# Scalable Interconnection Network Models for Rapid Performance Prediction of HPC Applications

Kishwar Ahmed, **Jason Liu**, Florida International University, USA
Stephan Eidenbenz, Joe Zerr, Los Alamos National Laboratory, USA

18th IEEE Int'l Conf. on High Performance Computing and Communications (HPCC'16)
December 12-14, 2016 ♦ Sydney Australia

# Outline

- Motivation and Related Work
- Performance Prediction Toolkit (PPT)
- Interconnect Models and Validation
- SNAP Performance Study
- Conclusion

# Motivation: HPC Architecture Is Changing Rapidly

- End of processor scaling leads to novel architectural design
  - Changes can be transitional and disruptive
- HPC software adaptation is a constant theme:
  - *No code is left behind*: must guarantee good performance
  - Need high-skilled software architects and computational physicists
  - New apps: big data analytics
- Traditional methods are insufficient
  - Middleware libraries, code instrumentation, mini-apps…
- **Need modeling & simulation of large-scale HPC systems and applications**
  - And the systems are getting larger (exascale is around the corner)

# Related Work: HPC Simulations

- **Full system simulators**:
  - Simics, SimpleScalar, GEM5, COTSon, PTLsim, Asim
- **Analytical tools**:
  - TAU, Vampir, HPCToolkit, Paraver, PACE, ASPEN, Palm, GROPHECY
- **Processor/core simulators**: McSimA+, Zsim, Manifold
- **Memory system simulators** (DRAM, NVM, Cache):
  - DRAMSim, USIMM, DrSim, Ramulator, NVMain
- **NoC simulators**: BookSim, GARNET, DARSIM, HORNET, TOPAZ, DNOC
- **FPGA-based simulators**: Ramp Gold, HAsim, DART, Arete
- **Large-scale HPC simulators**: BigSim, xSim, SST, CODES

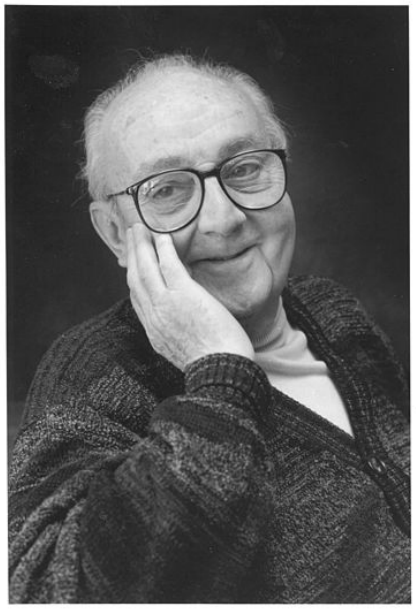
# Related Work: Interconnect Models

- **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
- **xSim (ORNL):** scale to128M 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):** focused on storage systems, built on ROSS using reverse computation simulation that scales well

# How about Full-Scale Cycle-Accurate Simulation of HPC Systems and Applications?

- It is unrealistic
  - Extremely high computational and spatial demand
  - Accurate models only limited to certain components and timescale
- And it is unnecessary
  - Modeling uncertainties greater than errors from cycle-accurate models
    - Languages, compilers, libraries, operating systems, …
    - System cross traffic
  - Design uncertainties defy specificity of cycle-accurate models

# "All models are wrong but some are useful"



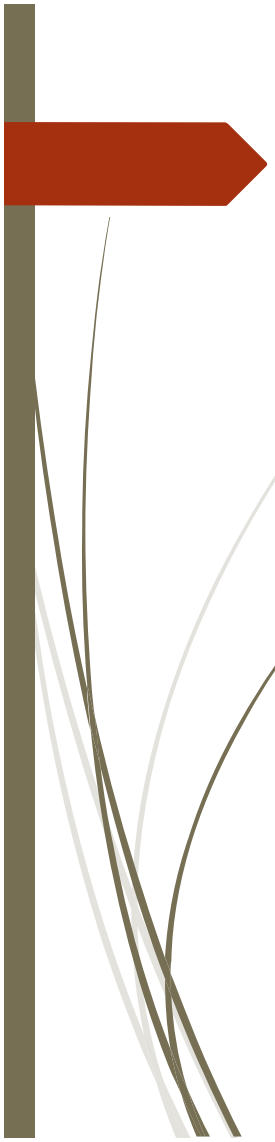
George Box (1919-2013)

- Managing expectations:
  - Ask what-if questions
  - Evaluate alternative designs
  - Explore parameter space
- Will models ever catch up with real-system refresh?
  - As valuable tools for prototyping new systems, new algorithms, new applications?

**Need tools for fast and accurate performance prediction enough**
**→ consider tradeoff**

# Modeling via Selective Refinement

- Maintain modeling scalability for large, complex systems
  - We are interested in performance of parallel applications (physics code) running on petascale and exascale systems
  - Having full-scale models at finest granularity is both unrealistic and unnecessary
- 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 satisfaction



