SAND2019-1533C

SYSTEM-LEVEL ARCHITECTURE SIMULATION FOR EXASCALE: CHALLENGES AND OPPORTUNITIES

ECP All-Hands Meeting

Hardware Evaluation Interconnect Working Group

Houston, Texas
17 January 2019

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Introduction

Hardware Evaluation Interconnect Working Group

The Hardware Evaluation Interconnect Working Group is focused on the analysis of current and future High Performance Computing Interconnects to inform the development and procurement of future HPC systems. This will be accomplished by evaluating architectural advancements proposed as part of the PathForward program and through studies of potential interconnects for future exascale-class systems. Analysis of future systems will be done using architectural simulation tools and informed by data collected on current HPC systems.

Tools and Capabilities in System Level Simulation/Analysis

- CODES
  - Available Open Source: https://xgitlab.cels.anl.gov/codes/codes.git and https://github.com/carothersc/ROSS
  - Version 1.0.0
- TraceR
  - Available Open Source: https://github.com/LLNL/tracer
  - Version 2.1
- SST
  - Available Open Source: https://github.com/sstsimulator
  - Version 8.1
- coNCePTuaL
  - Available Open source: http://conceptual.sourceforge.net/
  - Version 1.5.1

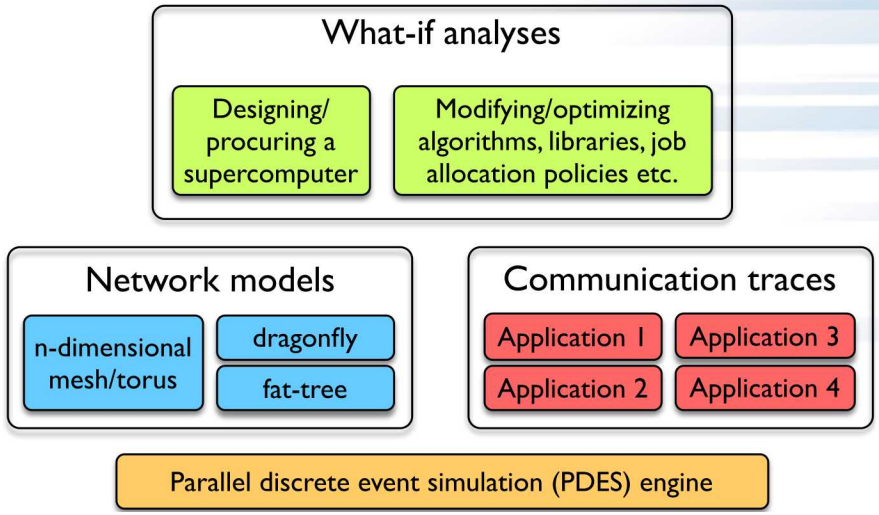
End Point Models

Representing HPC Workloads in Simulations

- **Trace driven simulation** of HPC communication
  - Support trace replay of DUMPI and OTF2 traces
  - **Benefits:** Detailed representation of communication.
  - **Challenges**: limited scalability and large trace sizes.
- **Skeleton applications**
  - Language or functions to simulate communication at high level. Can be hand or auto generated
  - **Benefits**: Scalability and lower memory overheads.
  - **Challenges**:
- **State machine representations**
  - Description
  - Benefits:
  - Challenges:
- **CoNCePTuaL domain specific language**
  - Ties in through one of the other methods

Trace-driven Simulation

- Trace-based simulation of production parallel applications
- Support for multi-job workloads



Example Skeleton Code: All to One

Scalable Workload Model:

```
void IncastAllToOne::call() {
  /* processes in the range of min_source_id and max_source_id send messages to a specific
     destination. */
  if ((proc_id != dst_rank_id) && (proc_id >= min_source_id && proc_id <= max_source_id)) {
    for (uint32_t iter=0; iter < iteration_cnt; iter++) {
      SWM_Isend(dst_rank_id, SWM_COMM_WORLD, this_tag, reqArg1, rspArg1, NO_BUFFER,
                msg_size, pkt_rsp_bytes, &(send_handles[send_count]), reqArg2,
                  rspArg2);
    }
    SWM_Waitall(send_limit, send_handles);
}
```

coNCePTuaL:

Tasks {t for each t in {min_source_id, ..., max_source_id} where t <> dst_rank_id} asynchronously send iteration_cnt msg_size byte messages to task dst_rank_id then all tasks await completion.

Notes:

- This is a *complete* coNCePTuaL program corresponding to the same all-to-one code above
- This can compile to a C+MPI program for execution on a real system or for directly driving a network simulator.
- coNCePTuaL implicitly converts all undeclared variables into command-line arguments, allocates memory for messages and MPI data structures, and matches sends and receives, letting the programmer focus solely on the communication pattern.

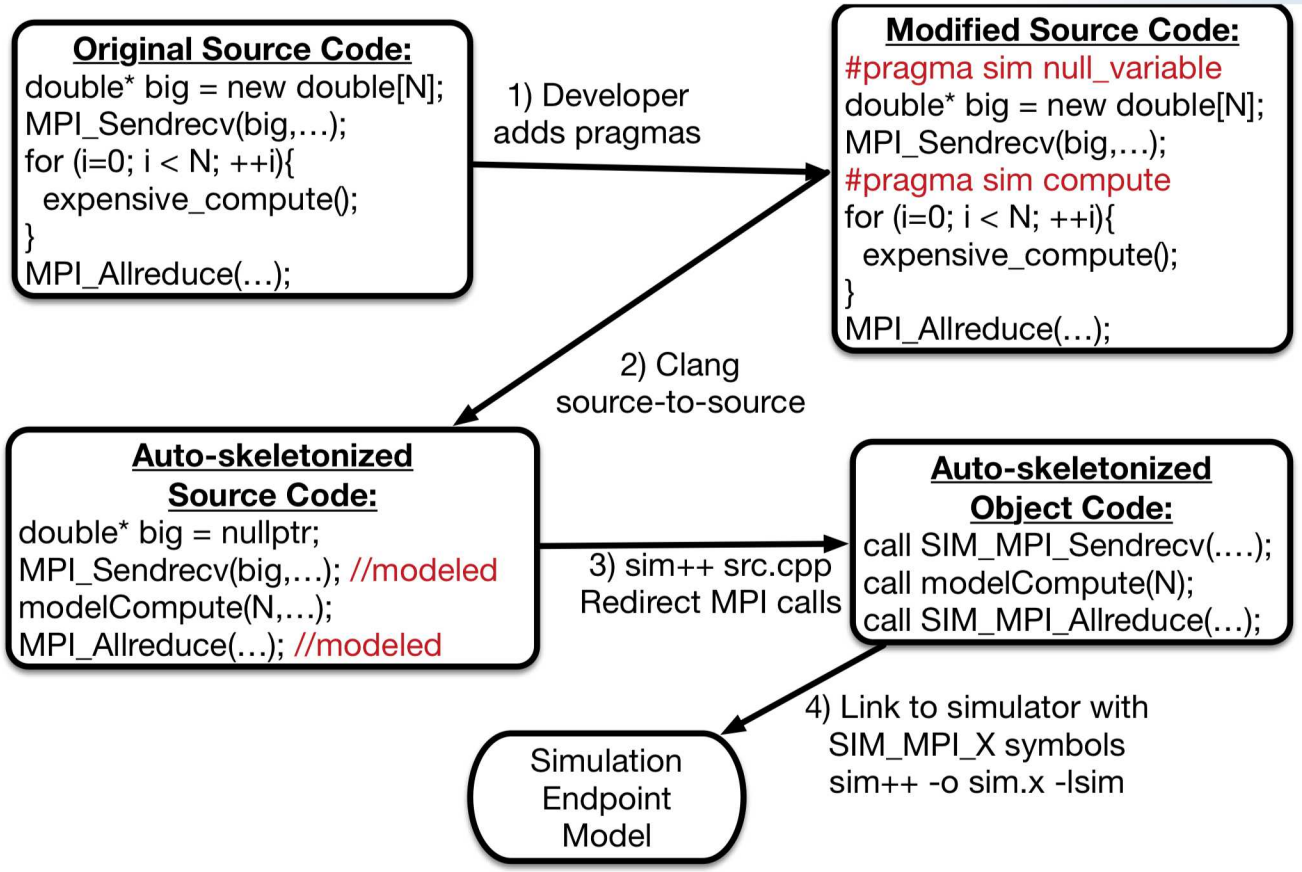
Skeletonized Apps: Compiler generated source-to-source skeletonization

Advantages

- Auto-generate flexible lightweight models
- True co-design: no simulation-specific models
- Realistic modeling of node compute based on app characteristics

