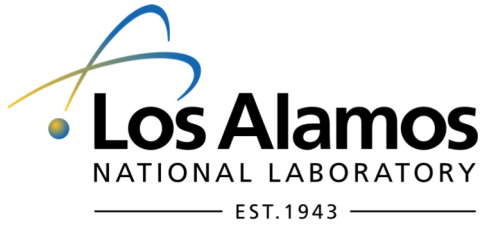


Interconnection Network Models for Rapid Performance Prediction of HPC Applications

Kishwar Ahmed

University of South Carolina Beaufort

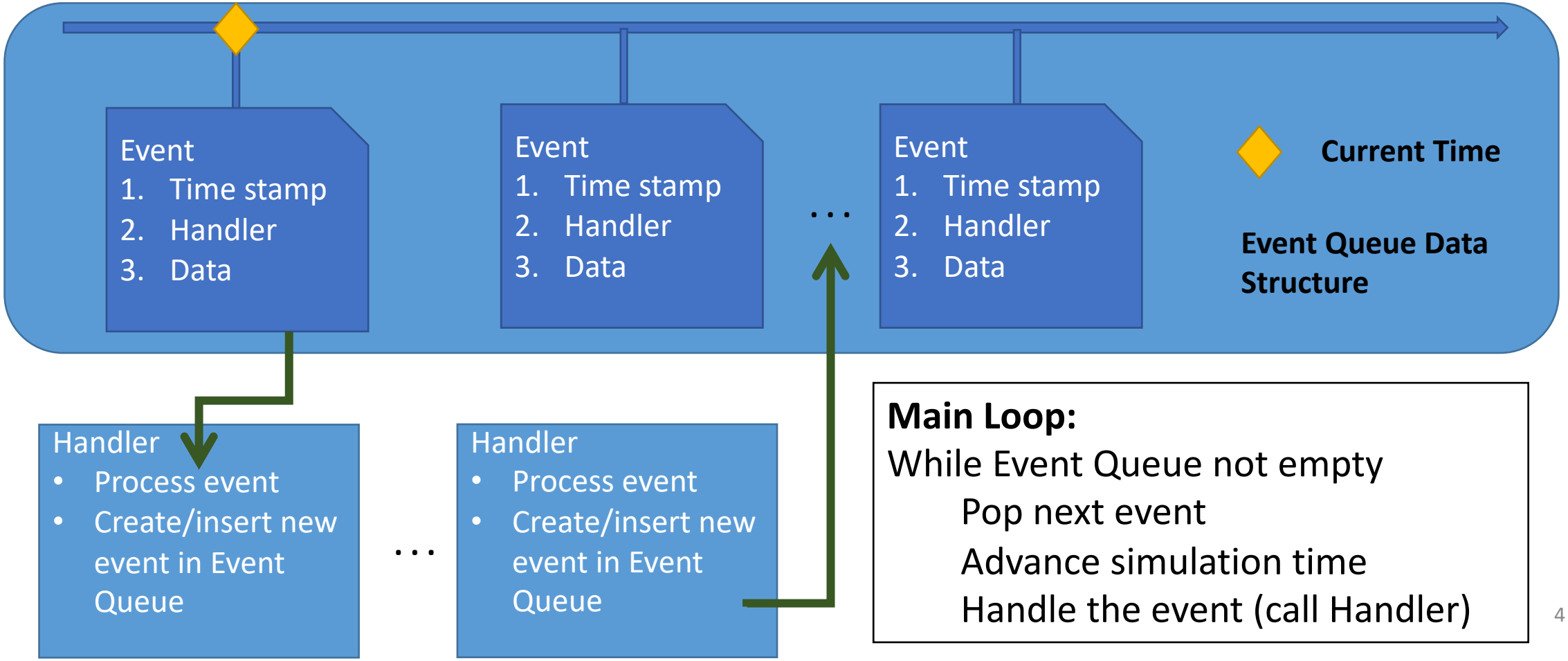
Outline

- Background
- Performance Prediction Toolkit (PPT)
- Interconnection Network Models
- Conclusions and Path Forward

Parallel Discrete Event Simulation

Discrete-Event Simulation (DES)

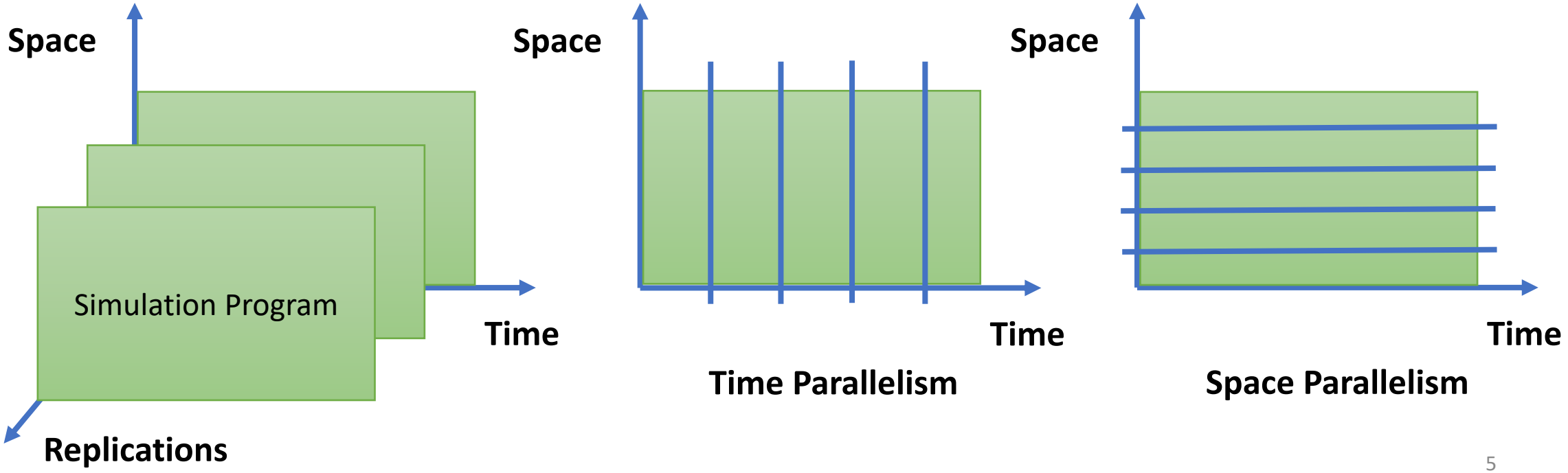
- Mimic operations over discrete instances of time (events)



Parallel Discrete-Event Simulation (PDES)

- Run DES in parallel
- Two purposes:
  - Reduce simulation time
  - Increase modeling size

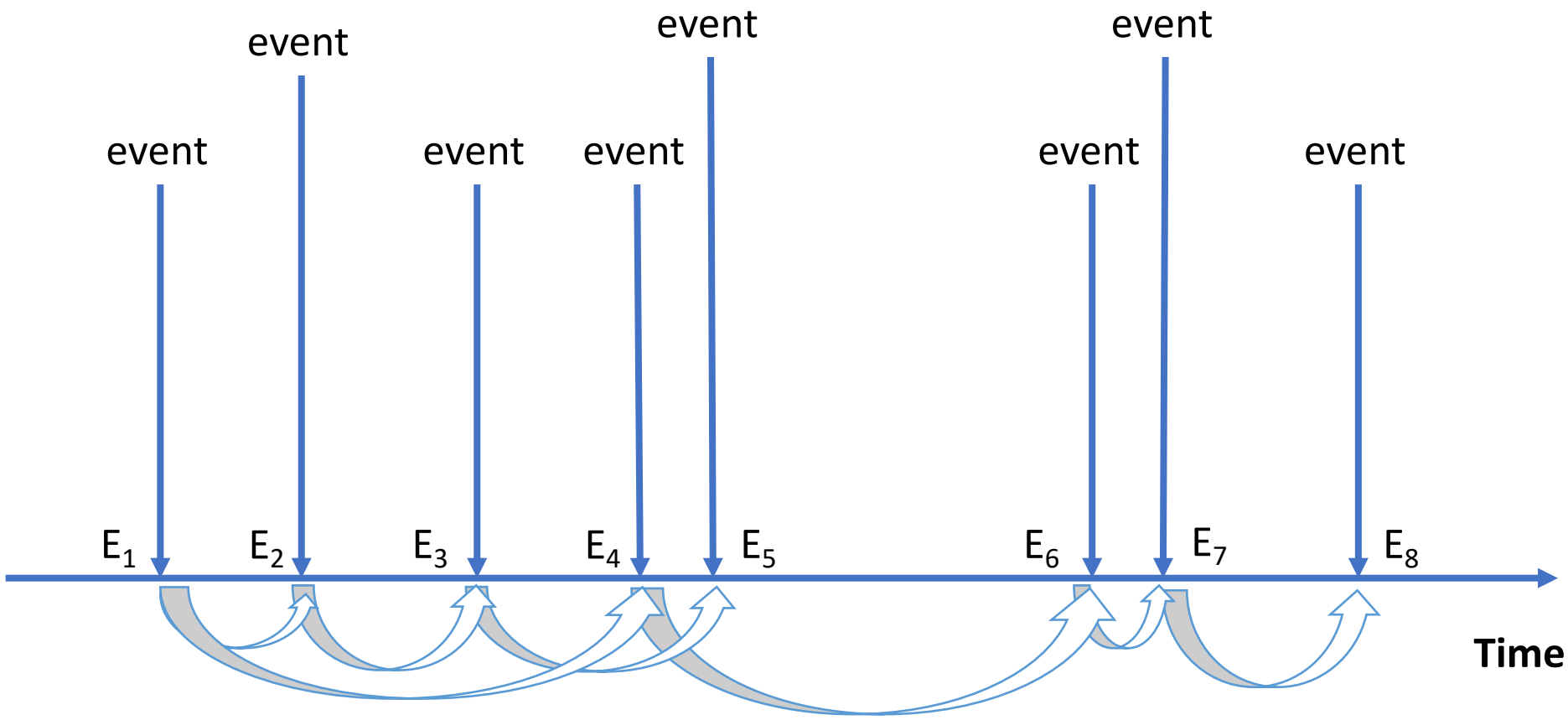
- More specifically:
  - Model large and complex systems
  - Design and parameter exploration