# Our Goals for **Rapid** Performance Prediction

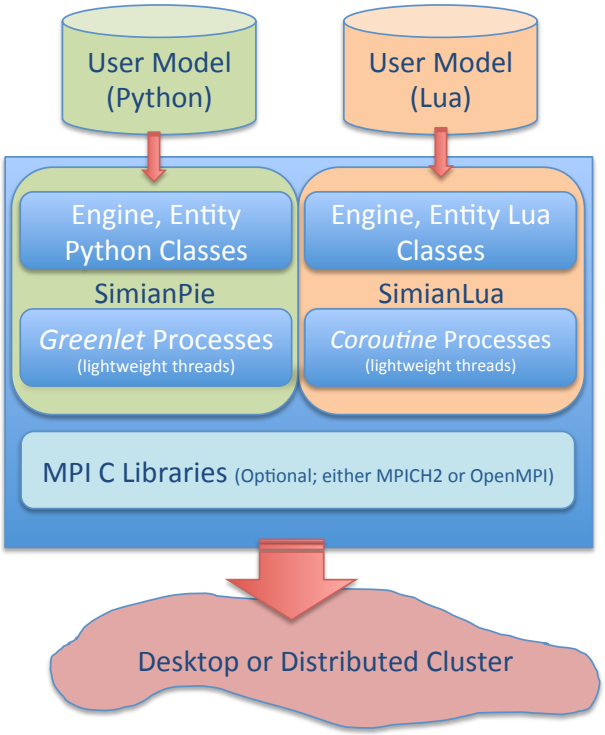
- Easy integration with selective models of varying abstractions
- Easy integration with applications (computation physics code)
- Short development cycle
- Performance and scale

# Performance Prediction Toolkit (PPT)

- **Make it simple, fast, and most of all useful**
  - Designed to allow rapid assessment and performance prediction of large-scale scientific applications on existing and future high-performance computing platforms.
  - PPT is a library of models of computational physics applications, middleware, and hardware that allows users to predict execution time by running stylized pseudo-code implementations of physics applications.
- "Scalable Codesign Performance Prediction for Computational Physics" project
  - **Simian** – parallel discrete-event simulation engine
  - **Configurable hardware models**: clusters, compute nodes, processes/cores, accelerators (GPU), interconnect, parallel file systems
  - **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)

# Simian: PDES using Interpreted Languages

- Open source, general purpose parallel discrete-event library
- Independent implementation in three interpreted languages: Python, Lua, Javascript
- **Minimalistic design: LOC=500 with 8 common methods (python implementation)**
- Simulation code can be Just-In-Time (JIT) compiled to achieve very competitive event-rates, even outperforming C++ implementation in some cases
- Support process-oriented world view (using Python greenlets and LUA coroutines)



# Interconnection Network Models

- Common interconnect topologies
  - Torus (Gemini, Blue Gene/Q)
  - Dragonfly (Aries)
  - Fat Tree (Infiniband)
- Distinction from previous approaches:
  - **Emphasis on production systems**
    - Cielo, Darter, Edison, Hopper, Mira, Sequoia, Stampede, Titan, Vulcan, …
  - **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)
  - **Seamlessly integrated with MPI**

# MPI Model

- MPI uses Fast Memory Access (FMA) for message passing
  - FMA allows a maximum of 64 bytes of data transfer for each network transaction
  - Larger messages must be broken down into individual 64-byte transactions
- Messaging are performed as either GET or PUT operations (depending on the size of the message)
- A PUT operation initiates data flow from the source to the target node. When a packet reaches destination, a response from the destination is returned to the source
  - A PUT message consists of a 14-phit request packet (each phit is 24 bits)
  - Each request packet is followed by a 1-phit response packet (3 bytes) from destination to source
- A GET transaction consists of a 3-phit request packet (9 bytes), followed by a 12-phit response packet (36 bytes) with 64 bytes of data

# Integrated MPI Model

- Developed based on Simian (entities, processes, services)
- Include all common MPI functions
  - Simplistic implementation
  - Point-to-point and collective operations
  - Blocking and non-blocking operations
  - Sub-communicators and sub-groups
- Process-oriented approach
  - Can easily integrate with most application models
- Packet-oriented model
  - Large messages are broken down into packets (say, 64B)
- Reliable data transfer

**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 |

# MPI Example

```
from ppt import *

# config hopper (17x8x24 gemini interconnect)
model_cfg = { # a dictionary
  "intercon_type" : "gemini",
  "host_type" : "mpihost",
  "torus" : configs.hopper_intercon,
  "mpiopt" : configs.gemini_mpiopt,
}
model = HPCSim(model_cfg, ..)

# mpi main function, n is matrix dimension
def cannon(mpi_comm_world, n):
  ... # we describe this later

# start 16 mpi ranks, pass matrix dimension
model.start_mpi(range(16), cannon, 10000)

# simulation starts
model.run()
```

Hardware Configuration

MPI application

Run MPI

# MPI Example

```
# cannon's algorithm on matrix multiplication
def cannon(mpi_comm_world, n):
  p = mpi_comm_size(mpi_comm_world)
  id = mpi_comm_rank(mpi_comm_world)
  # use p, id to calc i, j, and neighbor ranks

  # time for reading/initing submatrics
  sleep(sometime) # proportional to m^2

  # shift A(i,j) left by i columns
  mpi_sendrecv(left_i, None, m*m*8,
               right_i, mpi_comm_world)
  # shift B(i,j) up by j rows
  mpi_sendrecv(up_j, None, m*m*8,
               down_j, mpi_comm_world)

  for r in range(sqrt(p)-1):
    # time for multiplying A(i,j) and B(i,j)
    sleep(sometime) # proportional to m^3

    # shift A(i,j) to the left
    mpi_sendrecv(left, None, m*m*8,
                 right, mpi_comm_world)
    # shift B(i,j) upward
    mpi_sendrecv(up, None, m*m*8,
                 down, mpi_comm_world)

  mpi_finalize(mpi_comm_world)
```

# 3D Torus – Cray's Gemini Interconnect

- 3D torus direct topology
- Each building block:
  - 2 compute nodes
  - 10 torus connections:

    ±X*2, ±Y, ±Z*2
- Examples: Jaguar, Hopper, Cielo, …

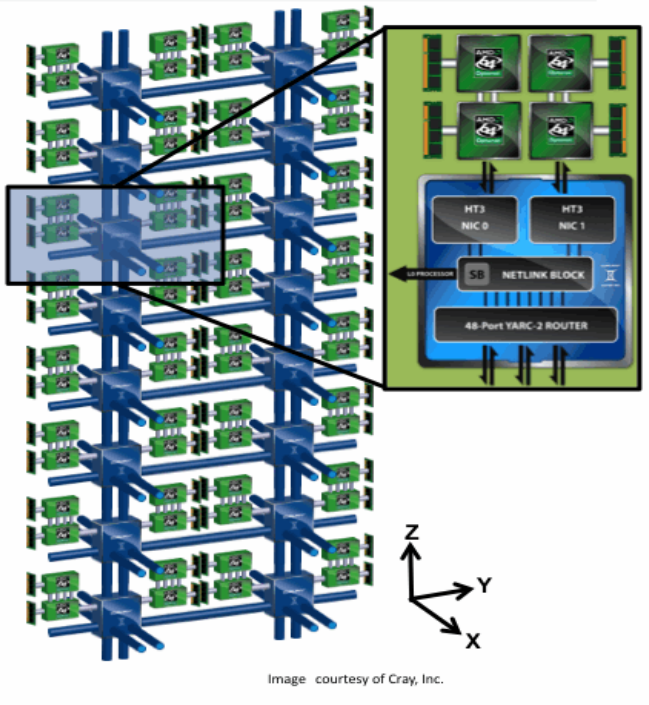
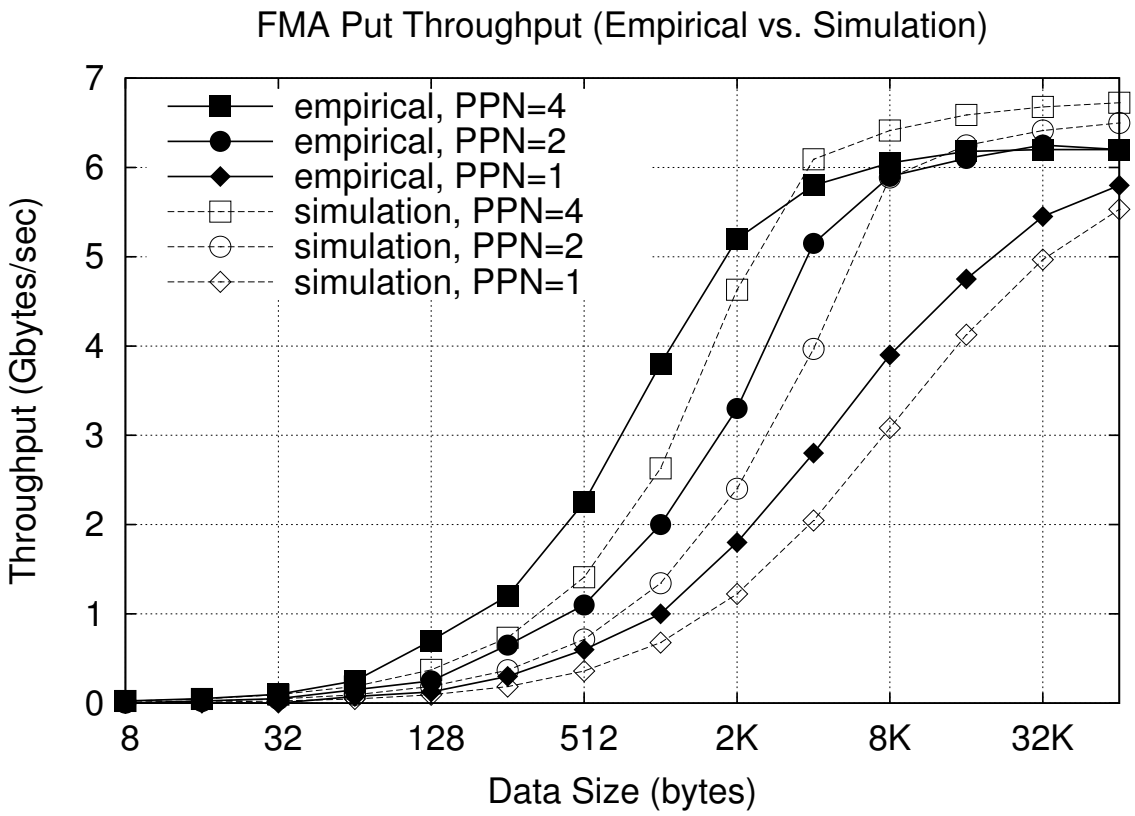



Image courtesy of Cray, Inc.