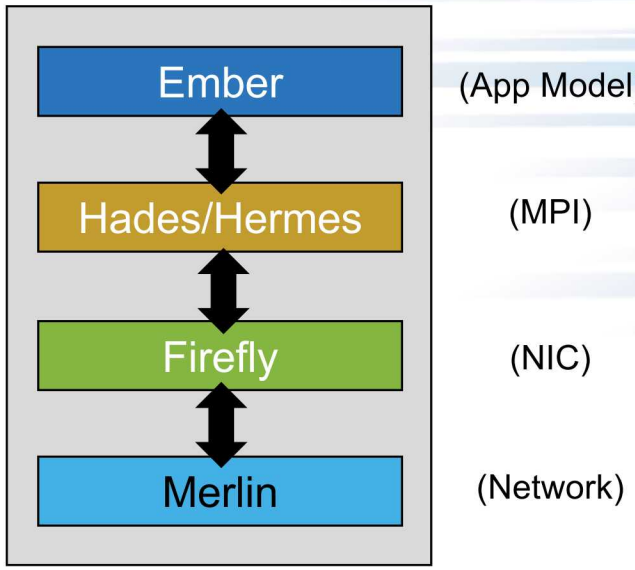
Challenges

- Not totally automatic – needs some pragma hints
- Need to build out LLVM instrumentation for better compute models



State Machine Models: Motifs

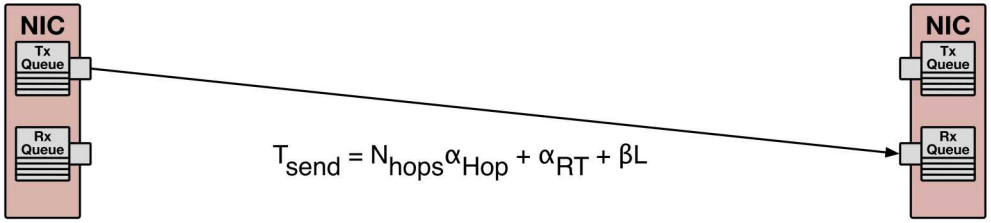
```
bool EmberAlltoallGenerator::generate( std::queue<EmberEvent*>& evQ) {
  if ( m_loopIndex == m_iterations ) {
    if ( 0 == rank() ) {
      double latency = (double)(m_stopTime-m_startTime)/(double)m_iterations;
      latency /= 1000000000.0;
      output( "%s: ranks %d, loop %d, bytes %d, latency %.3f us\n",
      getMotifName().c_str(), size(), m_iterations, m_bytes, latency * 1000000.0  );
    }
    return true;
  }
  if ( 0 == m_loopIndex ) {
    enQ_getTime( evQ, &m_startTime );
  }

  enQ_compute( evQ, m_compute );
  enQ_alltoall( evQ, m_sendBuf, m_bytes, CHAR, m_recvBuf, m_bytes, CHAR, GroupWorld );

  if ( ++m_loopIndex == m_iterations ) {
    enQ_getTime( evQ, &m_stopTime );
  }
  return false;
}
```



Network Models

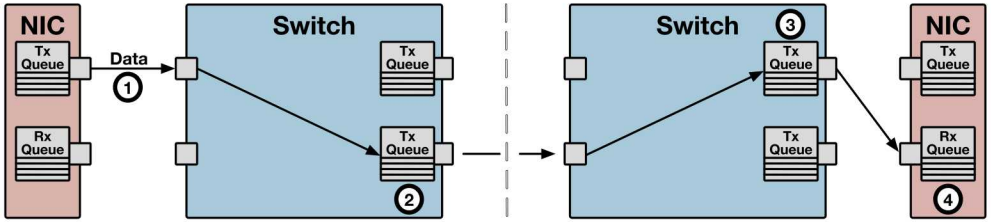
Range of efficient models to identify trends, quantify uncertainties in results

Lightweight simulation using analytic model



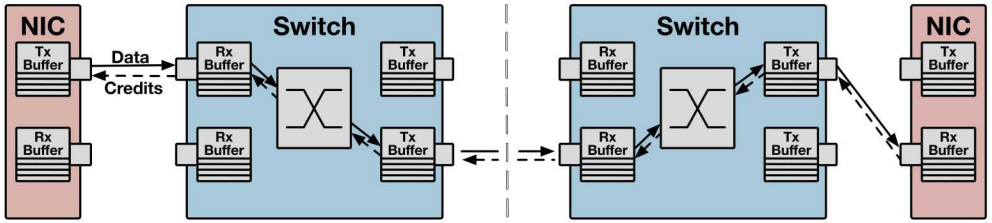
- Establishes *optimistic* upper-bound on performance in absence of contention
- Useful in validating software stack models for traffic patterns without contention

Packet-based with simple congestion modeling



- Establish baseline performance of different routing/congestion control strategies
- Efficient execution, agnostic to router flow control details

Packet-based with arbitration



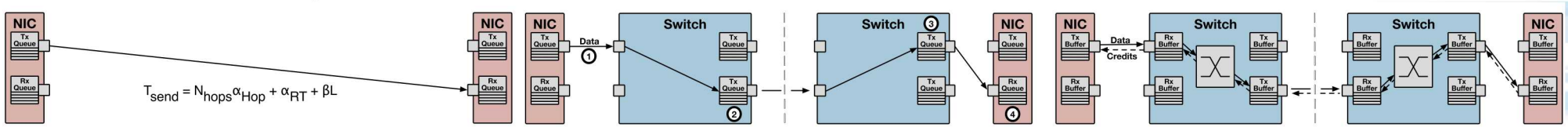
- Establishes *pessimistic* lower-bound on performance by tuning arbitration and token flow-control performance
- More complex and expensive

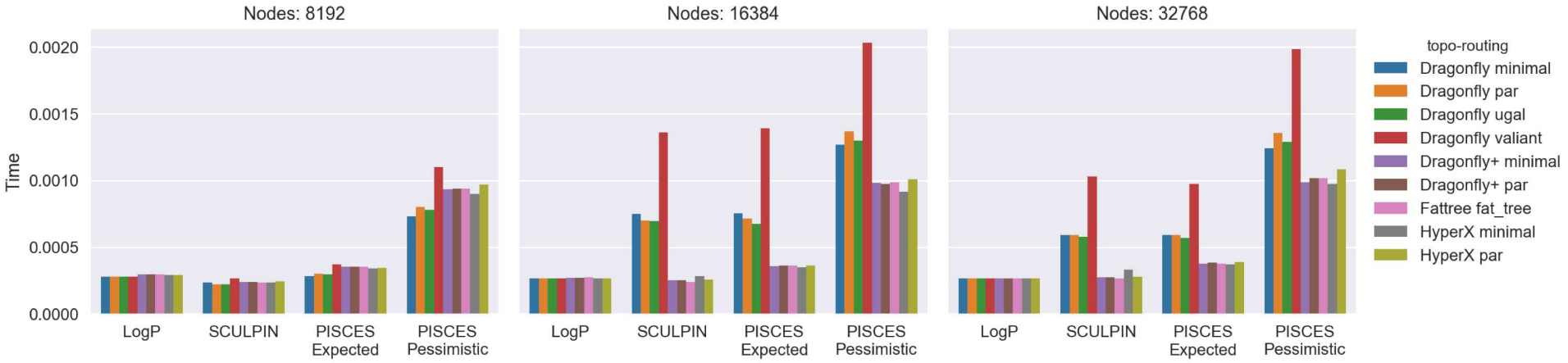
Range of efficient models to identify trends, quantify uncertainties in results

MACRELS: Analytic Model

SCULPIN: Efficient contention modeling

PISCES: Highest accuracy

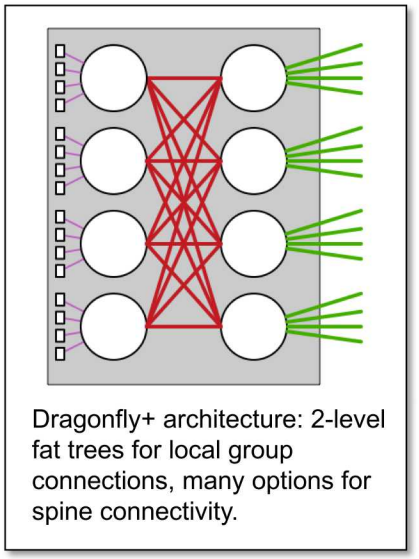




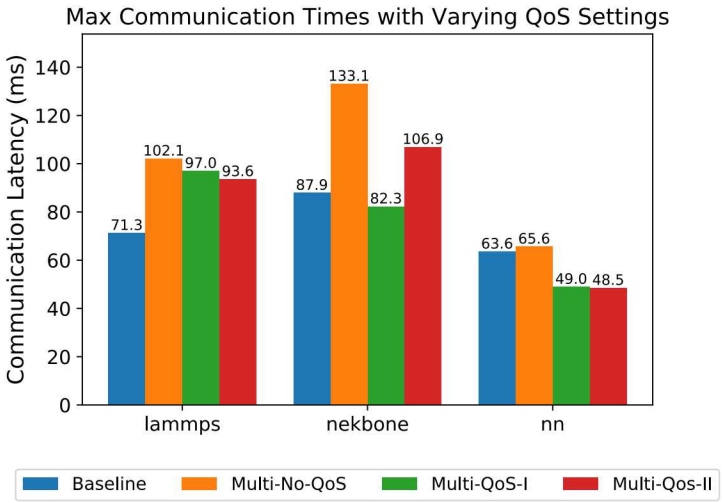
EXASCALE COMPUTING PROJECT

Topology and Routing

- Multiple topologies supported:
  - 1-D and 2-D dragonfly, fat tree, torus/mesh, slimfly, dragonfly+/megafly, hyperX
  - Multiple minimal and adaptive routing algorithms supported



Dragonfly+ architecture: 2-level fat trees for local group connections, many options for spine connectivity.