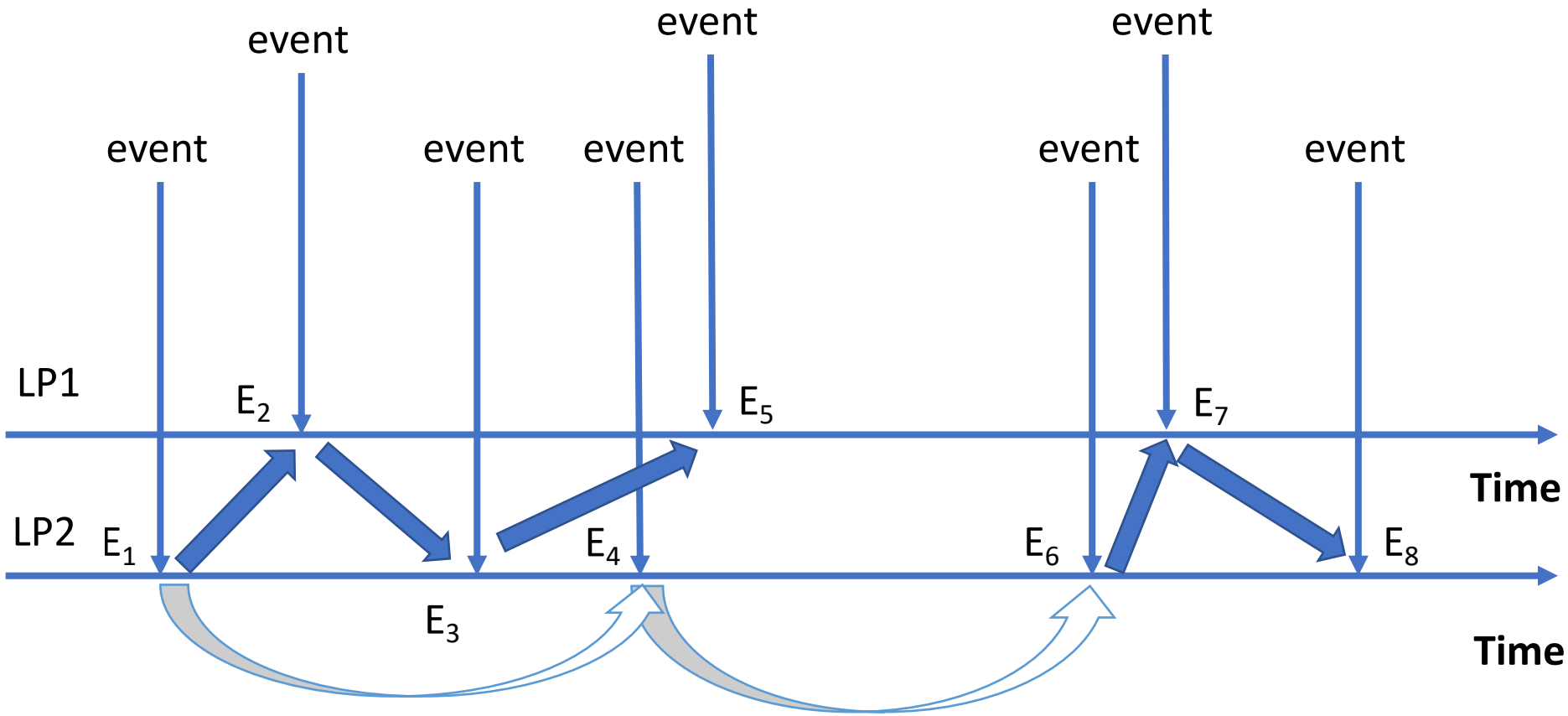
Spatial Decomposition

- Divide simulation model into sub-models that can be distributed to different processors
  - Each sub-model is called a Logical Process (LP)
- Each LP maintains its own event-list
  - No global simulation clock!
- LPs communicate via explicit messages

Spatial Decomposition



Spatial Decomposition (Contd.)



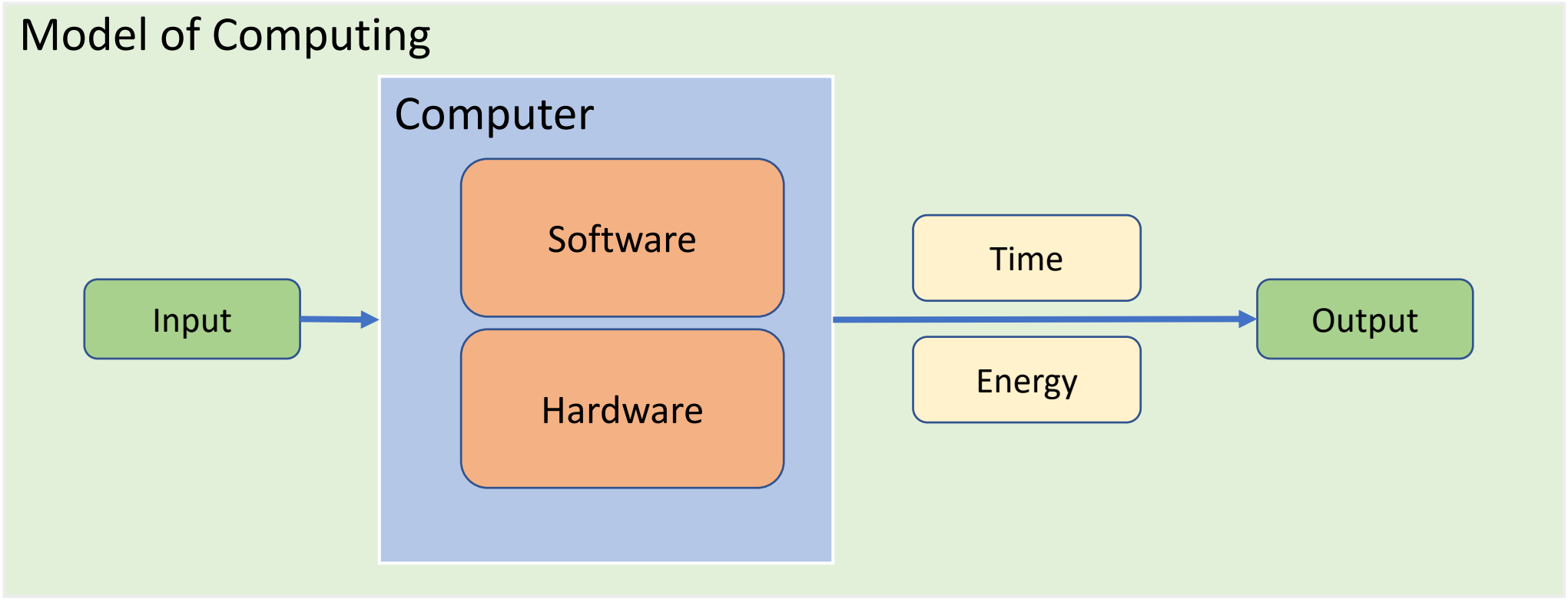
At each LP, the events shall be processed in non-decreasing timestamp order!

Parallel Simulation Synchronization

- An algorithm needed to ensure causality constraint (processing events in timestamp order)
  - Fundamental problem for PDES
- CMB algorithm
  - R.E. Bryant, *MIT Technical Report*, 1977
  - Jayadev Misra and K. Mani Chandy , *IEEE Transactions on Software Engineering*, 1979
  - **Conservative synchronization:** execute an event only ensuring that causality error never happens
- Time Warp Algorithm:
  - David Jefferson, *ACM Transactions on Programming Languages and Systems*, 1985
  - **Optimistic synchronization:** rolling back LP upon causality error (via reverse computation)

Performance Prediction Toolkit (PPT)

Codesign Modeling to Predict Performance of SW/Computational Methods on **Novel** HW Platforms



Key idea → Explore SW and HW design spaces and assess algorithmic variations

Predicted performance measured

Design spaces

Codesign Performance Modeling

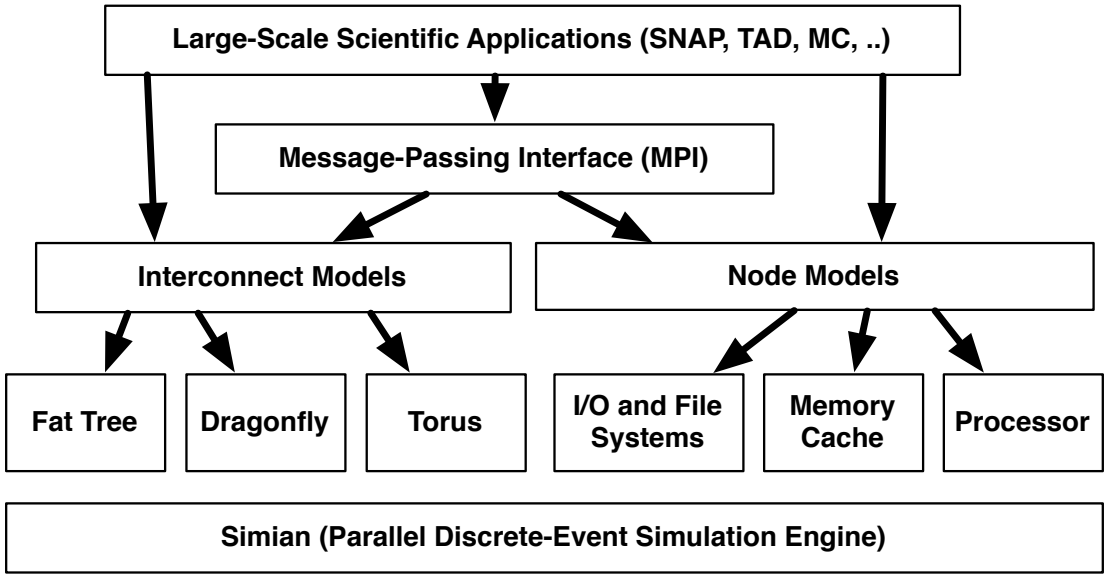
- Hardware resources are modeled as **Entities**
  - Compute nodes, interconnection networks, processors, memory systems
- Applications and algorithms are modeled as **Processes** running on entities
  - Processes independently advance in simulation time (sleep for computation or resource usage)
- **Selective refinement of modeling** details based on suspected performance bottlenecks both in hardware and software

Selective Refinement Modeling

- Goal: maintain modeling scalability for large, complex systems
  - We are interested in performance of parallel applications (physics code) running on petascale and exascale systems
- To find the “right” level of modeling details (just enough to answer the research questions) is an iterative process:
  1. Start from coarse-level models
  2. Gather experiment results
  3. Identify components as potential performance bottlenecks
  4. Replace those components by plugging in more refined models
  5. Go to #2 until satisfied

PPT Model Architecture

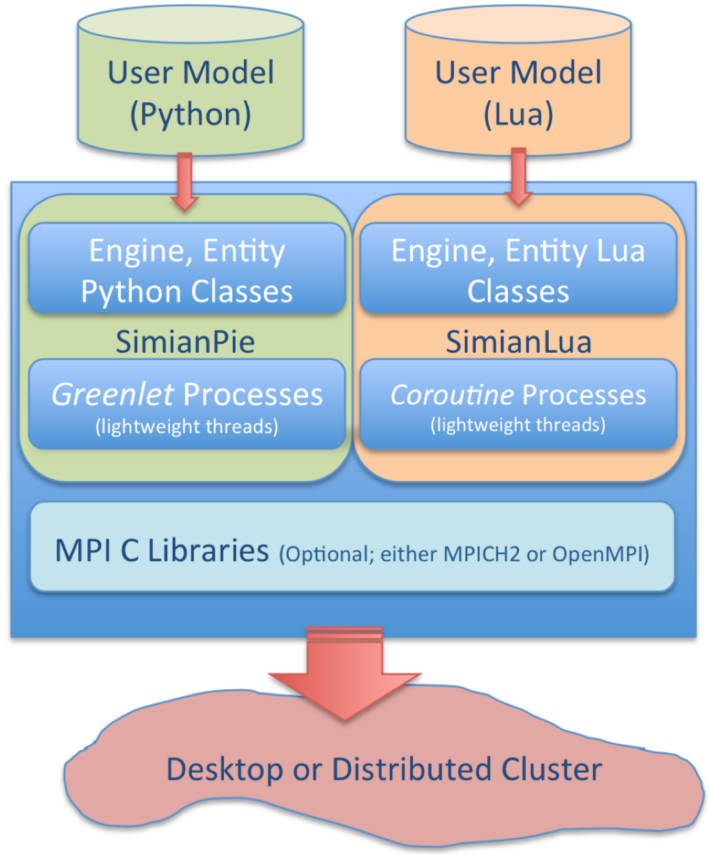
- **Simian** – parallel discrete-event simulation engine
- **Configurable hardware models**: clusters, compute nodes, processes/cores, accelerators (GPU), interconnect models, parallel file systems
- **Middleware models:** MPI, OpenMP
- **Application library**: benchmark applications (PolyBenchSim, ParboilSim), production applications (SNAPSim, SPHSim, SpecTADSim)