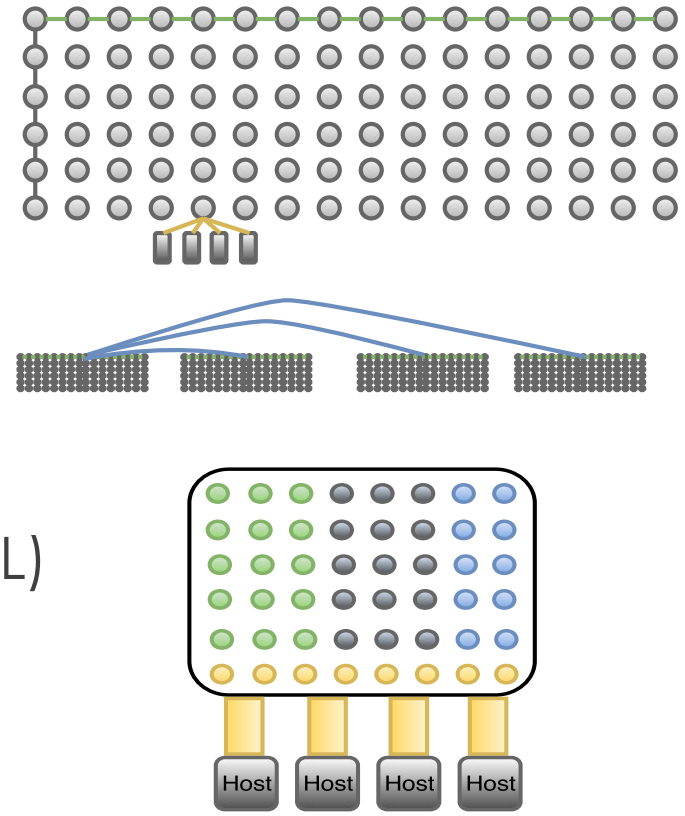
# Gemini Validation



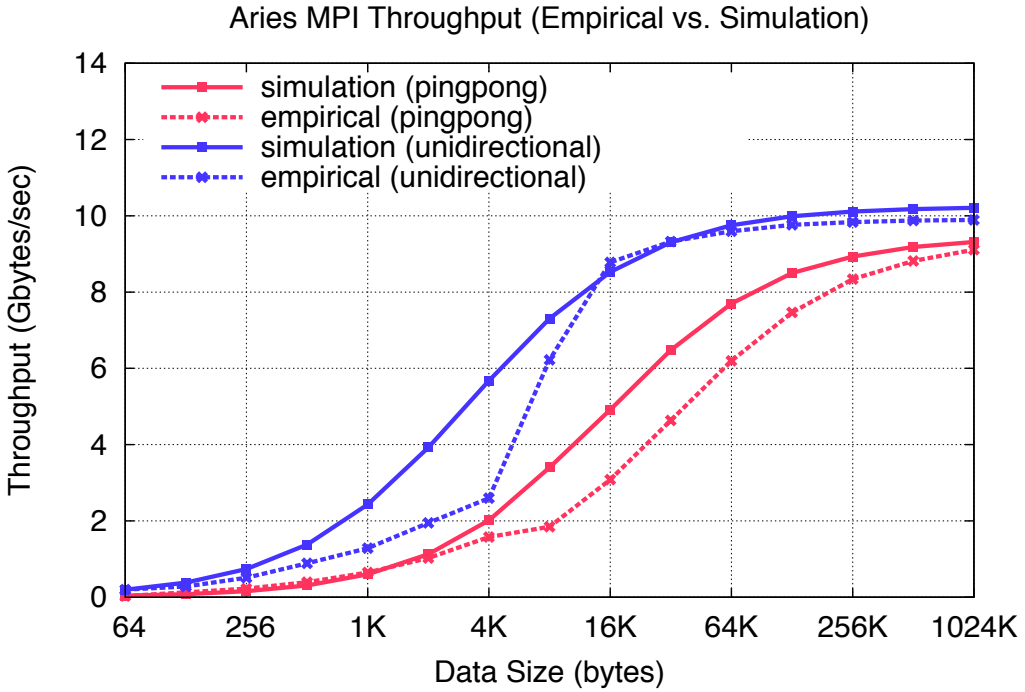

Gemini FMA put throughput (as reported in [2]) versus simulated throughput as a function of transfer size for 1, 2, and 4 processes per node.