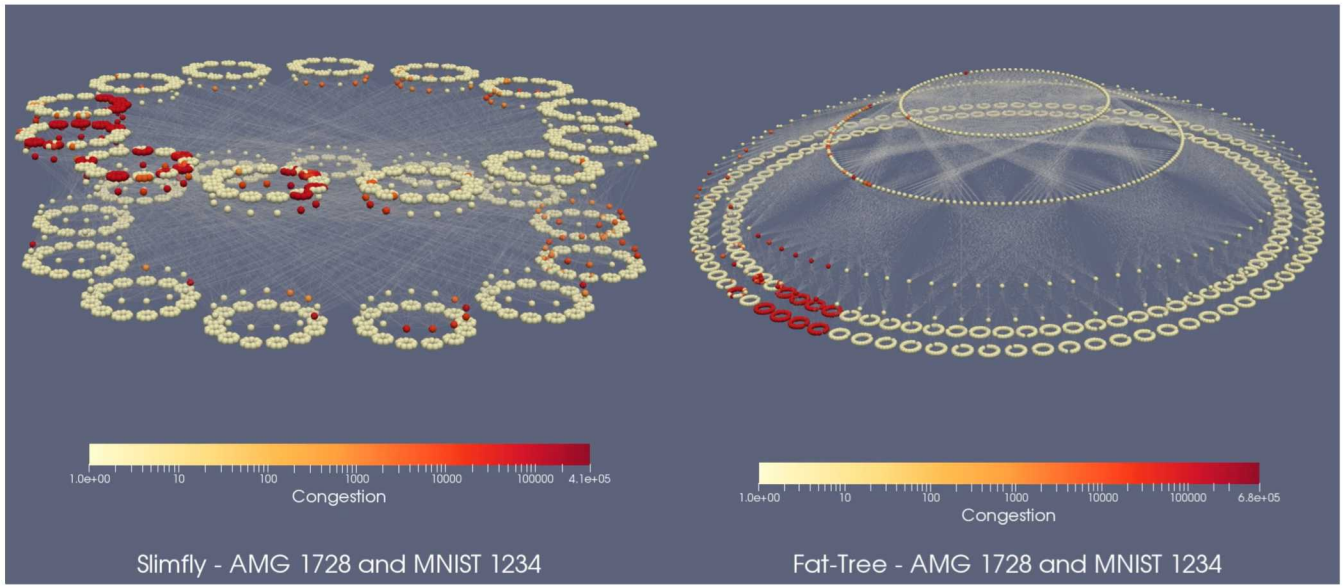


- Quality of Service and Advanced Congestion Management
  - Simulated QoS traffic classes by dedicating a set of virtual channels to each traffic class
  - Used bandwidth capping and traffic differentiation techniques to implement QoS on HPC networks

Insitu Analysis of Network Congestion

- **Just as in computational science simulations, output can be too large to realistically capture for post-processing.**
- Couple simulation with analysis code via Damaris data management system
- Damaris forwards to analysis tools (e.g., VisIt)

Visualizations of network congestion in two-application runs on candidate HPC networks. Visualization generated *in situ* would avoid excessive I/O.



https://project.inria.fr/damaris/

Collaborations

Collaborations and Users

Vendor Collaborations/Users

- Lab developed tools in use at:
  - Cray
  - Intel
  - IBM
  - HPE

National Laboratory Collaborations/Users

- Argonne National Lab
- Berkeley National Lab
- Fermi National Lab
- Livermore National Lab
- Oak Ridge National Lab
- Sandia National Labs

Universities

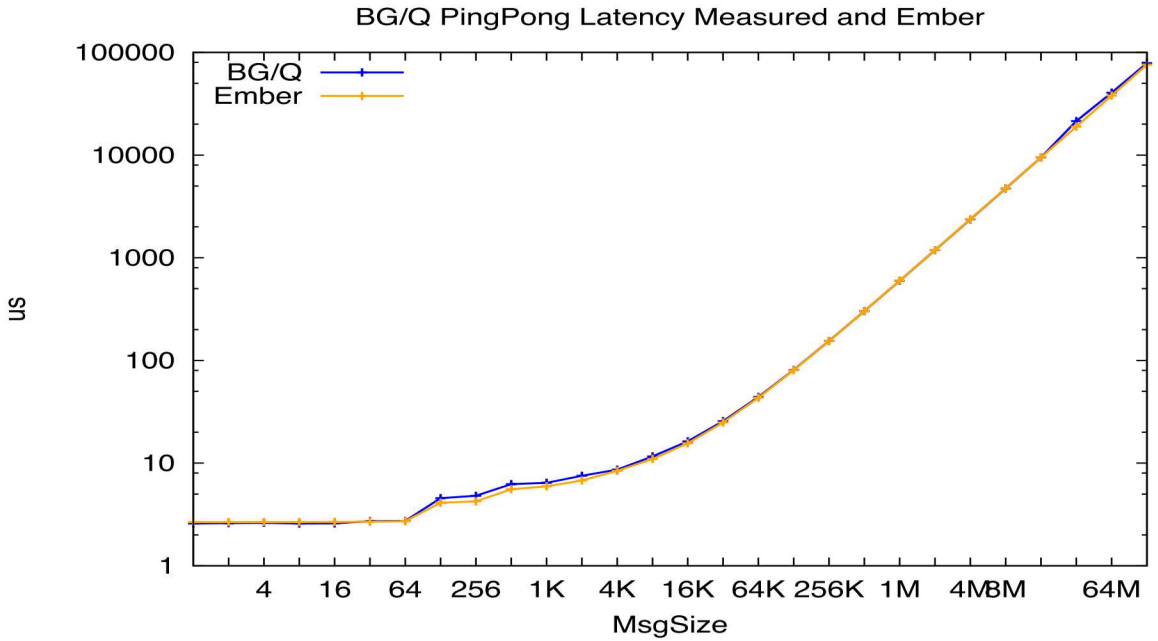
- Illinois Institute of Tech
- NC State University
- Tokyo Institute of Technology
- University of Tsukuba
- Florida State University
- Kyushu University
- The University of Arizona
- University of Oregon

Areas of Collaboration

- Improving accuracy of network simulations
- Exploring power-aware network links
- Studying impact of quality-of-service
- Improving MPI performance
- Use compiler support to combine motifs/skeletons with complete vendor MPI/provider software stack
- Use compiler support to leverage work from node/memory teams to improve endpoint models

Validation/Quantifying Uncertainties Examples

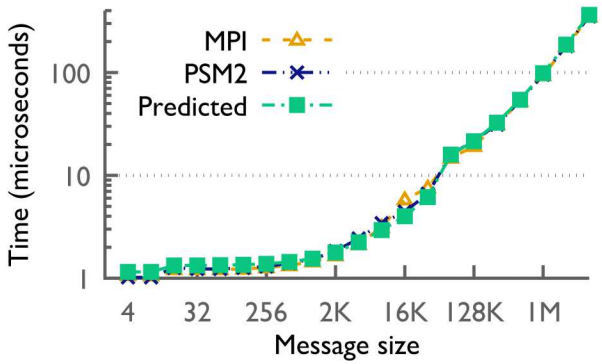
Validation Example: Network Stack (SST/Merlin)



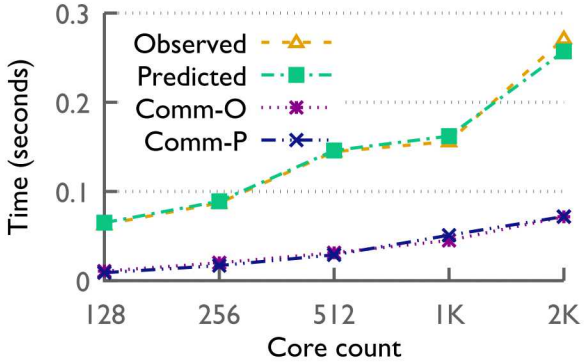
- Challenges:
  - Validate software stack timings
  - Match protocol switch over points
- Also validates quiet switch/router and link latencies

Validation Example: System Level Validation (TraceR)

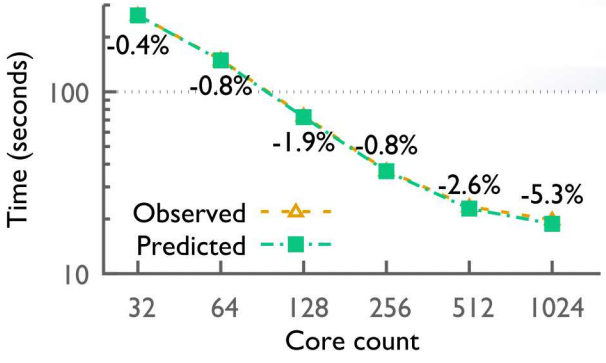
- Fat-tree network model validated against experiments on Quartz at LLNL using TraceR



