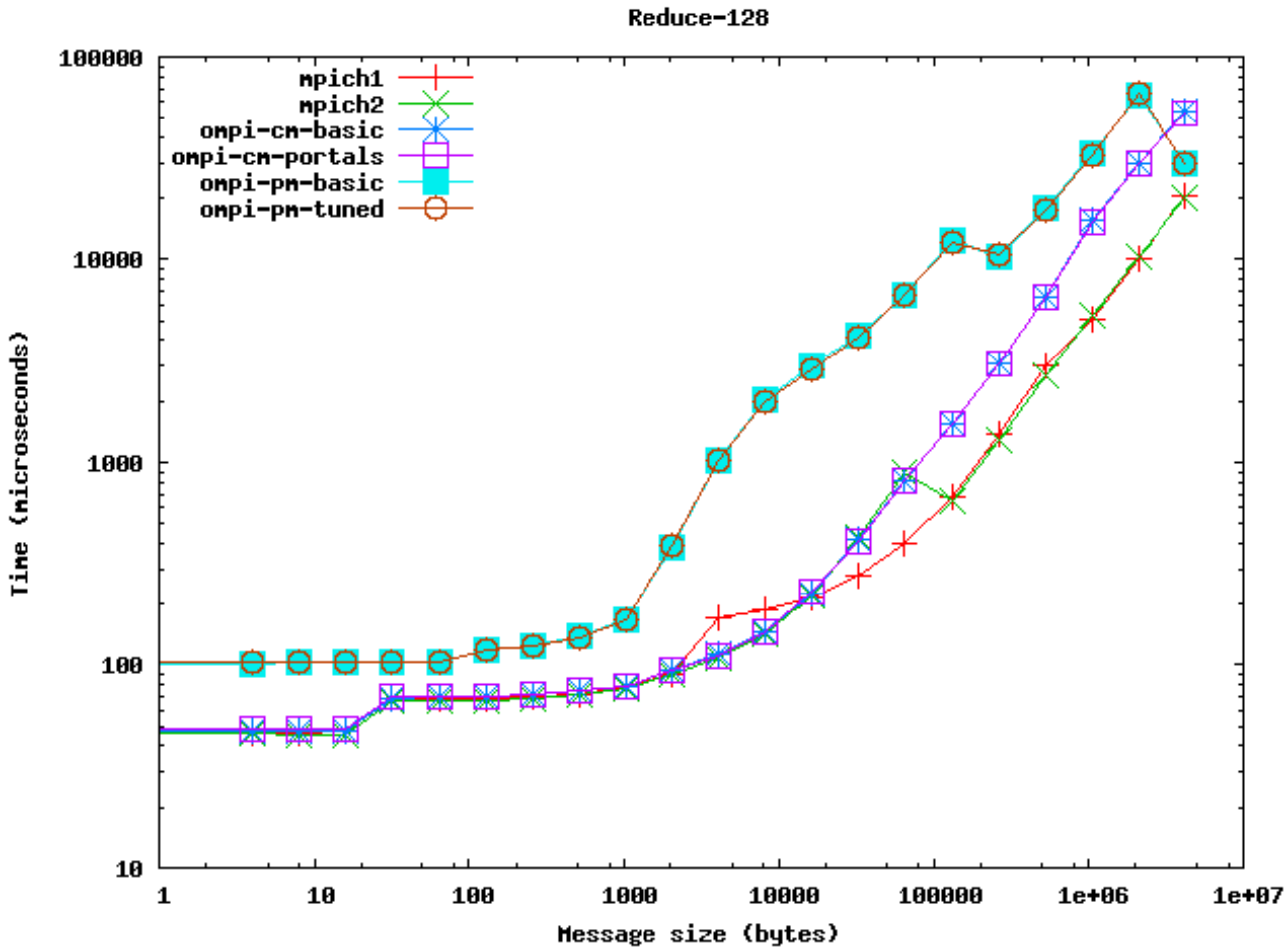
Performance Results

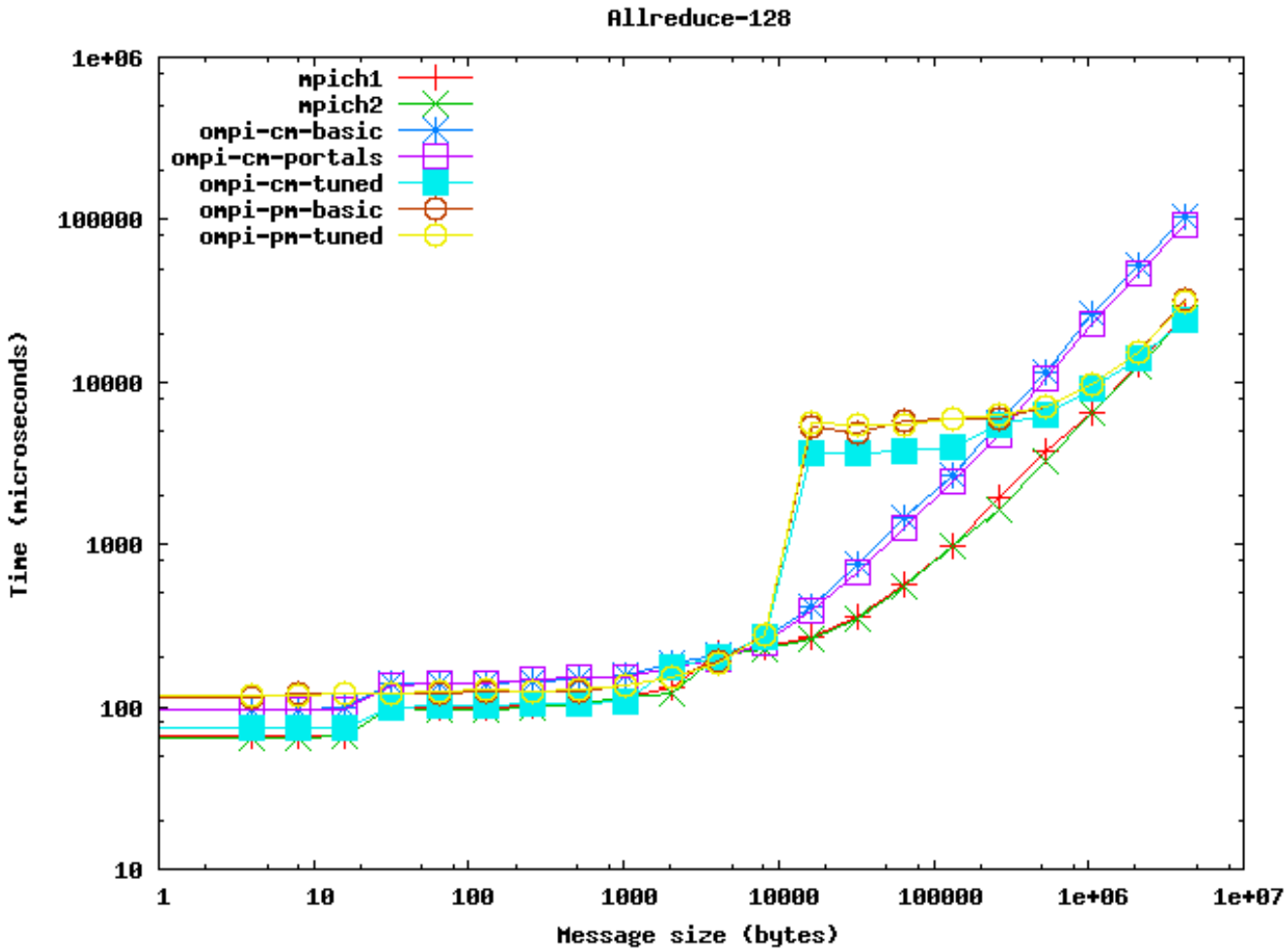
- Red Storm development cage
  - 2.0 GHz AMD Opteron
  - SeaStar 1.2
- Intel MPI Benchmark (IMB) Suite
- MPI implementations
  - Cray MPICH2
  - MPICH 1.2.6
  - Open MPI
    - Portals PM – matching in MPI
    - Portals CM – matching in Portals
    - Basic and tuned collectives


Barrier
mpich1
mpich2
ompi-cm-basic
ompi-cm-portals
ompi-pm-basic
ompi-pm-tuned
Time (microseconds)
100
10
1
1
10
100
Number of Nodes


Allgather-128
mpich1
mpich2
ompi-cm-basic
ompi-cm-portals
ompi-cm-tuned
ompi-pm-basic
ompi-pm-tuned
Time (microseconds)
1e+07
1e+06
100000
10000
1000
100
10
1
10
100
1000
10000
100000
1e+06
1e+07
Message size (bytes)


Bcast-128
mpich1
mpich2
ompi-cm-basic
ompi-cm-portals
ompi-pm-basic
ompi-pm-tuned
Time (microseconds)
100000
10000
1000
100
10
1
10
100
1000
10000
100000
1e+06
1e+07
Message size (bytes)


Reduce-128
mpich1
mpich2
ompi-cm-basic
ompi-cm-portals
ompi-pm-basic
ompi-pm-tuned
Time (microseconds)
100000
10000
1000
100
10
1
10
100
1000
10000
100000
1e+06
1e+07
Message size (bytes)


Allreduce-128
mpich1
mpich2
ompi-cm-basic
ompi-cm-portals
ompi-cm-tuned
ompi-pm-basic
ompi-pm-tuned
Time (microseconds)
1e+06
100000
10000
1000
100
10
1
10
100
1000
10000
100000
1e+06
1e+07
Message size (bytes)

Future Work

- Lots ☺
- Start optimizing
  - Eliminate startup costs
  - Avoid memory copies
  - Evaluate eager/rendezvous strategy
  - Make better use of Portals semantics
  - Alternative algorithms for large messages
  - Topology/routing optimizations
- Analysis using real applications