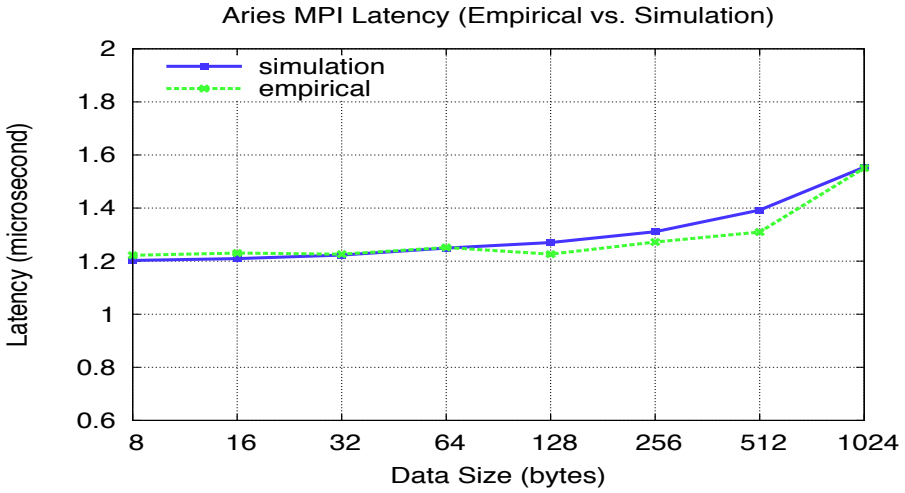
# Dragonfly - Cray's Aries Interconnect

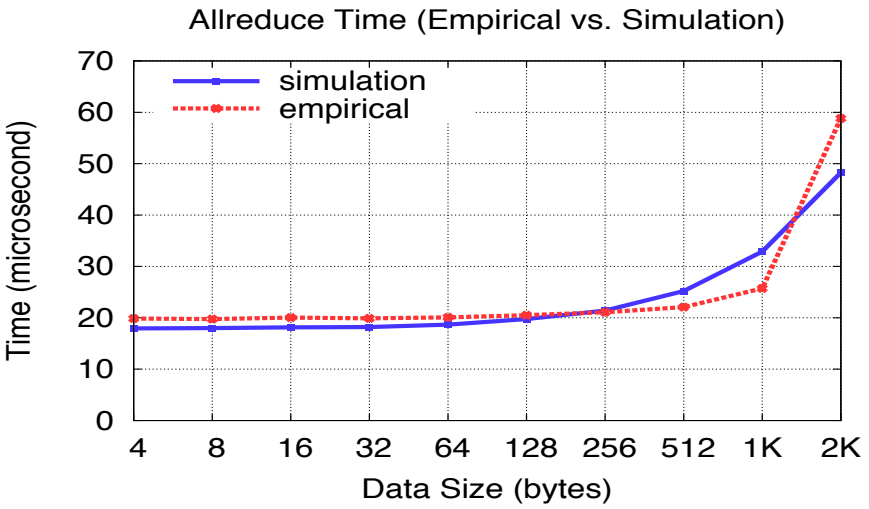
- 3-tier topology:
  1. Node-router connection
  2. Intra-group connection (local link arrangement)
  3. Inter-group connection (global link arrangement)
- Routing: MIN and VAL (UGAL)
- Examples: Trinity, …



# Aries Validation

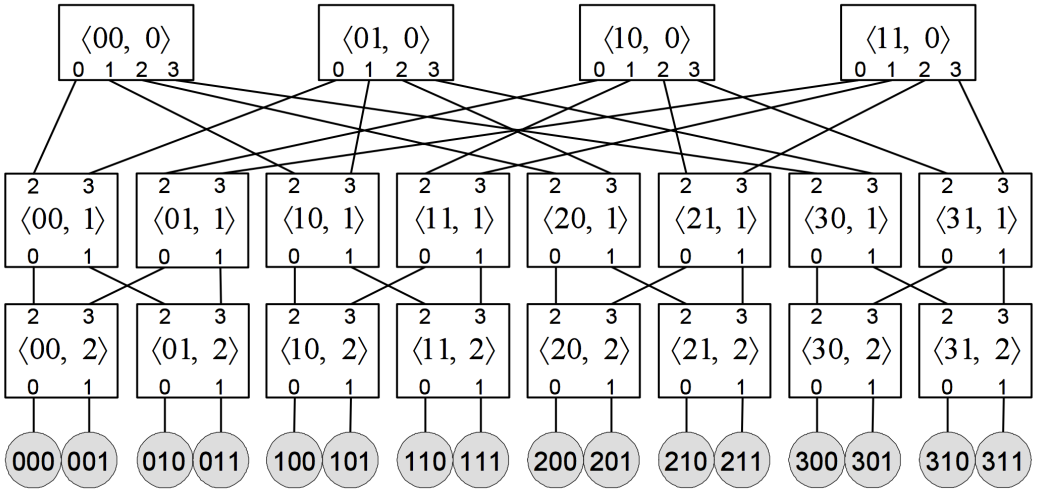


# Fat-tree FDR Infiniband

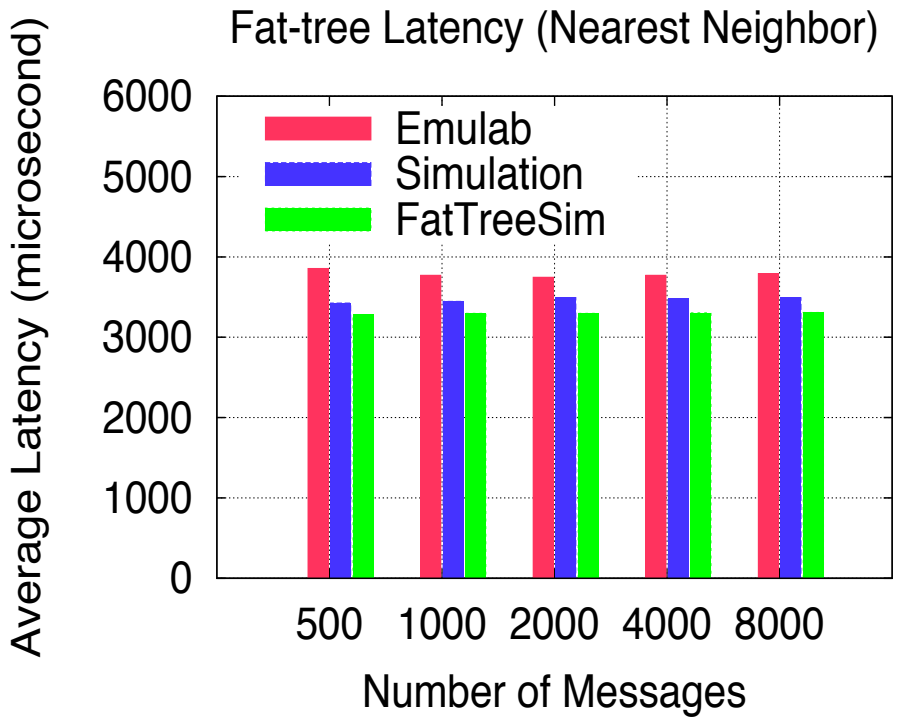
- An *m*-port *n*-tree:
  - Height is (n+1)
  - $2(m/2)^n$ processing nodes
  - $(2n-1)(m/2)^{n-1}$ m-port switches
- Routing has two separate phases:
  - Common root at LCA (lowest common ancestor)
  - Valiant, ECMP, MLID
- Examples: Stampede
  - 6400 nodes
  - 56 Gb/s Mellanox switches
  - 0.7 us uplink/downlink latency