- **Data**: application instrument data (PolyBench, SNAP, SPH, CloverLeaf), hardware specs data (Mustang, Haswell, IvyBridge, SandyBridge, Vortex), communication data (DOE mini-apps)

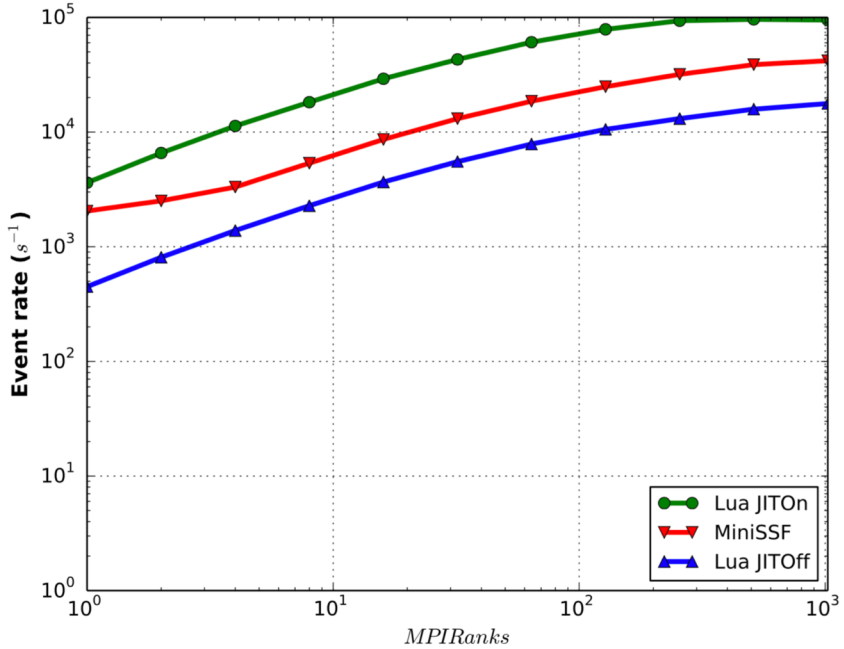
Simian: Parallel Discrete Event Simulation Engine

- Open source, general purpose parallel discrete-event library
- Independent implementation in two interpreted languages: Python and Lua, with optional C librarie (such as MPI)
- Minimalistic design: LOC=500 with 8 common methods
- Simulation code can be Just-In-Time (JIT) compiled to achieve very competitive event-rates, outperforming C++ implementation in some cases



Performance Comparison

- La-PDES benchmark: has 8 test case scenarios with 12 parameters
- 2 Simian implementations and MiniSSF (C++)



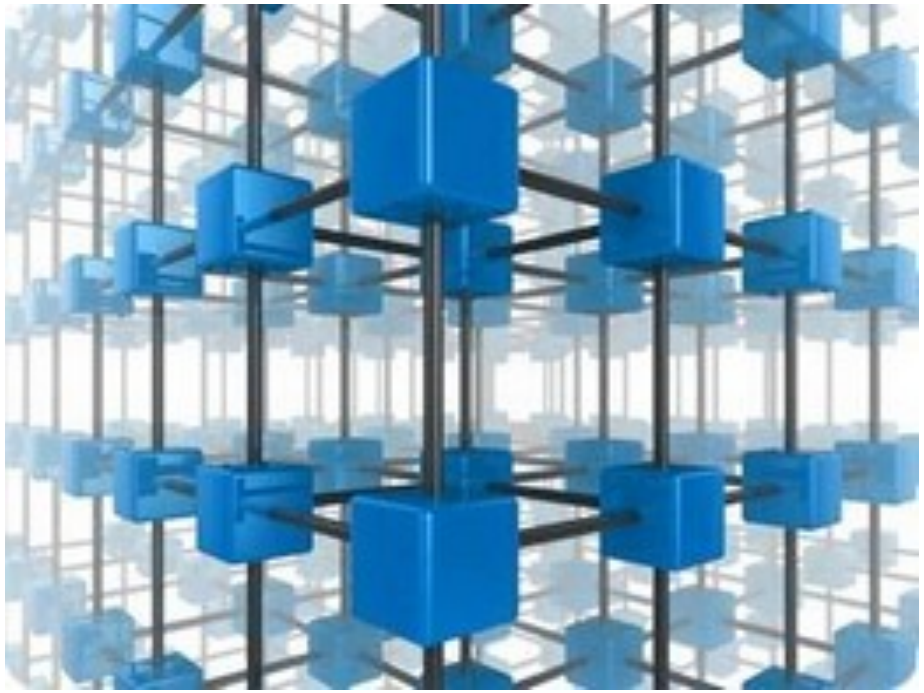
On a mid-size cluster of 1024 core

- Simian scales very well with available MPI ranks (upto 1024 ranks)
- SimianLUA performs 3x better than MiniSSF C++ engine

Interconnection Network Models

Interconnection Network

- Interconnect is a critical component of extreme-scale HPC architectural design
- Interconnection network model is essential for performance evaluation studies
  - Need to be scalable, efficient, and accurate
- Common interconnect topologies
  - Torus (e.g., Cray's Gemini)
  - Dragonfly (e.g., Cray's Aries)
  - Fat-tree (e.g., Mellanox Infiniband)



Existing Interconnection Network Models

- **BigSim (UIUC):** for performance prediction of large-scale parallel machines (with relatively simple interconnect models), implemented in Charm++ and MPI, shown to scale up to 64K ranks intially

- **xSim (ORNL):** scale to 128M MPI ranks using PDES with lightweight threads, include various interconnect topologies (high-level models, e.g., network congestion omitted)

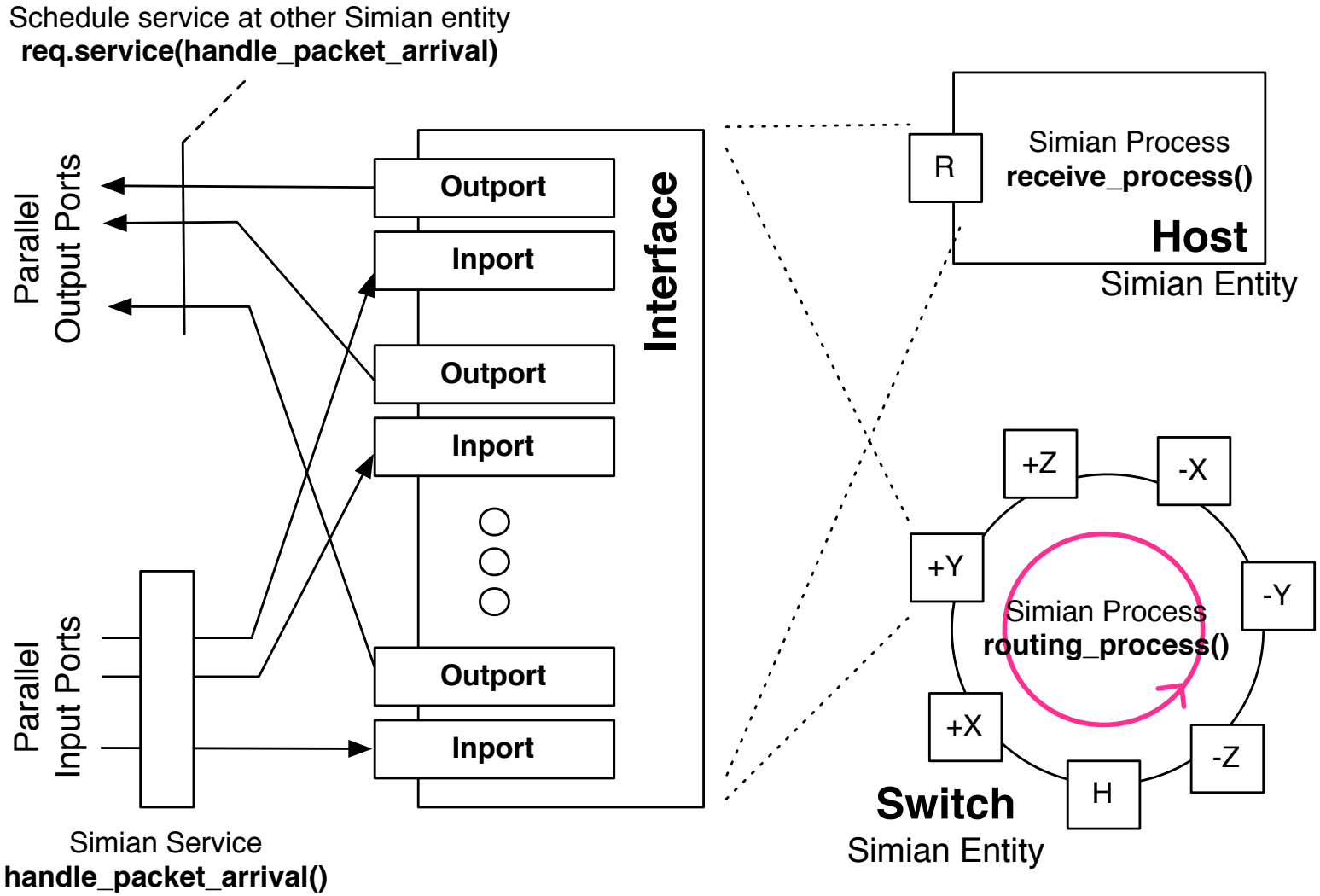
- **SST and SST Macro (SNL):** a comprehensive simulation framework, separate implementation, one intended with cycle-level accuracy and the other at coarser level for scale

- **CODES (ANL):** contains interconnect models and storage systems, built on ROSS using reverse computation simulation that also scales well

Our Focus on **Rapid** Performance Prediction

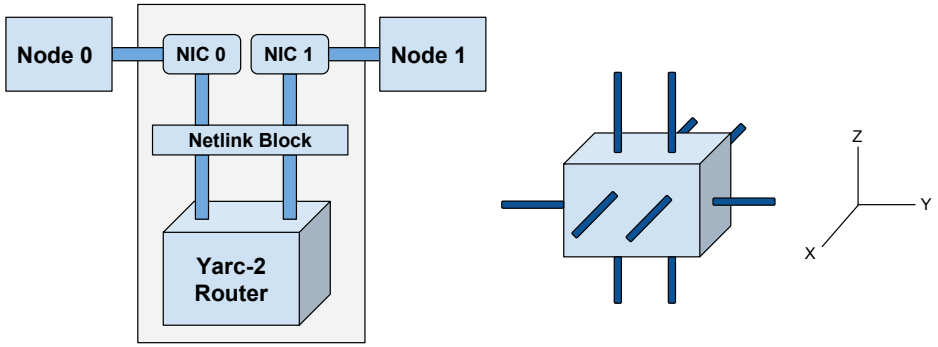
- Easy integration with selective models with varying abstraction
- Easy integration with physics applications
- Performance and scale
- Packet-level as opposed to phit-level
  - For performance and scale (speed advantage in several orders of magnitude, allow for full scale models, sufficient accuracy)
- Emphasis on production systems
  - Cielo, Darter, Edison, Hopper, Mira, Sequoia, Stampede, Titan, Vulcan, ...
- Seamlessly integrated with MPI
  - Implementation of all MPI common functions

Interconnect Model



Cray's Gemini Interconnect

- 3D torus direct topology
- Each building block:
  - 2 compute nodes
  - 10 torus connections
    - $\pm X*2, \pm Y, \pm Z*2$
- Routing
  - Adaptive dimension-order routing