(a) Ping-pong



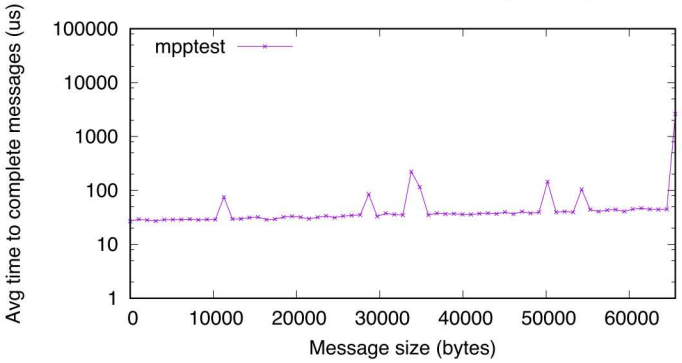
(b) 3D Stencil



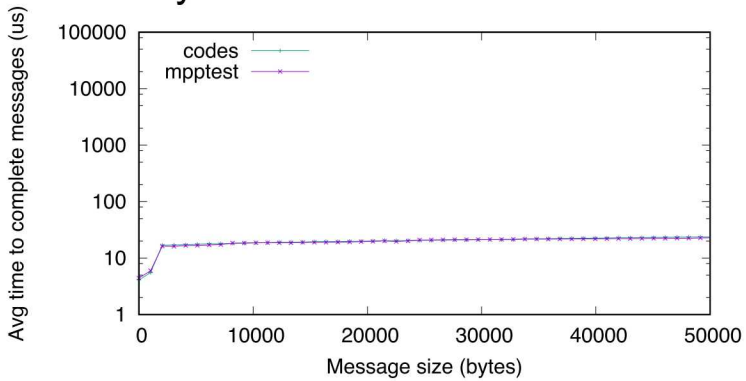
(c) Atratus

Validation Example: Interference Effects (CODES)

Interference effects on Dragonfly-based Theta Cray XC



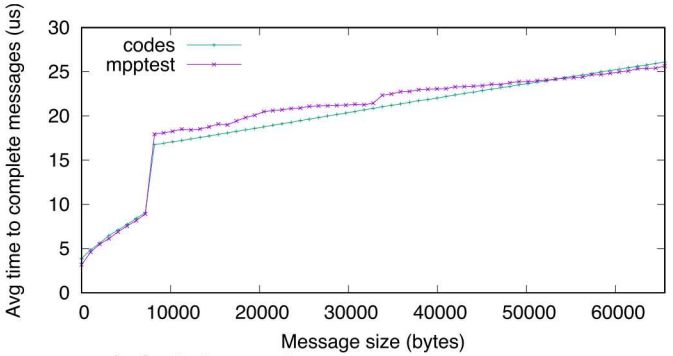
(a) With interfering jobs



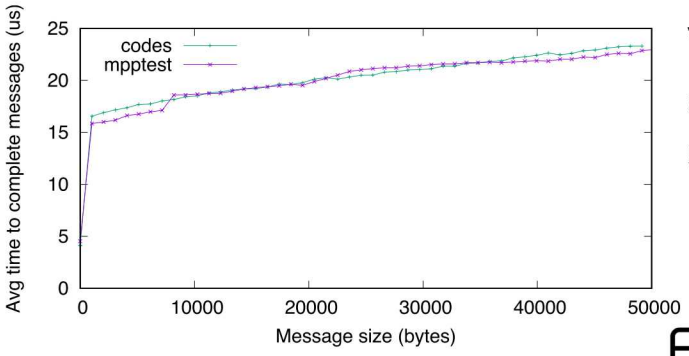
(b) without interfering jobs

Need to account for any interference effects on the system.

Dragonfly network model validated against experiments on Theta Cray XC



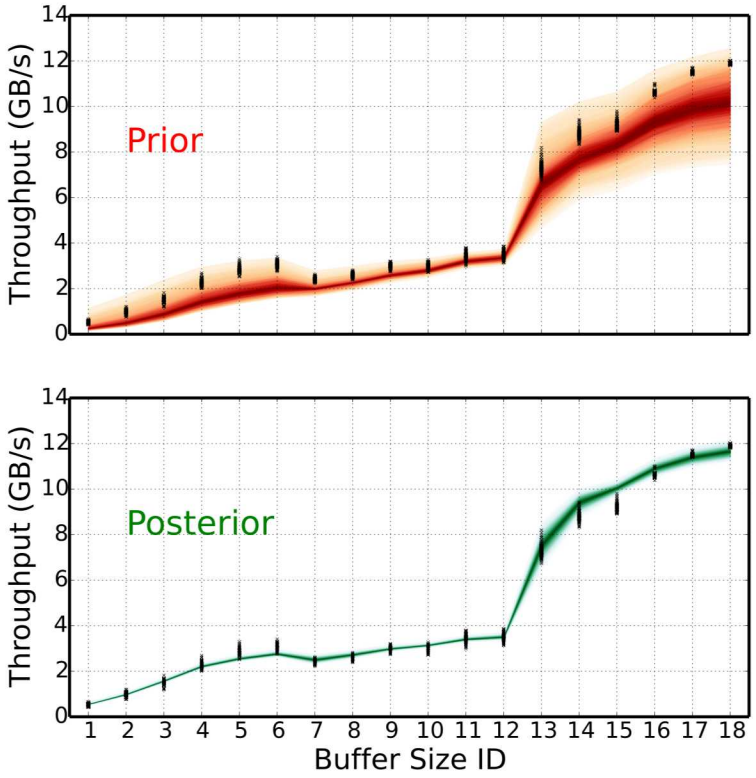
(a) 64 nodes



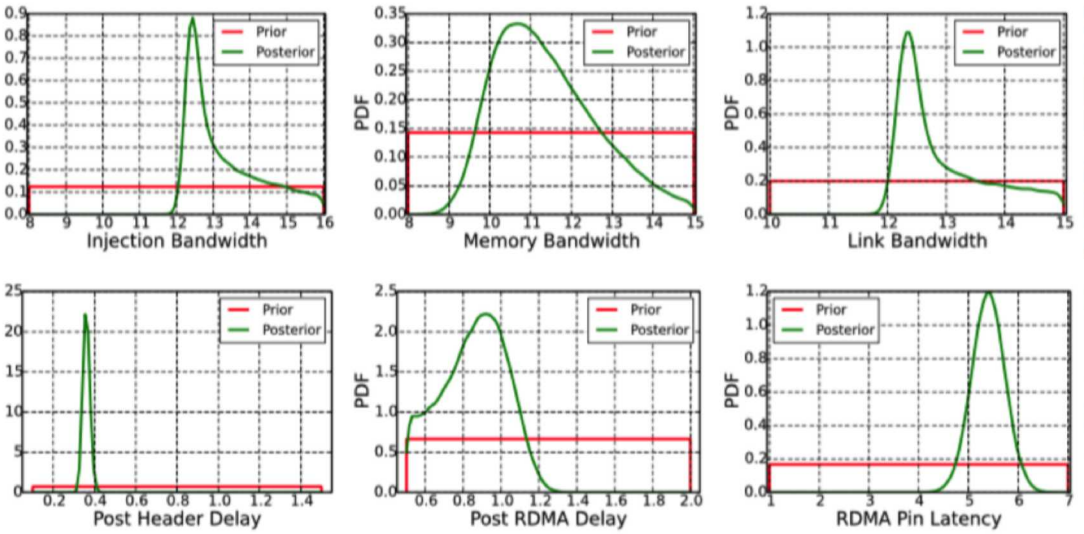
(b) 2048 nodes

Validation against a quiet Cray XC system

Quantifying Uncertainties Example (SST/Macro)



Combine parameter calibration with UQ to identify accuracy limits



Parameter	Max Likelihood[3]	Prior Range	Type
Injection Bandwidth (GB/s)	13.04	8.0 - 16.0	Network
Link Bandwidth (GB/s)	12.47	10.0 - 15.0	Network
Memory Bandwidth (GB/s)	11.20	8.0 - 15.0	System Software
Post Header Delay (us)	0.36	0.1 - 1.5	System Software
Post RDMA Delay (us)	0.88	0.5 - 2.0	System Software
RDMA Pin Latency (us)	5.43	1.0 - 7.0	System Software
RDMA Pin Delay Per Page (ns)	50.50	1.0 - 100.0	System Software
Hop Latency[4] (ns)	100	n/a	n/a

Milestone 3 (Q2FY19) – Validate simulator readiness to address full network stack design space