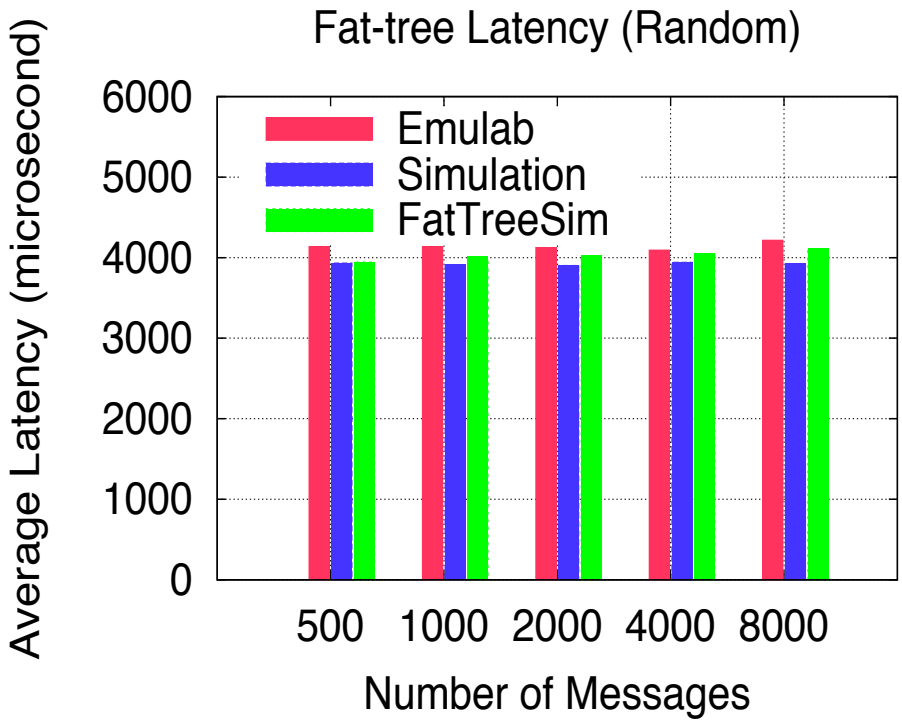



A 4-port 3-tree Fat-tree

Lin, Xuan-Yi, Yeh-Ching Chung, and Tai-Yi Huang. "A multiple LID routing scheme for fat-tree-based InfiniBand networks." *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004.

# FDR Infiniband Validation

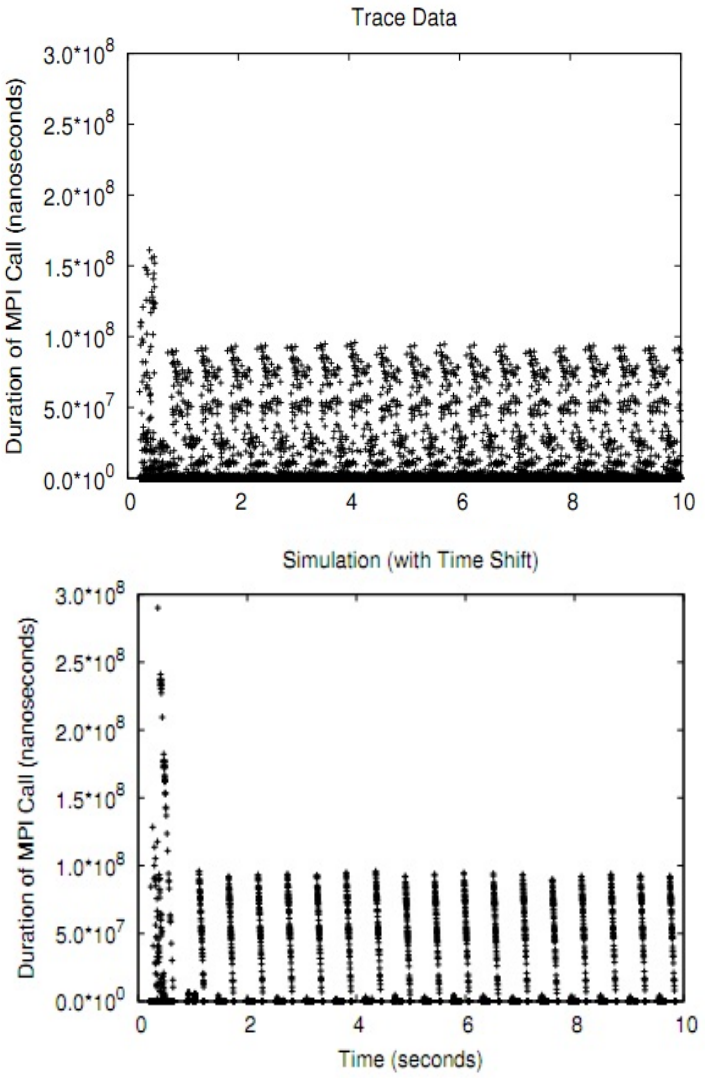


# Trace Driven Simulation



- Mini-app MPI traces:
  - Trace generated when running mini-apps on NERSC Hopper (Cray XE06 ) with <=1024 cores
  - Trace contains information of the MPI calls (including timing, source/destination ranks, data size, …)
- 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