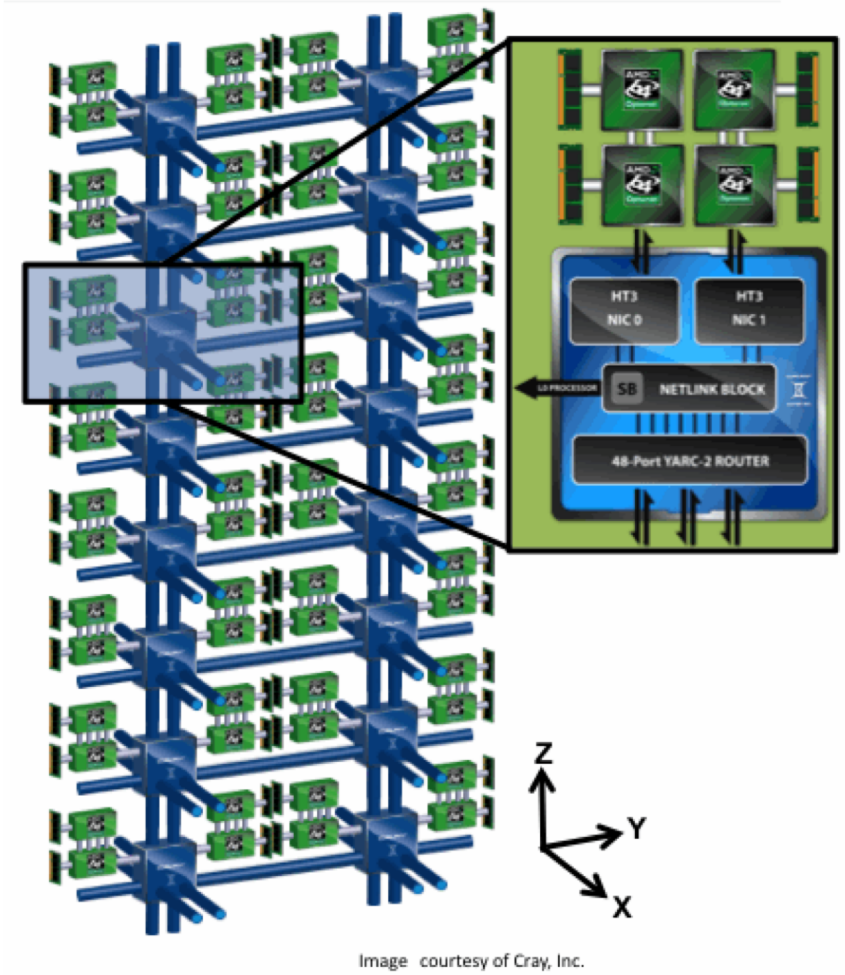
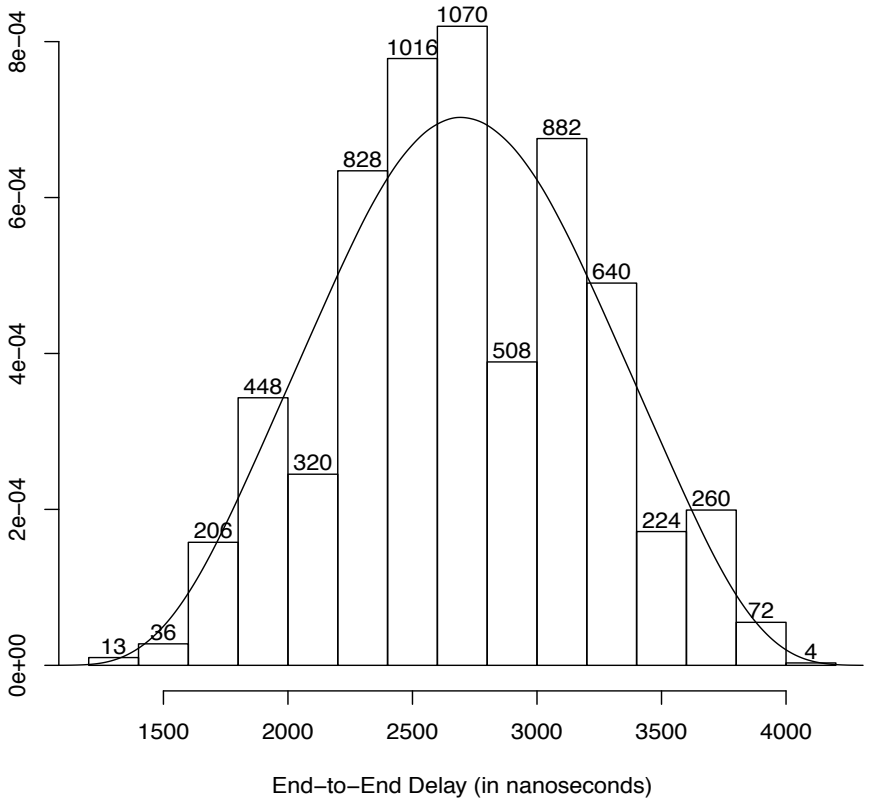
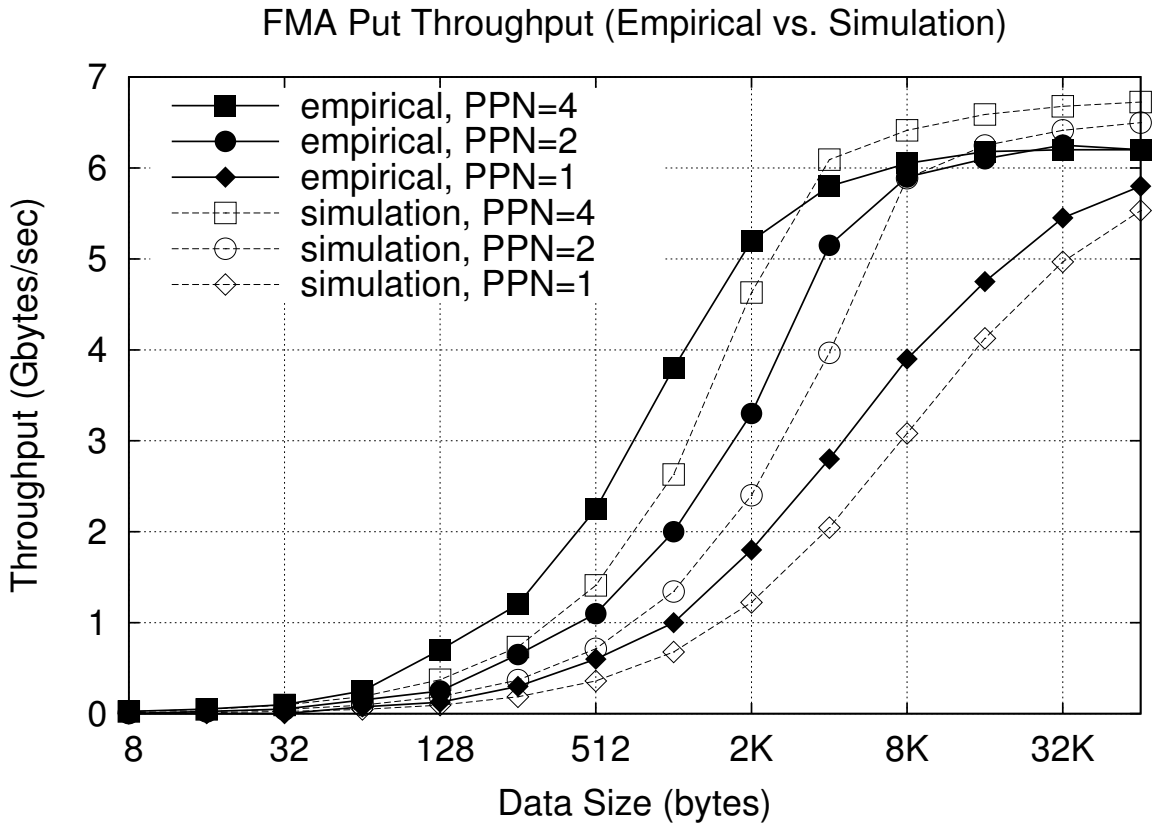


Image courtesy of Cray, Inc.

Gemini Validation

Compared against empirical results from Hopper @NERSC

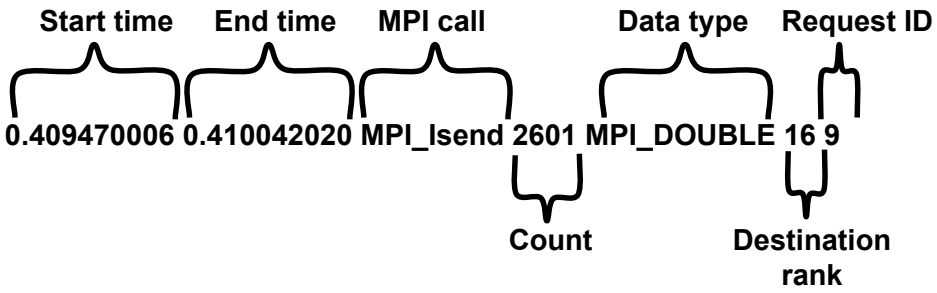


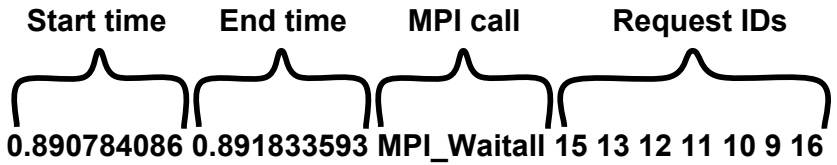


Inter-node latency: 1.27µs between nearest nodes
3.88µs between nearest nodes

Trace-Driven Simulation

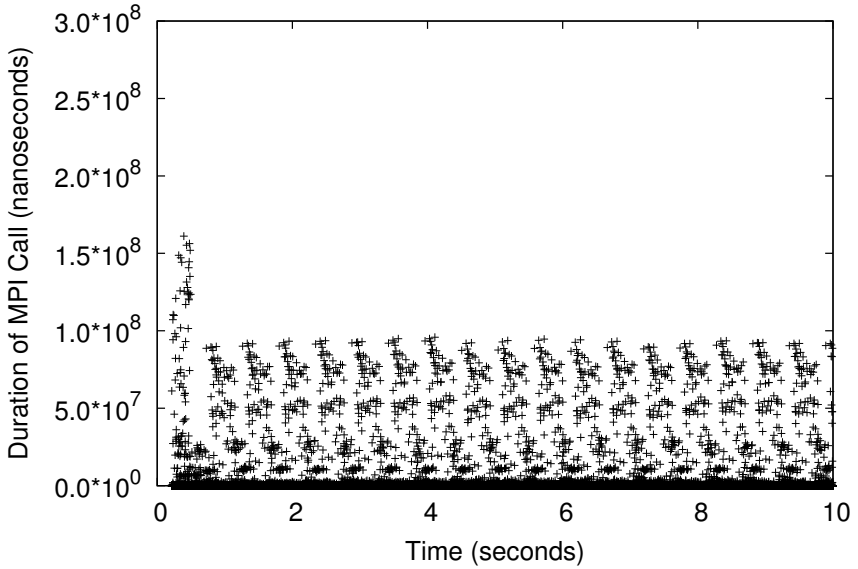
- Use application communication traces for different DOE mini-apps (from NERSC)
- For this experiment, we use:
  - LULESH mini-app from ExMatEx
    - Approximates hydro-dynamic model and solves Sedov blast wave problem
  - 64 MPI processes
- Run trace for each MPI rank:
  - Start MPI call at exactly same time indicated in trace file
  - Store completion time of MPI call
  - Compare it with the completion time in trace file



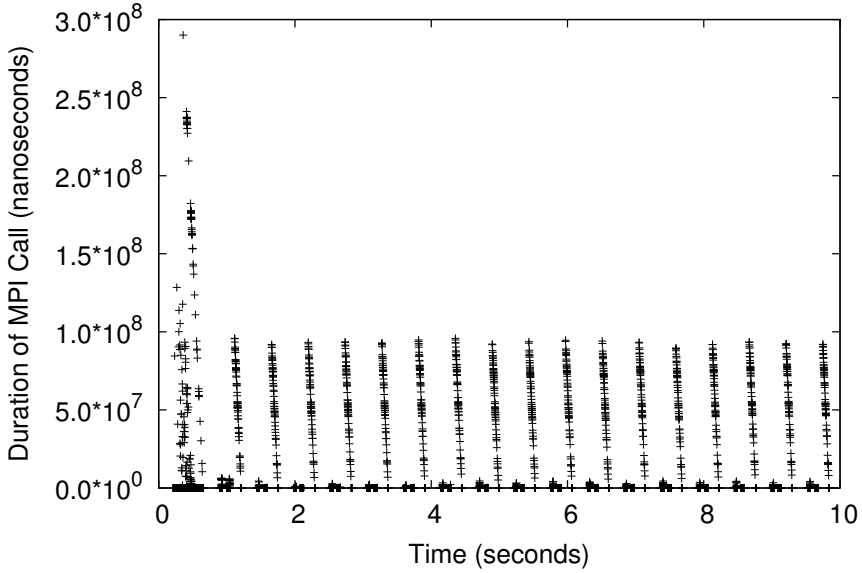


Trace-Driven Simulation (Contd.)