Goals

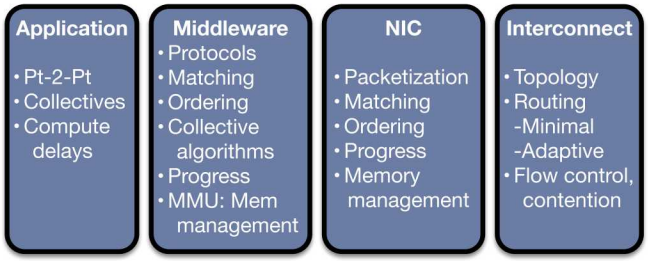
- ***Verify*** **simulator reproduces middleware behavior**
- ***Validate*** **simulators can explore all relevant design features**
- ***Check accuracy*** **of simulator approximations**

Methodology

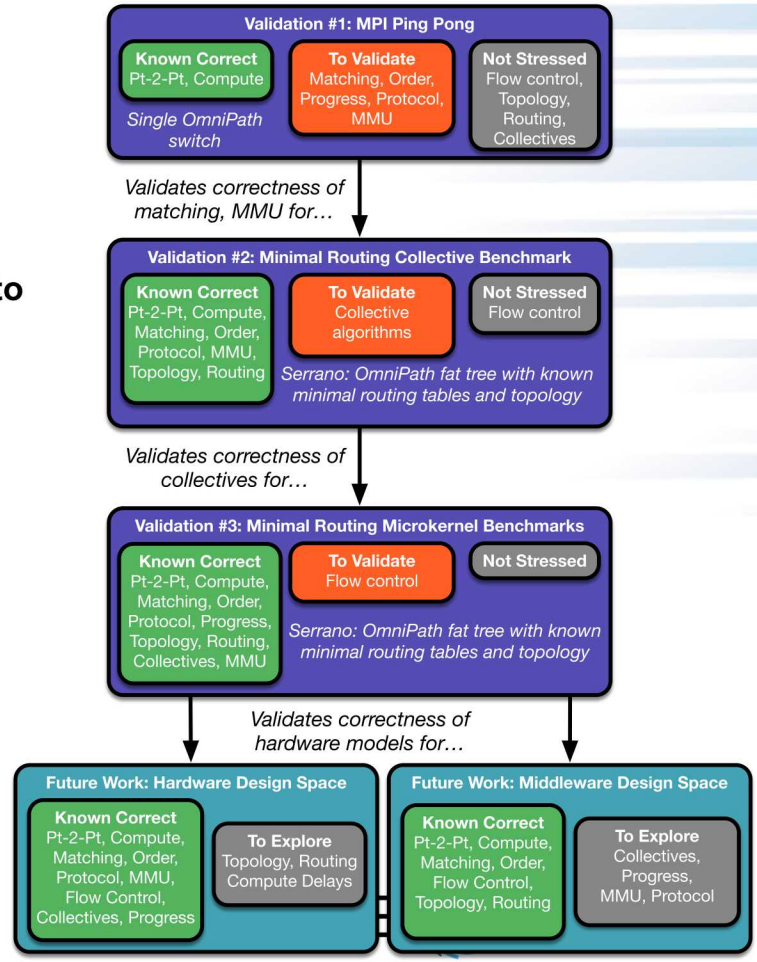
- **Controlled experiments of increasing complexity to validate individual pieces**
- **Use known minimal routing tables, topology of large OmniPath system to limit space of unknowns**
- **Microkernel benchmarks with known traffic patterns**

Ideal Outcomes

- **Validate simulator readiness to address full network stack design space**
- **Identify sources of disagreement, inaccuracy in simulation tools**



ECP HE Challenge: Huge permutational design space across network stack

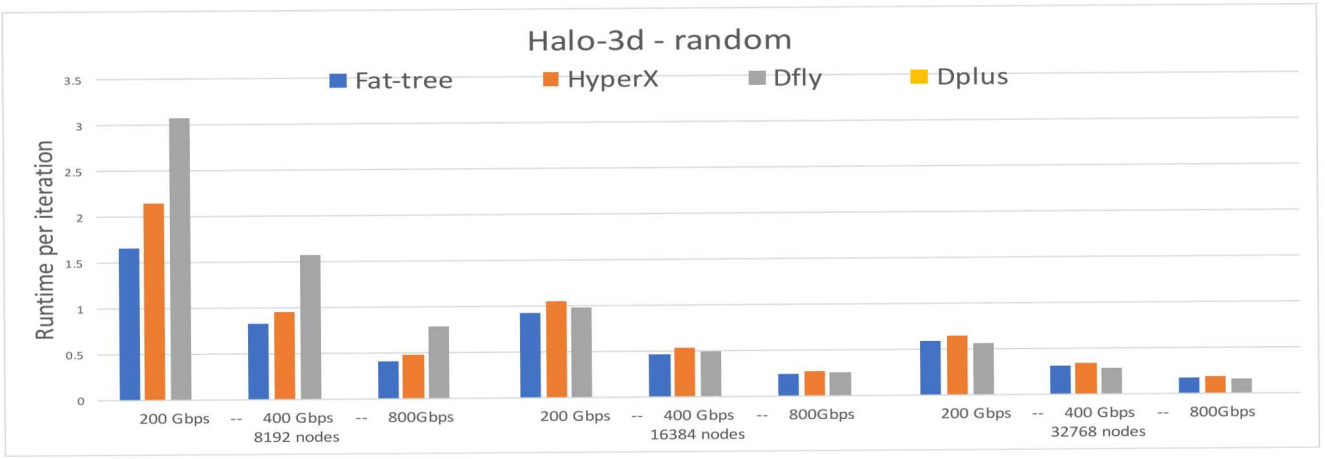


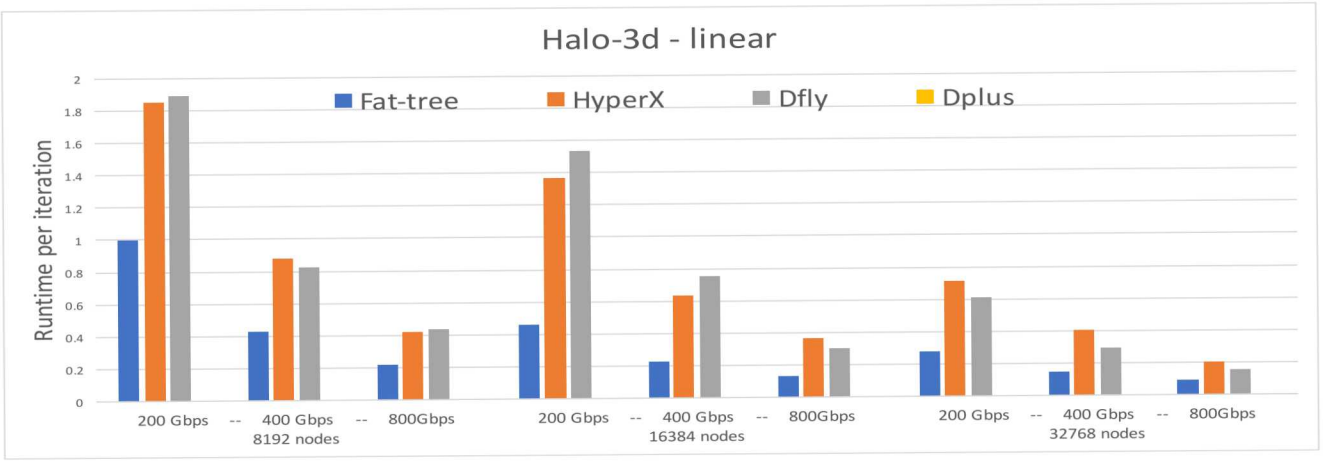
Milestone 1: Trade-offs in Exascale Interconnect Architectures

Activity 1 – Trade-offs in Exascale Interconnect Architectures

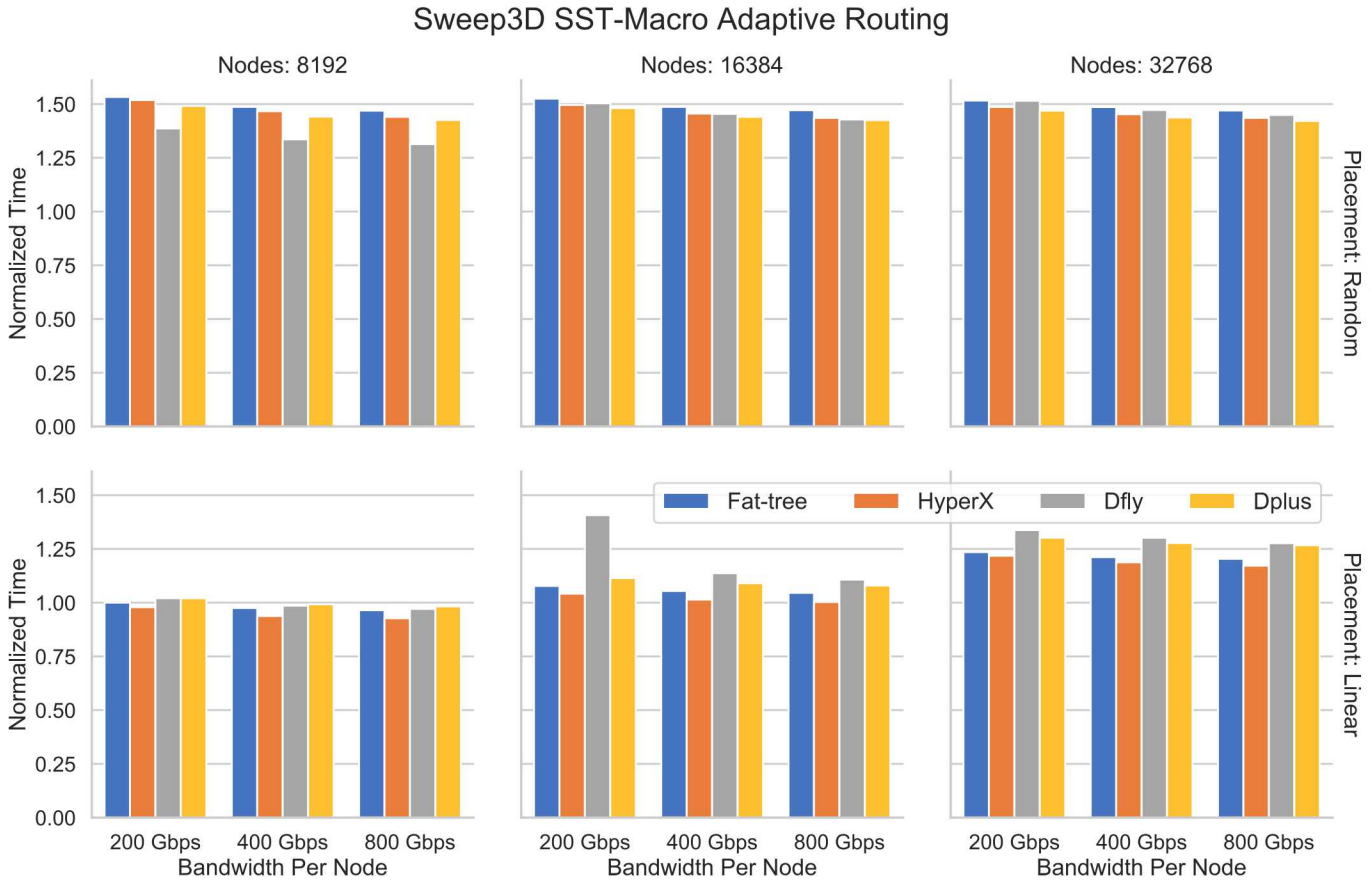
- Looks at the high-level trade-offs for interconnect architectures that are likely to be available in HPC machines in the 2021-2022 timeframe.
  - Machine size (8k, 16k, 32k nodes)
    - All use 32k ranks (4, 2, and 1 ranks/node)
  - Injection bandwidth (200, 400, 800 Gbps)
  - Topology (fat-tree, hyperX, dragonfly, dragonfly+/megafly)
  - Rank allocation (linear, random)
    - Helps determine worst case behavior
- Uses multiple simulation models (TraceR/CODES, SST/Macro, SST/Merlin)
  - Using multiple simulation environments can provide added confidence in the results, as well as providing added information for refining, understanding and improving the models.
- Benchmark communication patterns
  - Halo3D, Sweep3d, SubA2A/FFT3D

Halo3D26 – SST/Merlin



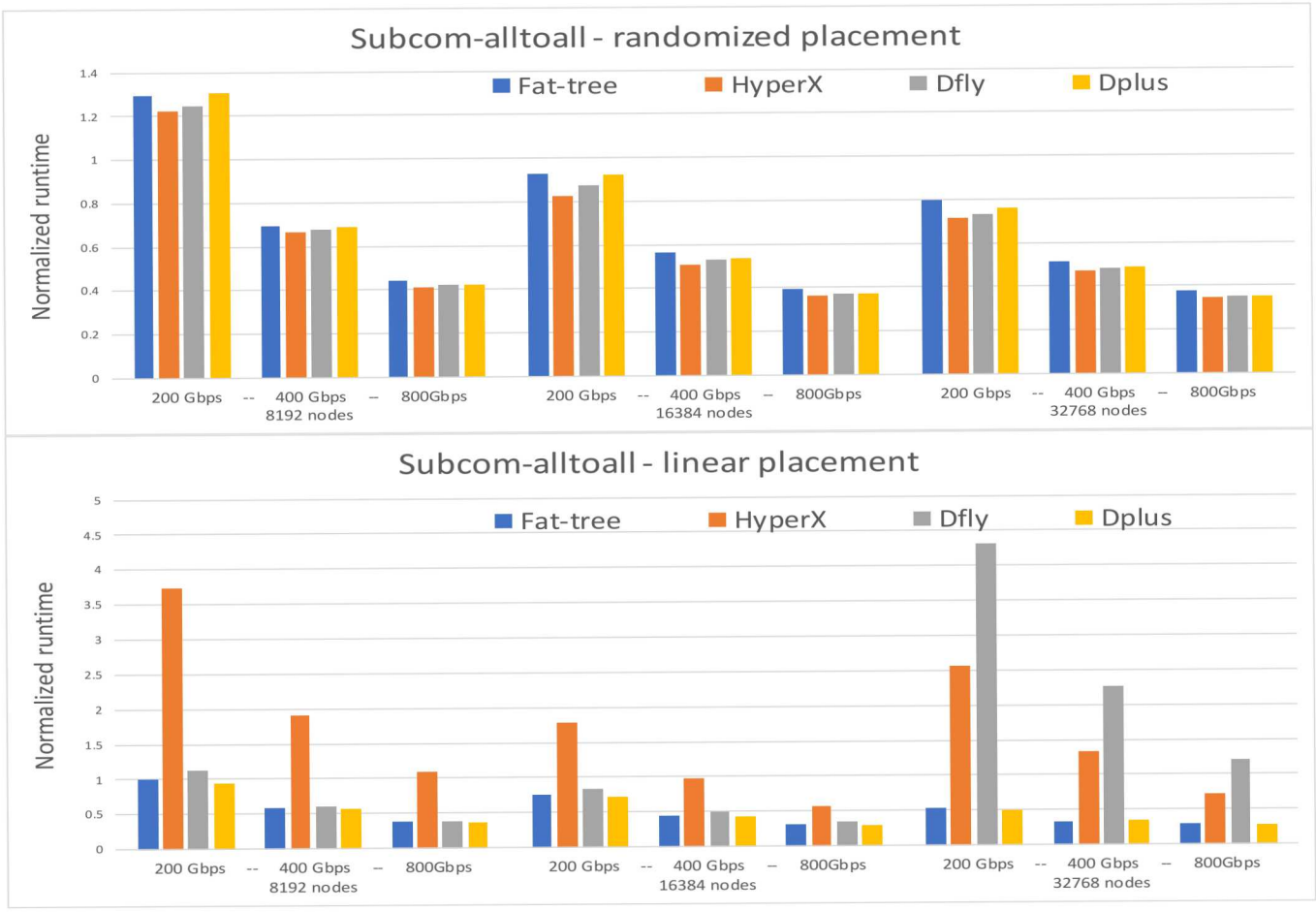


Sweep3D – SST/Macro



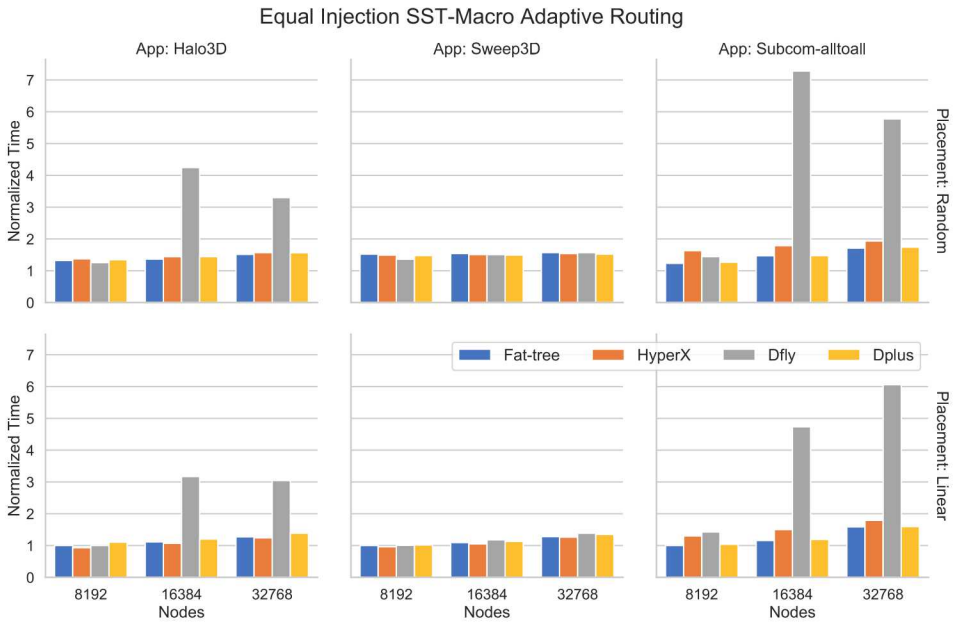
EXASCALE COMPUTING PROJECT

Subcom All-to-all - TraceR/CODES



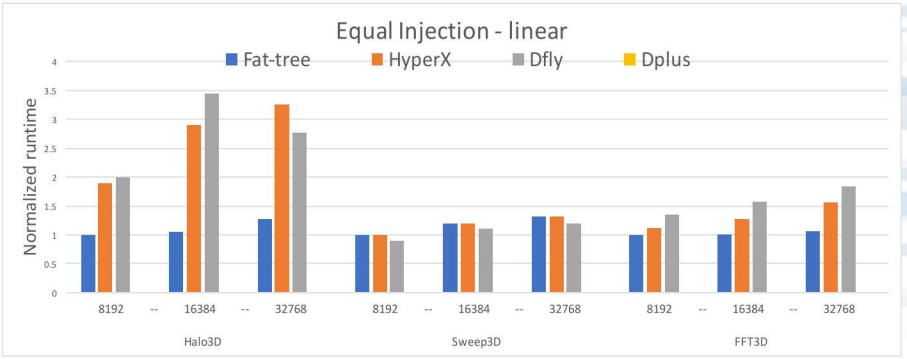
Equal Injection Bandwidth/Compute Ratio

SST-Macro

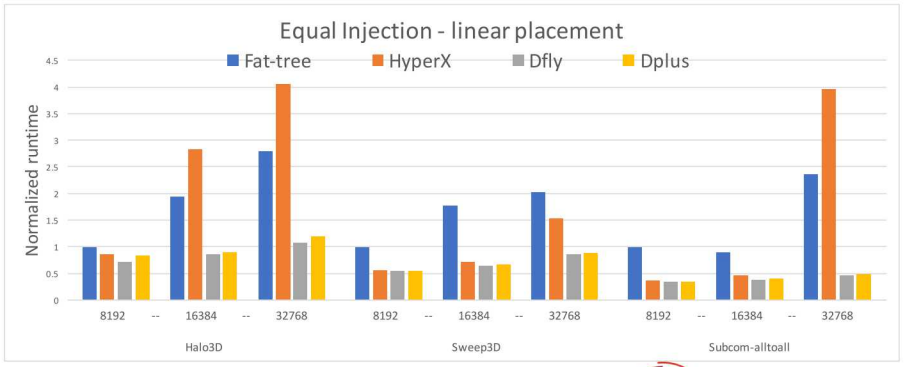


8192 – Quad rail (800 Gbps total)
16384 – Dual rail (400 Gbps total)
32768 – Single rail (200 Gbps total)

SST-Merlin



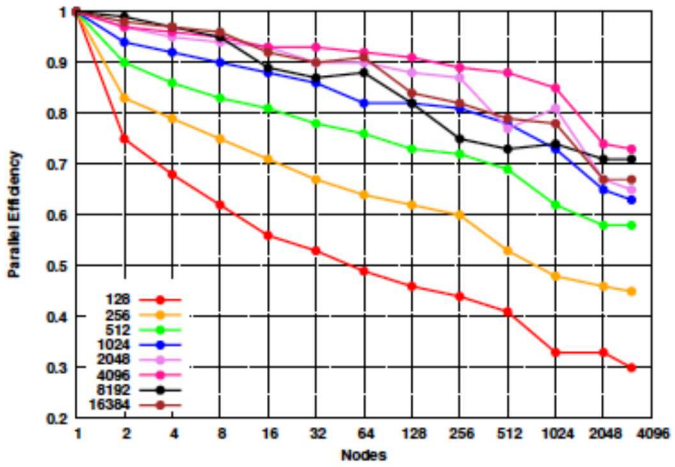
TraceR/CODES



Milestone 2: Analysis of interference sensitivity for next generation interconnects

Why quality of service on HPC systems?

- Emerging trend in HPC systems is that hierarchical topologies (such as dragonfly) are exhibiting run-to-run variability
- Poses significant challenge for job schedulers and application developers
- Possible ways to address variability:
  - Isolate job partitions (such as on Blue Gene systems with a torus network)
  - Explore alternate topologies and job mappings (minimize interference on fat-tree, hyperX)
  - Incorporate Quality of Service and traffic differentiation



Weak scaling of different Nekbone problem sizes on Theta Cray XC40 system (left). Nekbone itself is highly scalable as shown on right Mira Blue Gene/Q system.

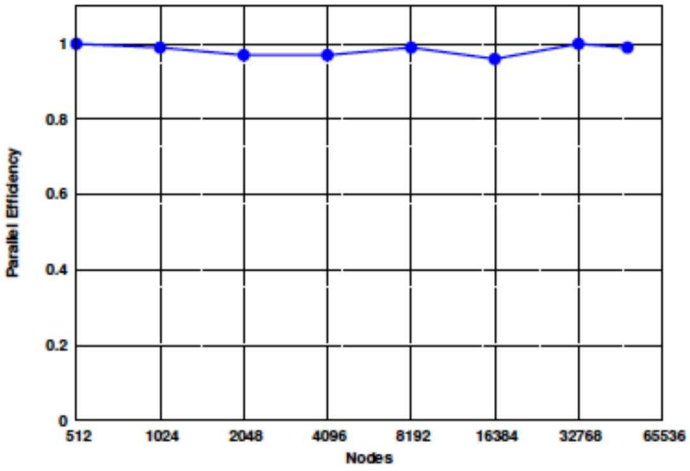


Image credit: “Early evaluation of the Cray XC40 Xeon Phi System ‘Theta’ at Argonne” (Tech Report) by S. Parker et al.

Key questions on applying QoS to HPC networks

- How effective are QoS traffic classes in regulating performance variability?
- What are the different ways in which applications can benefit from QoS traffic classes?
- How to make use of multiple traffic classes at the software (MPI, OpenMP) levels?
- How to make the job scheduler utilize multiple traffic classes effectively?

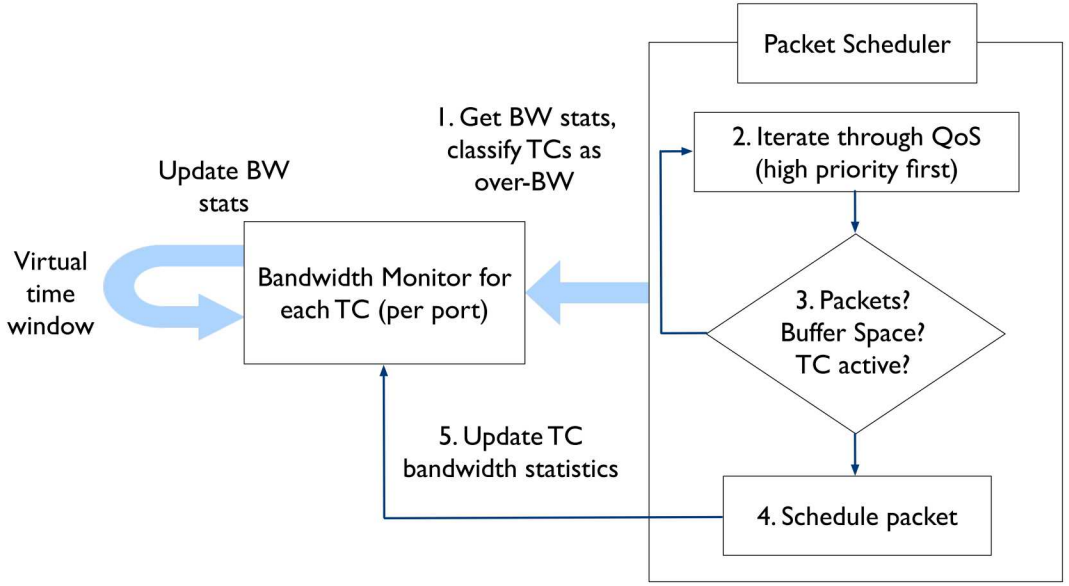


Fig: Traffic Differentiation and bandwidth capping algorithm fo switch/NIC scheduler (**TC- Traffic Class**)

Used Modeling and Simulation techniques to answer the first two questions

Prioritizing Entire HPC Applications

- Simulated slowdown in comm. time by generating background network traffic in parallel with Nekbone skeleton application
- Introduced QoS by assigning ***high priority & bandwidth*** to Nekbone skeleton app.
- While Nekbone doesn't utilize full bandwidth, the low priority application takes its BW share
- ***Takeaway: Traffic differentiation with bandwidth shaping and prioritization can mitigate variability while causing minimal slowdown to background traffic***

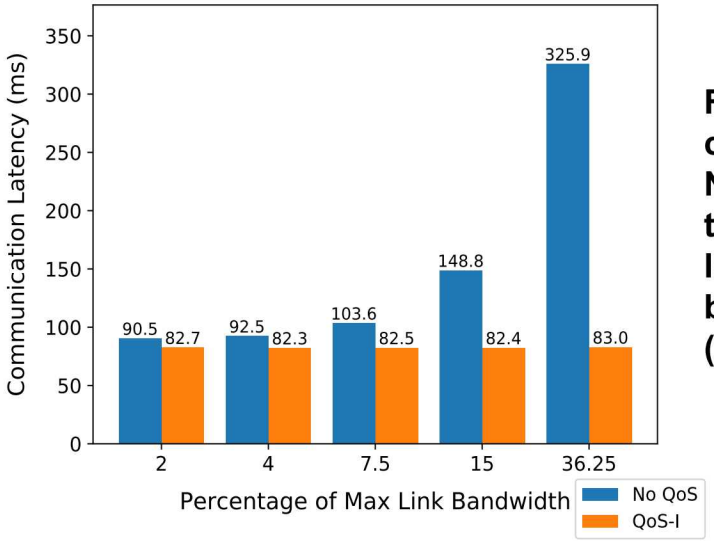
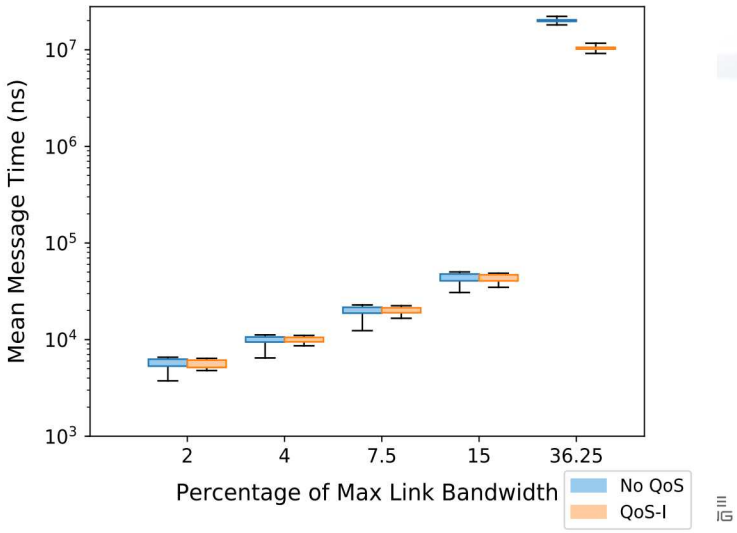


Figure: Two traffic classes on Megafly Network. High priority to Nekbone (left) and low priority to background traffic (right)



Prioritizing latency-sensitive operations

- Assigned ***high priority and guaranteed small fraction of BW*** to collectives
- High bandwidth and medium/low priority to rest of the traffic
- Nekbone skeleton application heavily relies on collective operations
- ***Takeaway: Traffic differentiation with small bandwidth guarantee and prioritization to collectives can bring up to 60% speed up in communication time with Nekbone skeleton app.***

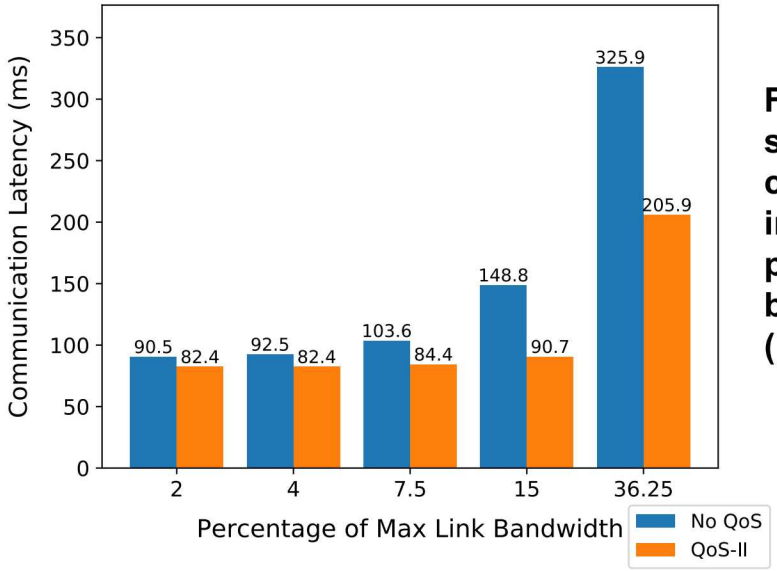
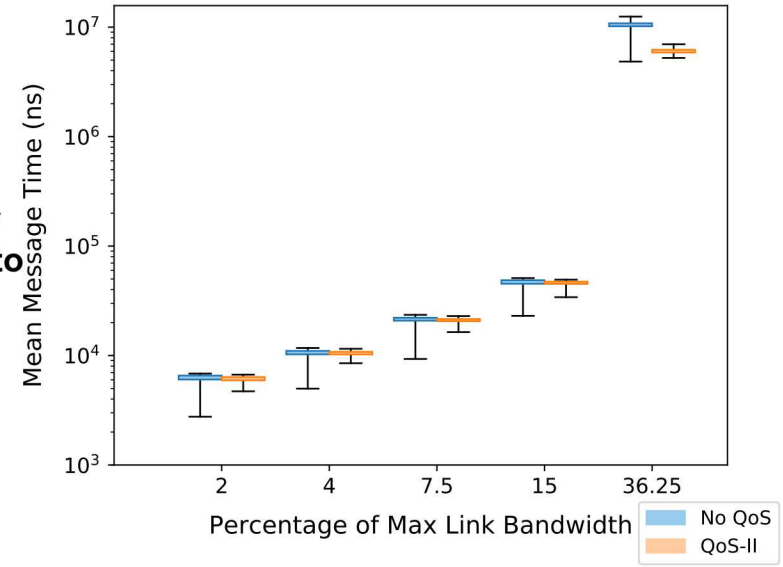
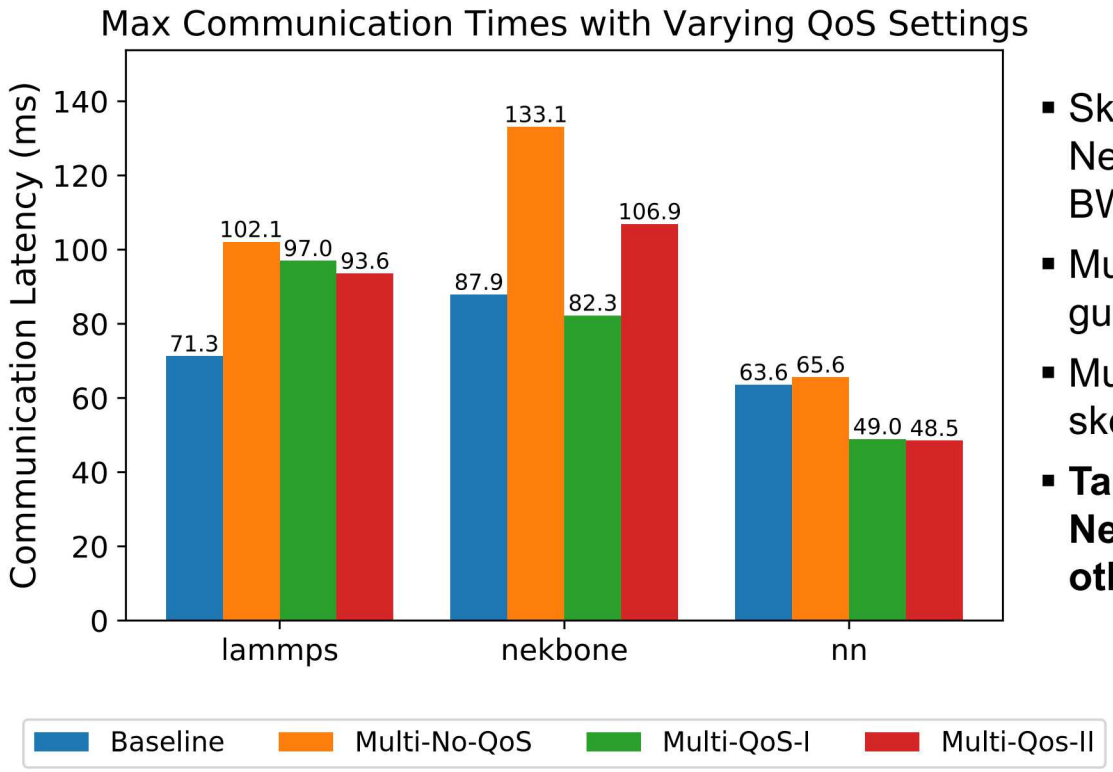


Figure: High priority, small BW cap to collective operations in Nekbone (left). Low priority and high BW to background traffic (right)



Multiple applications running In parallel



- Skeleton Applications in parallel: Nekbone, Nearest Neighbor and LAMMPS (Nekbone is BW intensive as compared to others)
- Multi-QoS-I: Prioritizing Nekbone and guaranteeing 1/3 BW
- Multi-QoS-II: Prioritizing collectives of all the skeleton applications
- **Takeaway: adding bandwidth cap on Nekbone helps improve the performance of other skeleton applications as well**