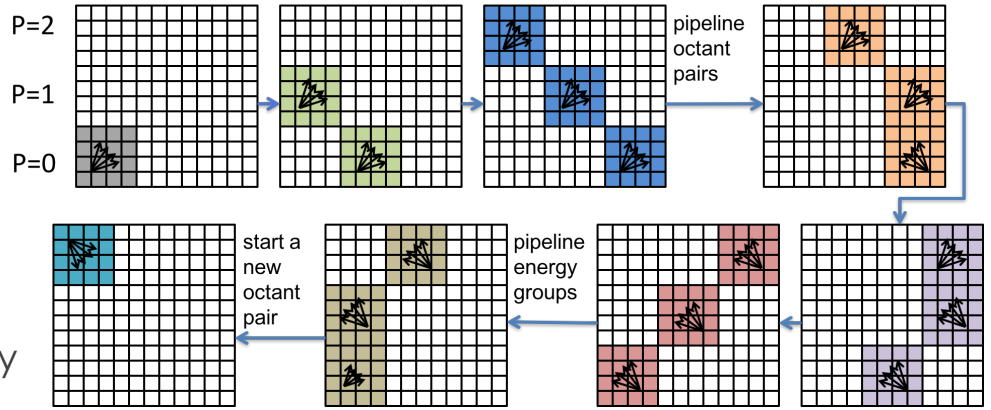
# Case Study: SN Application Proxy

- **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")
  - Scattering integrals approximated with expansion function of finite number of terms ("angular moments")
- **Outer/Inner solution strategy**
  - Outer iterations resolve group-to-group couplings
  - Inner iterations solve transport equation for each group, over all cells, along all angles, each time step -> *mesh sweeps*

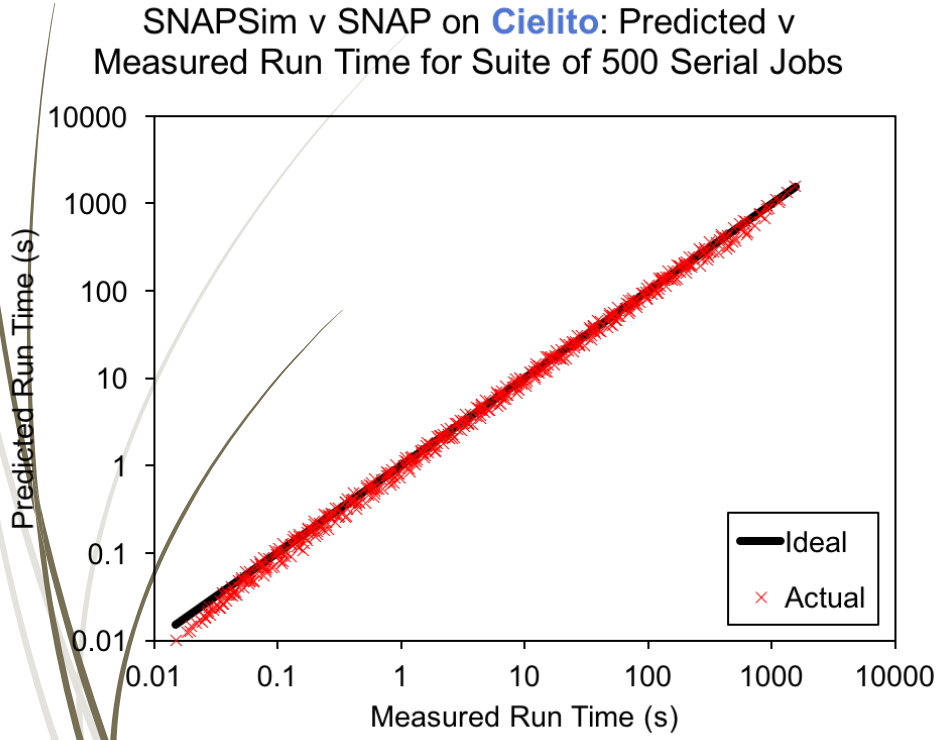
# Application Models

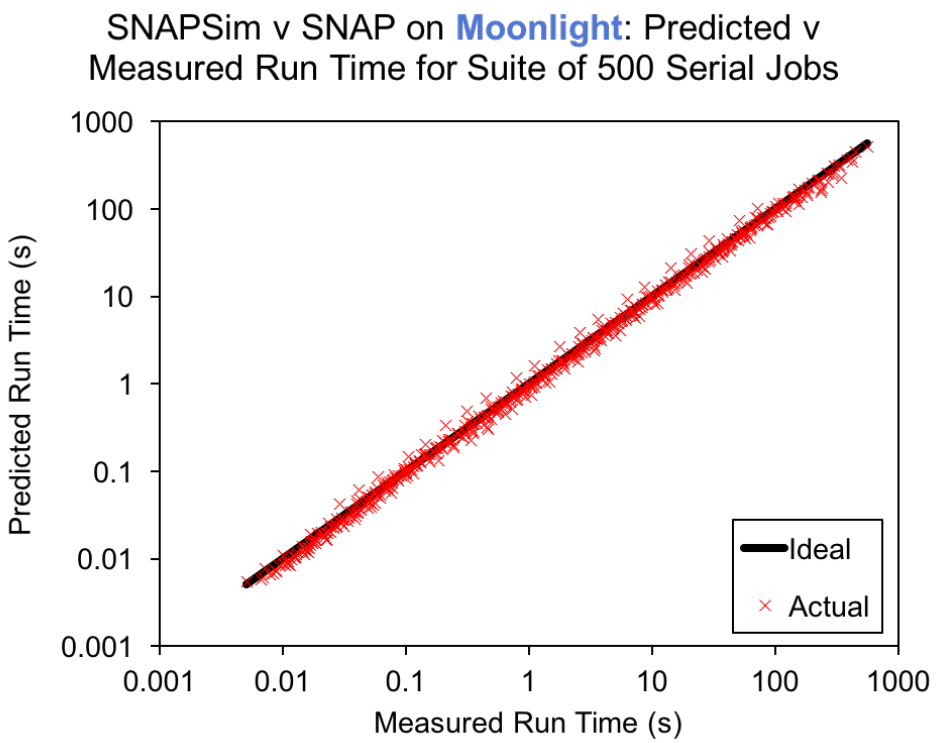
- Stylized version of actual applications
  - Focus on loop structures, important numerical kernels
- Use MPI to facilitate communication
- Use node model to compute time:
  - Hardware configuration based on clock-speed, cache-level access times, memory bandwidth, etc.
  - Input is a task-list that consists of a set of commands to be executed by the hardware, including, for example, the number of integer operations, the number of floating-point operations, the number of memory accesses, etc.
  - Predict the execution time for retrieving data from memory, performing ALU operations, and storing results