- MPI calls:
  - MPI_Isend, MPI_Irecv, MPI_Wait (123,336 each)
  - MPI_Waitall (12,864)
  - MPI_Allreduce (6,336)
  - MPI_Barrier, MPI_Reduce (64 each)
- Simulation runtime 55 seconds



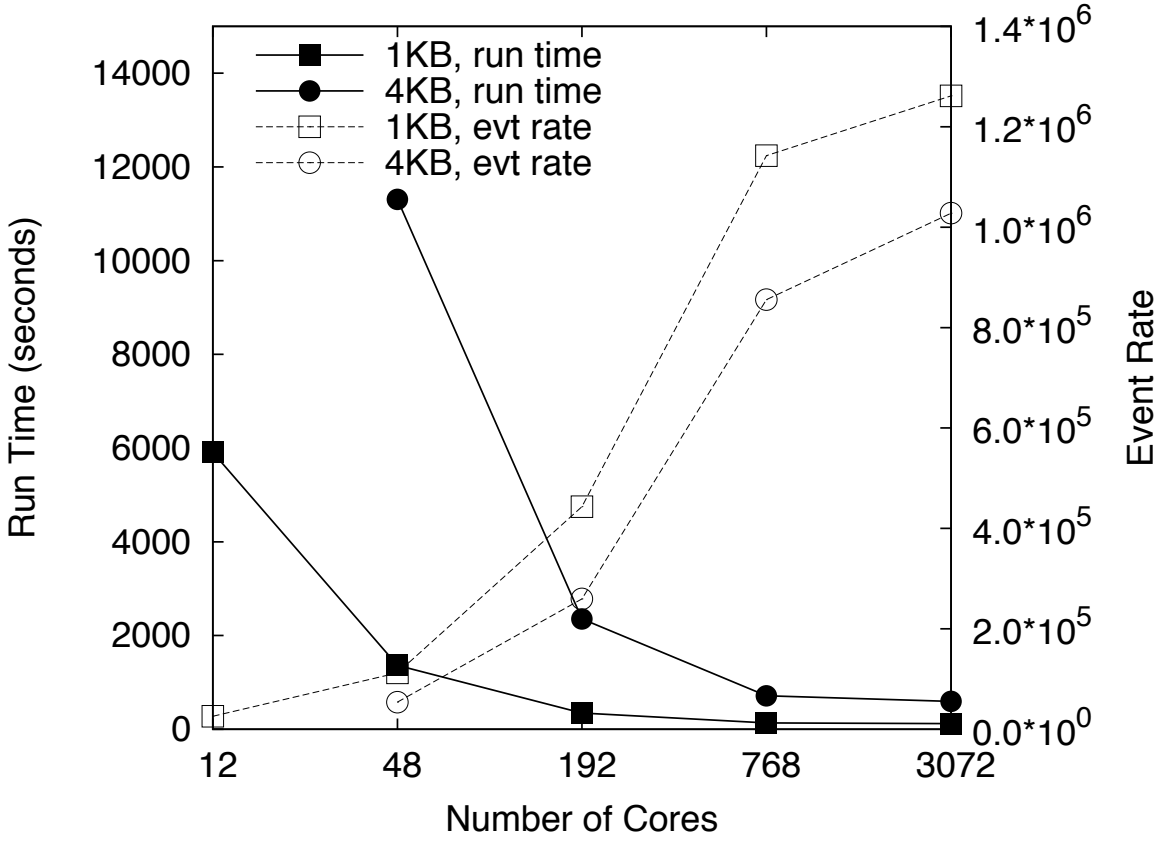
Trace output



Simulation output

Parallel Performance

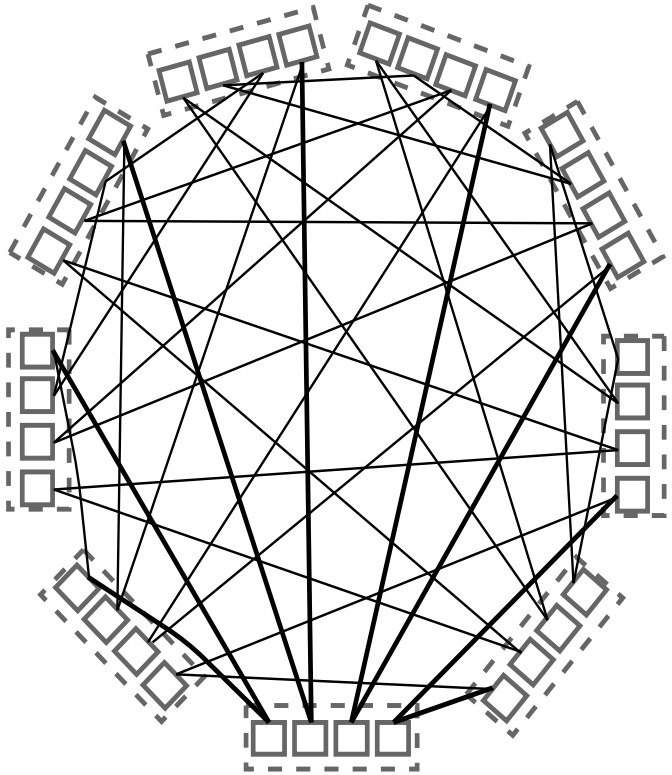
- A 1500-node cluster located at Los Alamos National Laboratory
- We varied number of compute nodes, from 1 (12 cores) to 256 nodes (3,072 cores)
- MPI_Allreduce, with different data size (1K or 4K bytes)

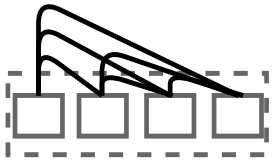


Three times event-rate of an optimized C++ simulator (MiniSSF)

Cray's Aries Interconnect

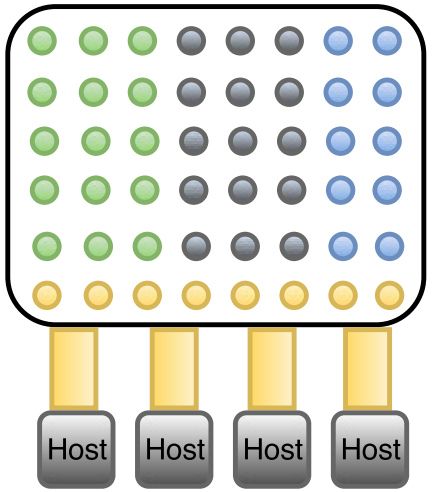
- Dragonfly: A cost-efficient topology
  - Nodes grouped together (high-radix router)
  - Economical, optical signaling technologies for distance routing
- Connections
  - Local link (completely connected)
  - Global link (consecutive connected)
- Routing
  - Minimal routing (benign traffic pattern)
  - Valiant routing (adversarial traffic pattern)

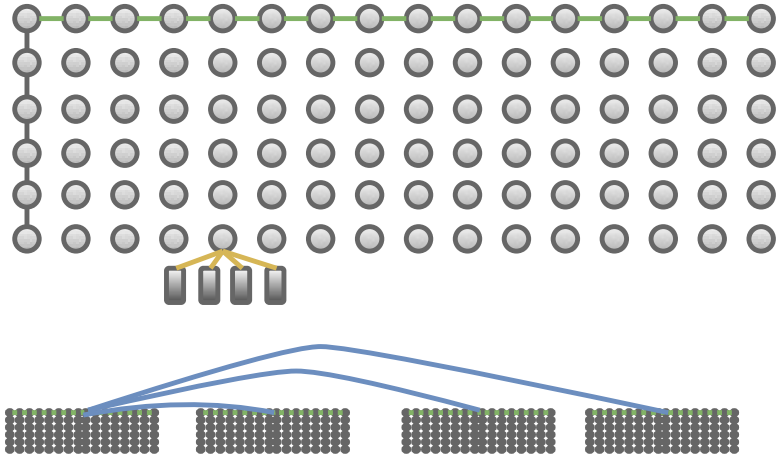




Cray's Aries Interconnect (Contd.)

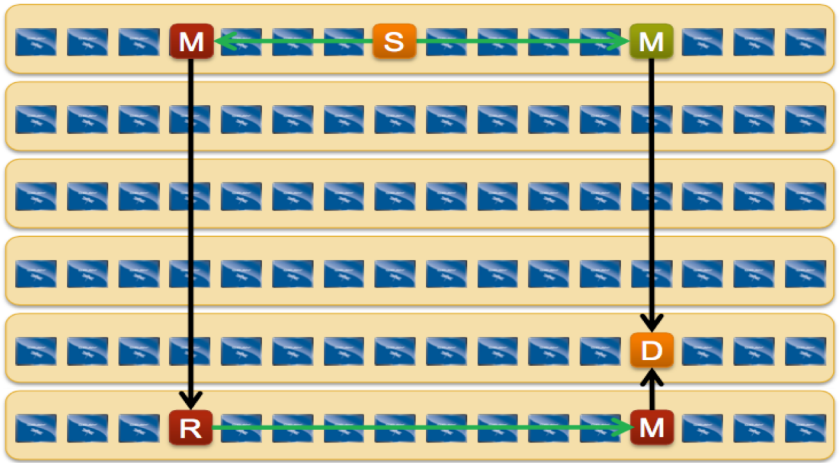
- Each group
  - 2 cabinets
  - 6 chassis
  - 96 Aries blades
- Connections
  - First dimension (green links)
    - Connects each blade in a chassis to other 15 blades in the same chassis
  - Second dimension (black links)
    - Connects each blade to 5 other blades of the other chassis
  - Third dimension (blue links)
    - Ten connections per blade to other groups

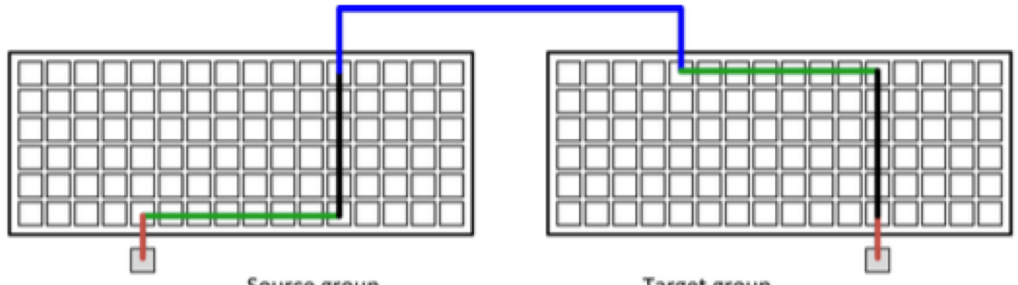


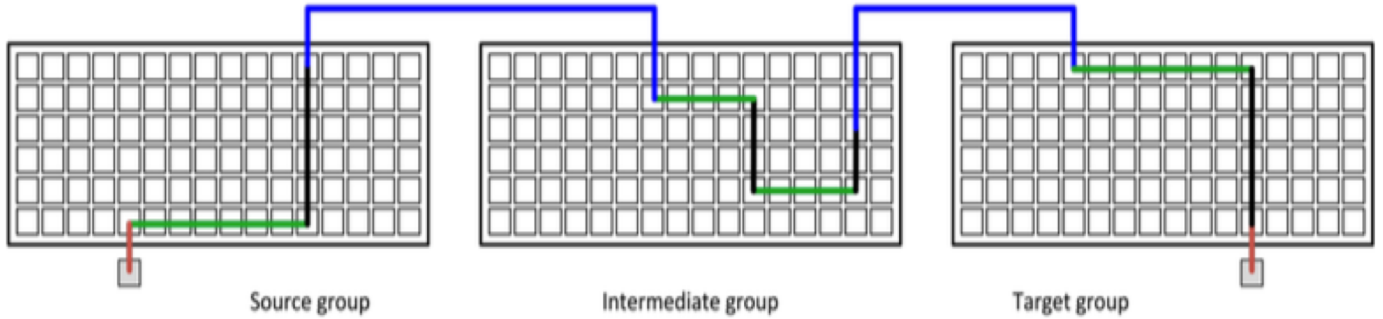


Cray's Aries Interconnect (Contd.)

- Intra-group routing
  - MIN requires
    - Two hops among switches
  - VAL requires
    - Four hops among switches
- Inter-group routing







MPI Example using Dragonfly Topology

```
# helloworld.py :- use mpi as simple as possible
# test dragonfly topology

def main(mpi_comm_world, msg):
    n = mpi_comm_size(mpi_comm_world) # total num of ranks
    p = mpi_comm_rank(mpi_comm_world) # rank of this process

    print("%f: myapp: rank %d sends msg to rank %d" %
          (mpi_wtime(mpi_comm_world), p, (p+1)%n))
    succ = mpi_send((p+1)%n, "hello", 10, mpi_comm_world)
    print("%f: myapp: rank %d done sent: %s" %
          (mpi_wtime(mpi_comm_world), p, "success" if succ else "failed"))

    mpi_finalize(mpi_comm_world)

model_dict = {
    # simian parameters
    "model_name" : "helloworld", # name of the model
    "sim_time" : 1e6, # end simulation time in seconds
    "use_mpi" : True, # whether mpi is activated

    # parameters for interconnect model
    "intercon_type" : "Dragonfly",         # IMPORTANT: type is case sensitive
    "dragonfly" : configs.dragonfly_intercon,   # use sample dragonfly config

    # compute node parameters; IMPORTANT: type is case sensitive
    "host_type" : "Host",       # generic compute node with or without mpi installed

    # optional libraries/modules to be loaded onto compute nodes
    "load_libraries": set(["mpi"]),     # IMPORANT: lib names are case sensitie

    # mpi configurations (necessary if mpi is loaded)
    "mpiopt" : configs.aries_mpiopt,  # standard mpi config for Aries
}

cluster = Cluster(model_dict)
hostmap = range(10) # 10 mpi processes on separate hosts
cluster.start_mpi(hostmap, main, "hello world")
cluster.run()
```

MPI application

Hardware configuration

Run MPI

Model Configuration Example: Dragonfly

```
# dragonfly_config.py :- configuration for a simple dragonfly and
# Aries topo configs

dragonfly_intercon = {
    # This is a sample dragonfly topology taken as example from the
    # paper "Technology-Driven, Highly-Scalable Dragonfly Topology" by
    # William J. Dally

    "num_groups": 9,
    "num_switches_per_group": 50,
    "num_hosts_per_switch": 2,
    "num_ports_per_host": 7,
    "num_inter_links_per_switch": 2,
    "num_intra_links_per_switch": 49,

    "inter_group_delay" : 1.92e-6,
    "intra_group_delay" : 1.545e-6,
    "switch_host_delay" : 1.498e-6,
    "inter_group_bdw" : 8.16e10, # 10.2GB/s
    "intra_group_bdw" : 1.5e11, # 18.75 GB/s
    "switch_host_bdw" : 4.2e10, # 5.25 GB/s

    "intra_group_topology": "all_to_all",
    "inter_group_topology": "consecutive",
    "route_method": "minimal",
    "intra_link_dups": 1,
    "inter_link_dups": 1,
}
```

Interconnect model parameters

Delay and bandwidth parameters

Topology and routing options

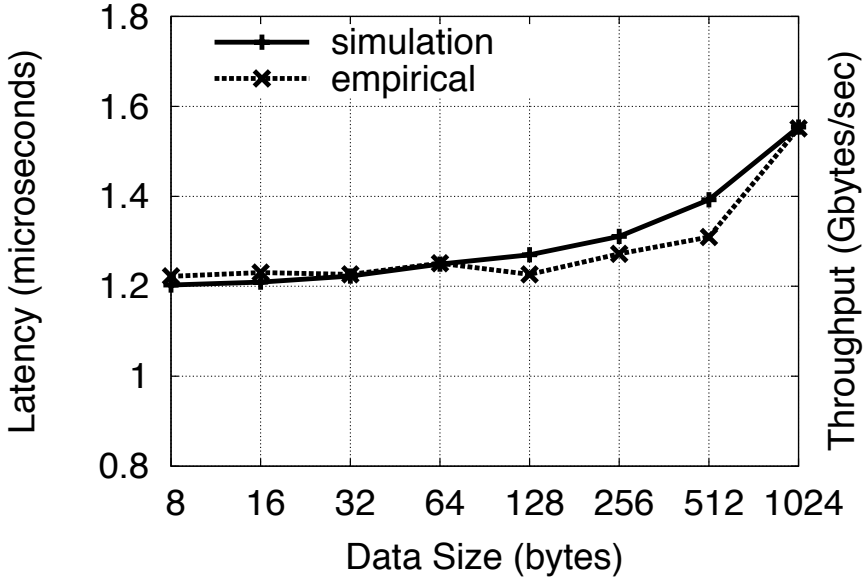
MPI Configuration Example

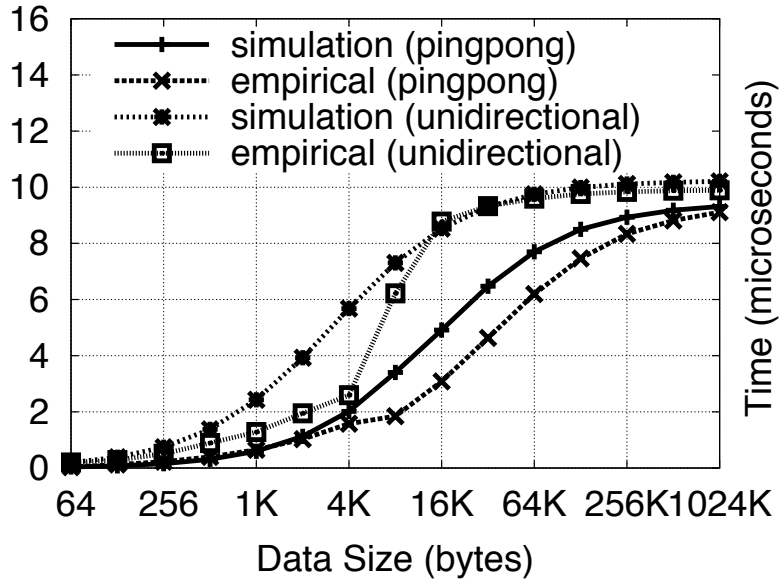
```
# MPI configuration for Cray's Aries network

# From "Cray XC series network" by Bob Alverson, Edwin Froese, Larry
# Kaplan and Duncan Roweth. Cray Inc., White Paper WP-Aries01-1112.

# In Cray XC, remote references are performed as gets/puts and atomic
# memory operations.  A put causes data to flow directly across the
# network to target node. When node issues commands, NIc packetizes
# these requests and issues the packets to the network. When packet
# reaches its destination, the destination node returns a response to
# the source.  Packet contains up to 64 bytes of data.

# Each 64-byte write (put) requires 14 request phits and 1 response
# phit.  Each 64-byte read (get) requires three request phit and 12
# response phits.

# The Aries NIC can perform a 64-byte read or write every five cycles
# (10.2GB/s at 800MHz). This number represents the peak injection rate
# achieveable by user processes.

aries_mpiopt = {
    "min_pktsz" : 0,
    "max_pktsz" : 64,
    "put_data_overhead" : 42, # 14 phits
    "put_ack_overhead" : 3, # 1 phit
    "get_data_overhead" : 36, # 12 phits
    "get_ack_overhead" : 9, # 3 phit3
}
```

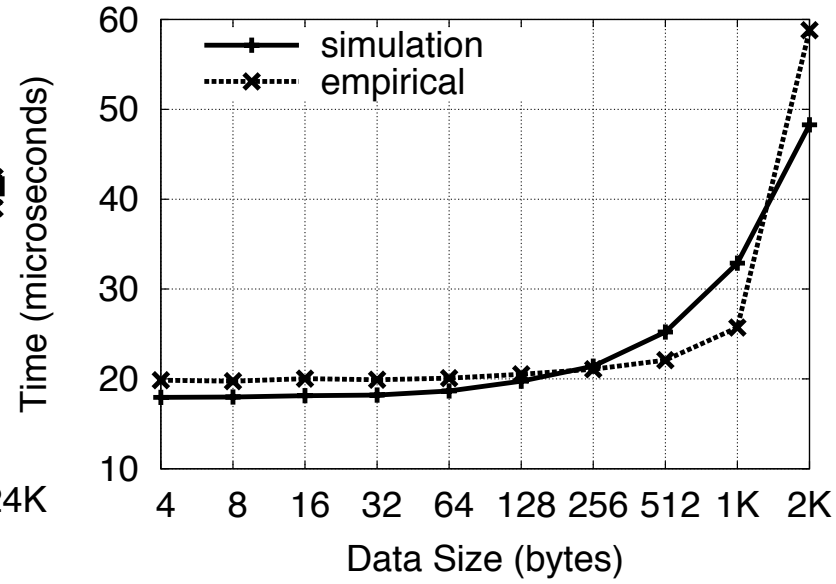
Aries Validation

- Trinity@LANL
- 9436 nodes, uses Cray XC40 system
- Average end-to-end latency and throughput between nodes

- Darter@University of Tennessee
- 748 nodes, uses Cray XC30 system
- MPI_Allreduce time

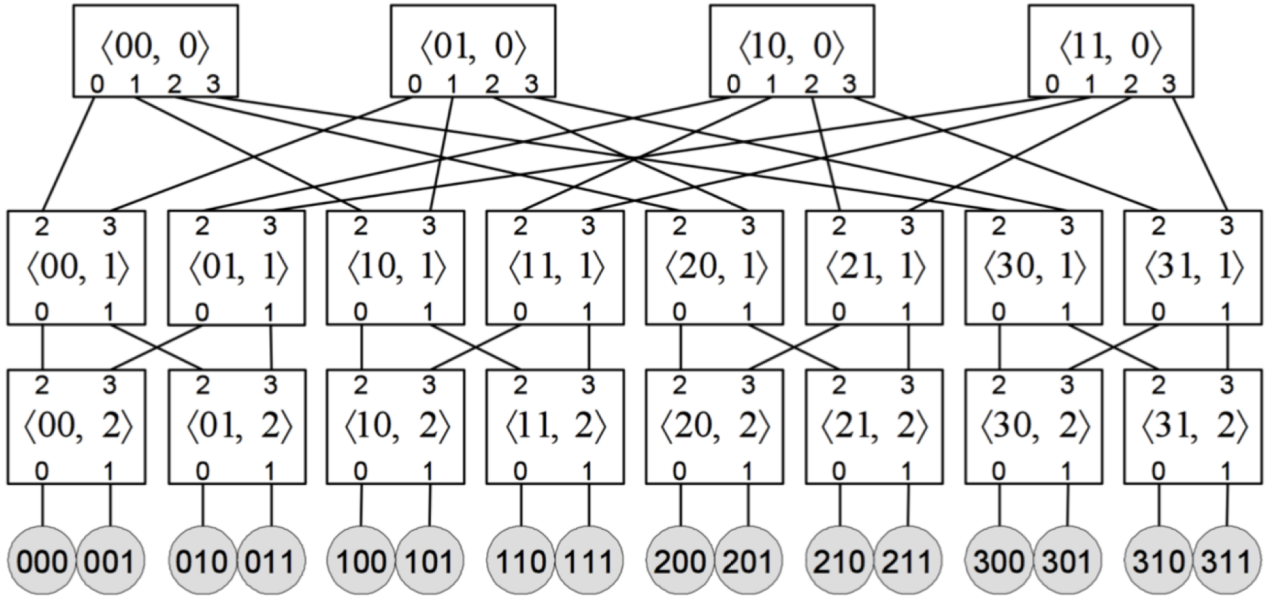






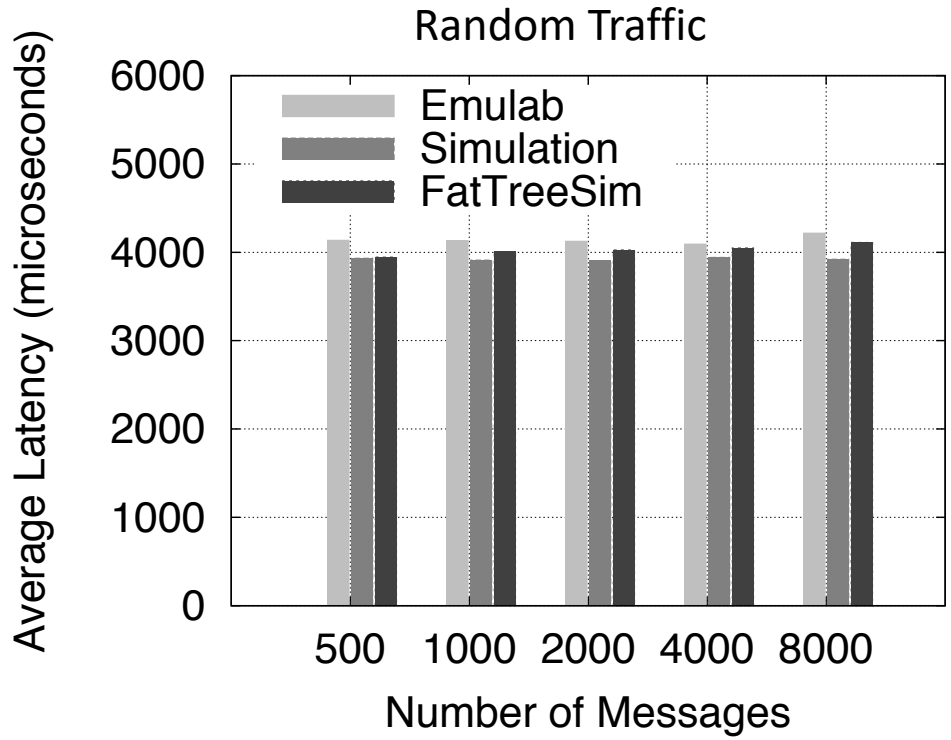
Fat-Tree Infiniband FDR

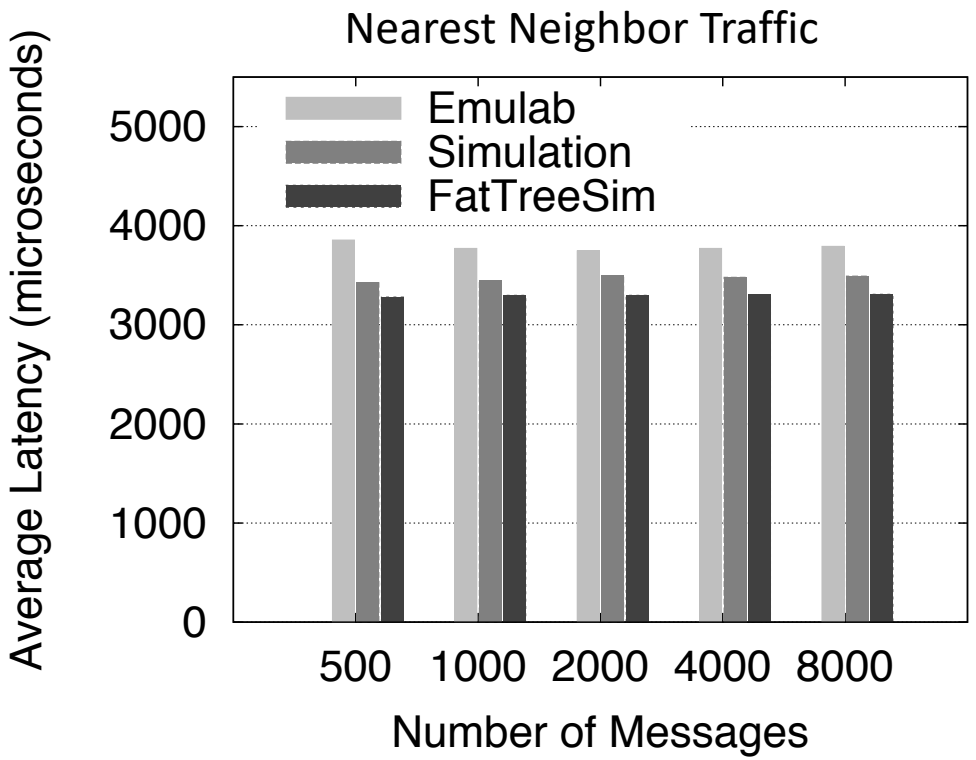
- An m-port n-tree
  - Height is (n+1)
  - $2(m/2)^n$ processing nodes
  - $(2n-1)(m/2)^{n-1}$ m-port switches
- Routing
  - Upward and downward phases
  - Valiant, ECMP, MLID
- Examples: Stampede
  - 6400 nodes
  - 56Gbps Mellanox switches
  - 0.7 µs uplink and downlink latency



Fat-Tree Validation

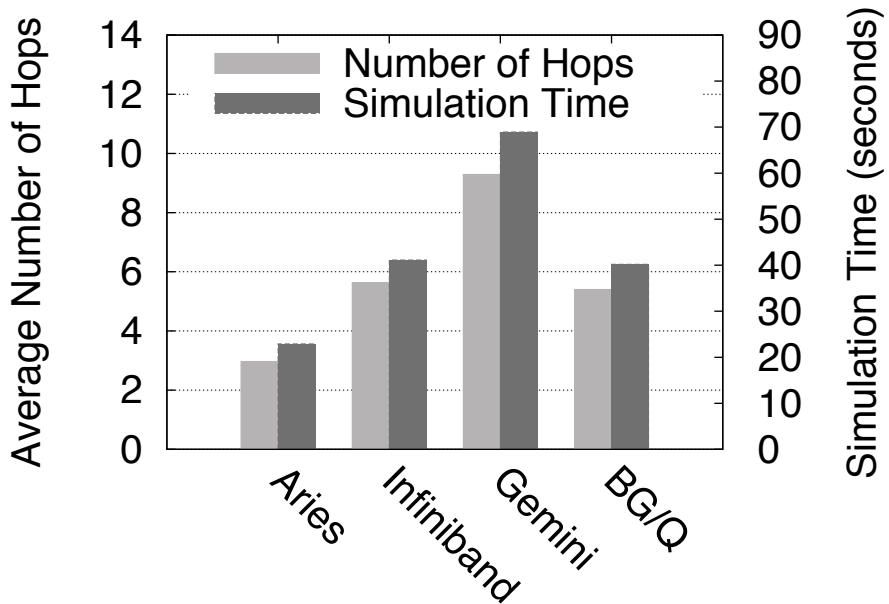
- Stampede@TACC
- 6400 nodes, uses fat-tree-based Infiniband system
- FatTreeSim: A ROSS-based simulator for fat-tree





Application Simulation

- Configurations
  - Aries: Trinity
  - Fat-tree: Stampede
  - Gemini: Hopper
    - 6,834 nodes connected via Gemini interconnect at 17X8X24
  - Blue Gene/Q: Mira
    - 49,152 nodes connected via Blue Gene/Q at 8X12X16X16X2
- Application: BigFFT
  - MPI_Alltoallv (400)
  - MPI_Barrier (500)
  - MPI_Group_free (4000)
  - MPI_Group_incl (2000)
  - MPI_Comm_create (2000)
  - MPI_Group_group (2000)

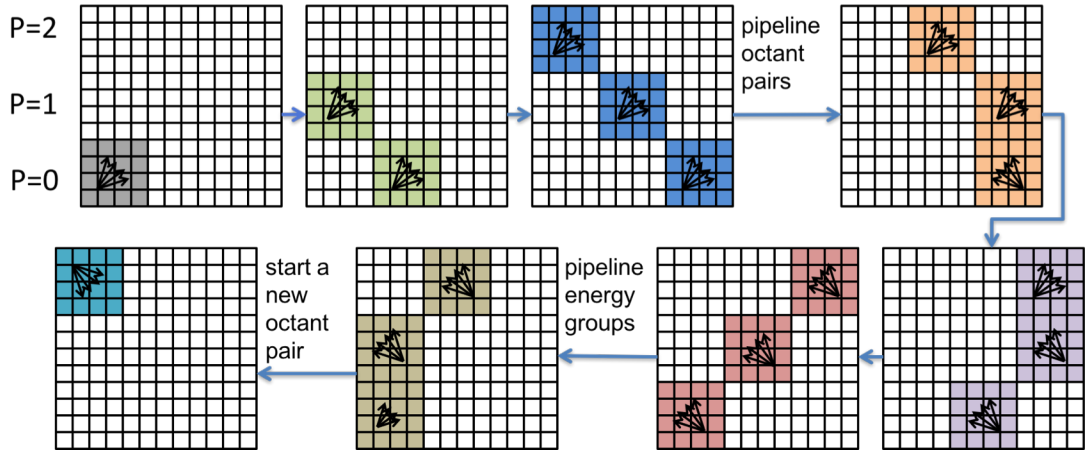


Case Study: SN Application Proxy

- SNAP is “mini-app” for PARTISN
- PARTISN is code for solving radiation transport equation for neutron and gamma transport
  - Structured spatial mesh (“cells”)
  - Multigroup energy treatment (“groups”)
  - Discrete ordinates over angular domain (“directions”)
  - Finite difference time discretization (“time steps”)

Application Model: SNAPSim

A 2-D illustration of the parallel wavefront solution technique



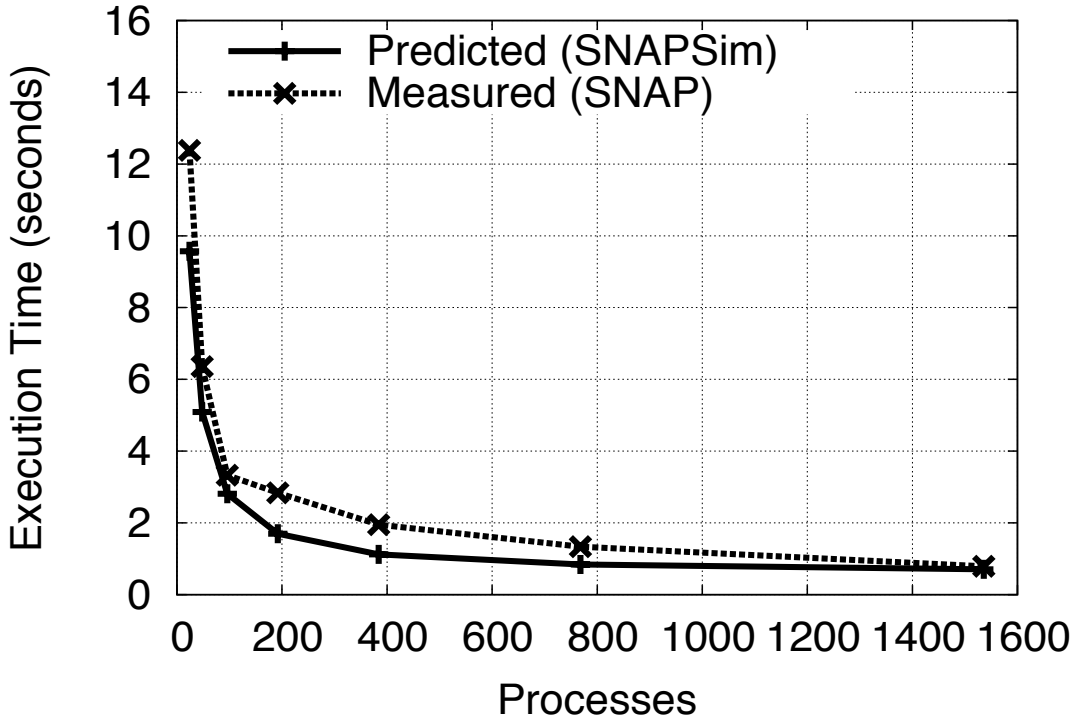
- Stylized version of actual applications
  - Focus on loop structures,

  important numerical kernels
- Use MPI to facilitate communication
- Use node model to compute time:
  - Hardware configuration based on clock-speed, cache-level access times, memory bandwidth, etc.
  - Predict the execution time for retrieving data from memory, performing ALU operations, and storing results

Strong Scaling Experiments

Edison Strong Scaling Study

64 × 32 × 48 Spatial Mesh 384 Angles, 42 Energy Groups



Conclusions and Path Forward

Conclusions

- Building a full HPC performance prediction model
  - Part of codesign process to test/improve code

- PPT – Performance Prediction Toolkit
  - MPI model and Interconnection network models (torus, dragonfly, fat-tree)
  - Validation (throughput, latency, execution time, trace-driven, applications)
  - Scalability testing (parallel performance)
  - Application modeling, such as SNAPSim

Path Forward

- **Interconnect performance bottleneck analysis**
  - Multiple applications on different interconnect topologies
  - Node size impact on applications

Path Forward (Contd.)

- **Improve interconnect performance**
  - Reduce application tail latency
  - Completely avoid (jump) the queue

Meet Slingshot: An Innovative Interconnect for the Next Generation of Supercomputers

OCTOBER 30, 2018 | BY STEVE SCOTT | LEAVE A COMMENT

dramatically reduced in the network. Slingshot's focus on bringing down tail latency (the latency that the slowest 1%, 0.1% or even 0.01% of packets experience) is key to making latency-sensitive and synchronization-heavy applications perform well. It can have a pretty dramatic impact on the performance of these applications.

Path Forward (Contd.)

- **Improve interconnect performance**
  - In Gemini, link faults occur "very frequently"
  - Analyze link faults in dragonfly
  - Fault-tolerant routing

"We observed that lane degrade event take place at a high rate of one event per minute"

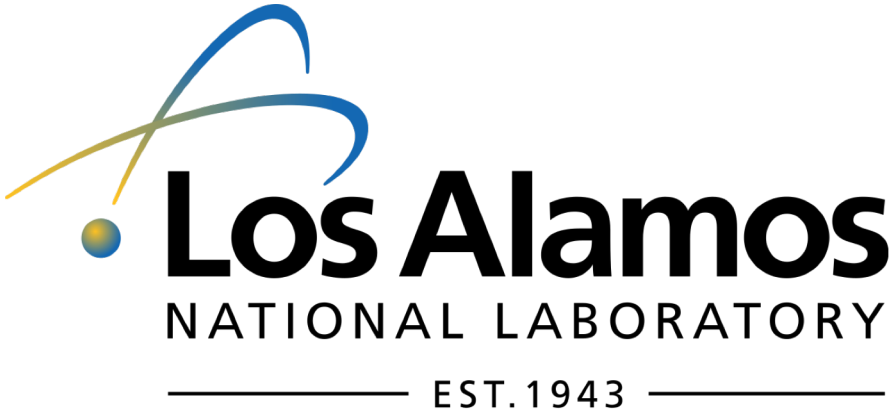
Kumar, M., Gupta, S., Patel, T., Wilder, M., Shi, W., Fu, S., ... & Tiwari, D. (2018, June). Understanding and analyzing interconnect errors and network congestion on a large scale HPC system. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (pp. 107-114). IEEE

Selected Publications

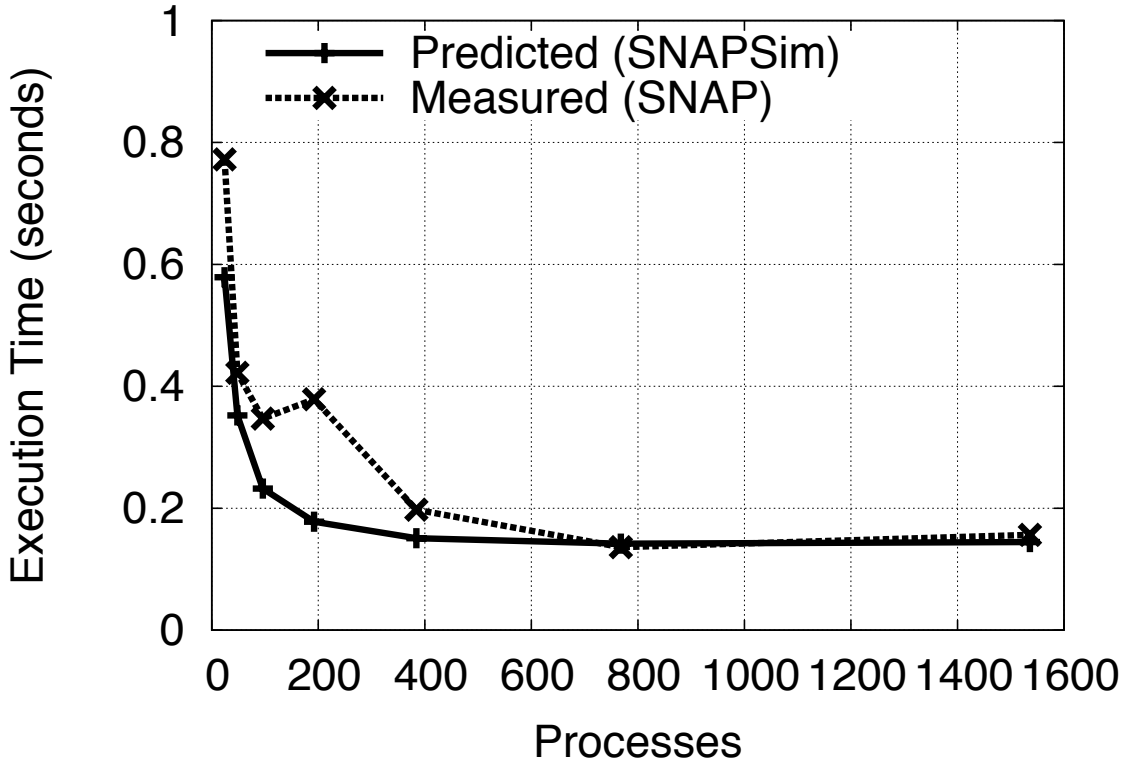
- Kishwar Ahmed, Jesse Bull, and Jason Liu, ”Contract-Based Demand Response Model for HPC Systems,” IEEE International Symposium on Parallel and Distributed Processing with Applications (**ISPA**), 2018.
- Mohammad A. Islam, Kishwar Ahmed, Hong Xu, Nguyen Tran, Gang Quan, and Shaolei Ren. Exploiting Spatio-Temporal Diversity for Water Saving in Geo-Distributed Data Centers. **IEEE Transactions on Cloud Computing (TCC)**, 2018.
- Kishwar Ahmed, Jason Liu, and Xingfu Wu, “An Energy Efficient Demand-Response Model for High Performance Computing Systems,” IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (**MASCOTS**), 2017.
- Kishwar Ahmed, Mohammad Obaida, Jason Liu, Stephan Eidenbenz, Nandakishore Santhi, and Guillaume Chapuis. “An integrated interconnection network model for large-scale performance prediction,” ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (**PADS**), 2016.
- Kishwar Ahmed, Jason Liu, Stephan Eidenbenz, and Joe Zerr. Scalable interconnection network models for rapid performance prediction of HPC applications. In High Performance Computing and Communications (**HPCC**), 2016.
- Kishwar Ahmed, Mohammad A. Islam, and Shaolei Ren. “A Contract Design Approach for Colocation Data Center Demand Response,” IEEE International Conference on Computer-Aided Design (**ICCAD**), 2015.

Thank You!
Questions?

Acknowledgements:



Strong Scaling Experiments



Edison Strong Scaling Study #1

32 × 32 × 48 Spatial Mesh 192 Angles, 8 Energy Groups

Backup Slide: MPI Functions

Table 1: Implemented MPI Functions

MPI_Send	blocking send (until message delivered to destination)
MPI_Recv	blocking receive
MPI_Sendrecv	send and receive messages at the same time
MPI_Isend	non-blocking send, return a request handle
MPI_Irecv	non-blocking receive, return a request handle
MPI_Wait	wait until given non-blocking operation has completed
MPI_Waitall	wait for a set of non-blocking operations
MPI_Reduce	reduce values from all processes, root has final result
MPI_Allreduce	reduce values from all, everyone has final result
MPI_Bcast	broadcast a message from root to all processes
MPI_Barrier	block until all processes have called this function
MPI_Gather	gather values form all processes at root
MPI_Allgather	gather values from all processes and give to everyone
MPI_Scatter	send individual messages from root to all processes
MPI_Alltoall	send individual messages from all to all processes
MPI_Alltoallv	same as above, but each can send different amount
MPI_Comm_split	create sub-communicators
MPI_Comm_dup	duplicate an existing communicator
MPI_Comm_free	deallocate a communicator
MPI_Comm_group	return group associated with communicator
MPI_Group_size	return group size
MPI_Group_rank	return process rank in group
MPI_Group_incl	create new group including all listed
MPI_Group_excl	create new group excluding all listed
MPI_Group_free	reclaim the group
MPI_Cart_create	add cartesian coordinates to communicator
MPI_Cart_coords	return cartesian coordinates of given rank
MPI_Cart_rank	return rank of given cartesian coordinates
MPI_Cart_shift	return shifted source and destination ranks