*A 2-D illustration of the parallel wavefront solution technique*



# Serial validation testing: 500 job test suite

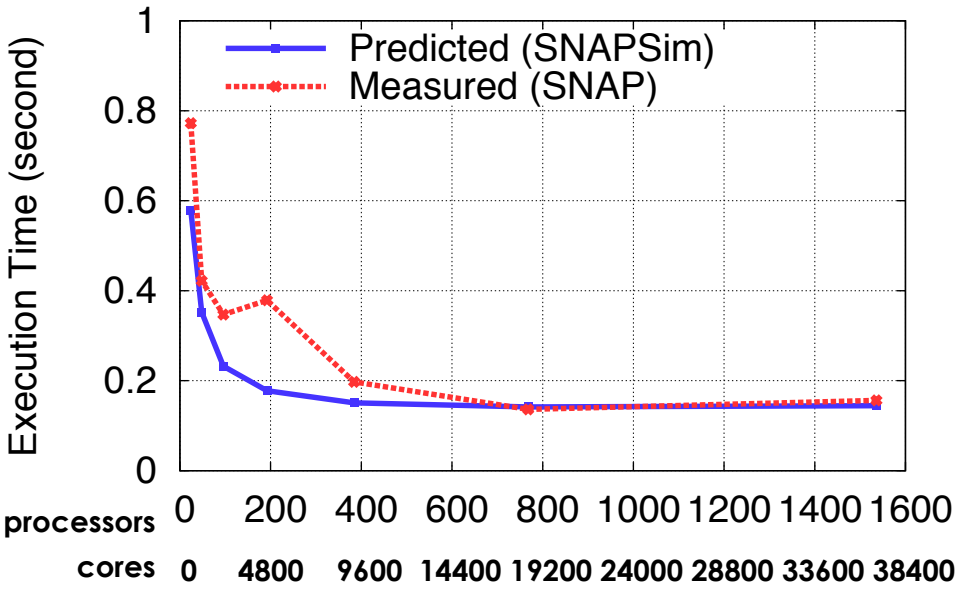


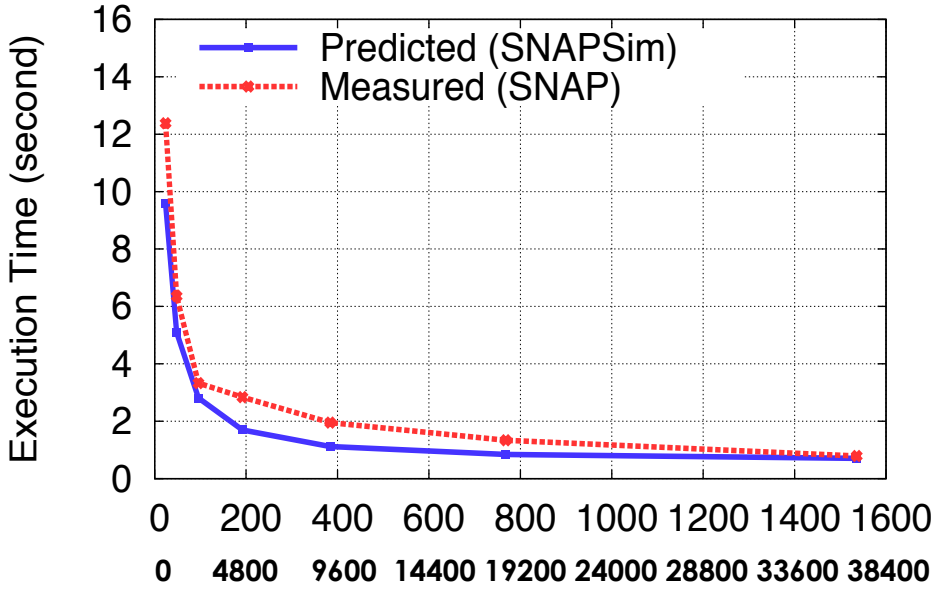

*A suite of 500 SNAP and SNAPSim jobs, varying the number of spatial cells, the number of angular directions per octant, the number of energy groups, and the number of angular moments for particle scattering approximation. Changing them has effects on memory hierarchy and parallelism.*

# Strong Scaling Experiments

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




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




NERSC's Edison supercomputer, which is Cray XC30 system with Aries interconnect (dragonfly). Each node has two sockets, each with 12 Intel Ivy Bridge cores and 32 GB of main memory.

# Conclusion

- Building a full HPC performance prediction model
  - Part of codesign process to test/improve code
  - Proper abstractions to simplify performance prediction problem
- PPT – Performance Prediction Toolkit
- Interconnection network models (torus, dragonfly, fat-tree)
- Validation (throughput, latency, trace-driven, applications)
- Application modeling: SNAP
- Future work:
  - Large-scale performance scaling studies
  - More applications

# Thank you


- Acknowledgments: