CODES-TRACER TUTORIAL

ENABLING HPC DESIGN SPACE EXPLORATION VIA DISCRETE-EVENT SIMULATION

NIKHIL JAIN
Compute Architect
NVIDIA

NEIL MCGLOHON
PhD Student Researcher
Department of Computer Science
Rensselaer Polytechnic Institute

Adapted from N. Jain and M. Mubarak's slides for HOTI2017

LINKS

- Slides available for download at:
  - http://nikhil-jain.github.io/codes-hoti.pptx
- QoS Slides available for download at:
  - https://nmcglo.com/public-talks/df-dfp-qos-isc19.pdf
- CODES Repo and Wiki:
  - https://github.com/codes-org/codes
- Tracer Repo:
  - https://github.com/LLNL/TraceR
- NERSC DOE Design Forward DUMPI Traces:
  - https://portal.nersc.gov/project/CAL/designforward.htm

TUTORIAL OUTLINE

- Session I: Introduction to CODES and TraceR, and their use for simulations
- Session II: Replaying HPC workloads, simulating MPI operations and collectives
- Session III: Storage simulations, Misc

PROJECT CONTRIBUTORS

- Argonne National Laboratory, Mathematics and Computer Science
- Lawrence Livermore National Laboratory, Center for Applied Scientific Computing
- Nvidia, Compute Arch
- Rensselaer Polytechnic Institute, Computer Science
- University of Arizona, Computer Science
- UC Davis, Computer Science
- University of Maryland, Computer Science

SESSION I: INTRODUCTION, CASE STUDY, AND INTERCONNECT MODELS

GOALS OF SESSION I

- Introduction
- Case study
- Features
- HPC Interconnect models
- Configuring interconnect simulations
- Interpreting simulation output

INTRODUCTION & BACKGROUND

BUILDING BLOCKS OF HPC INTERCONNECTS

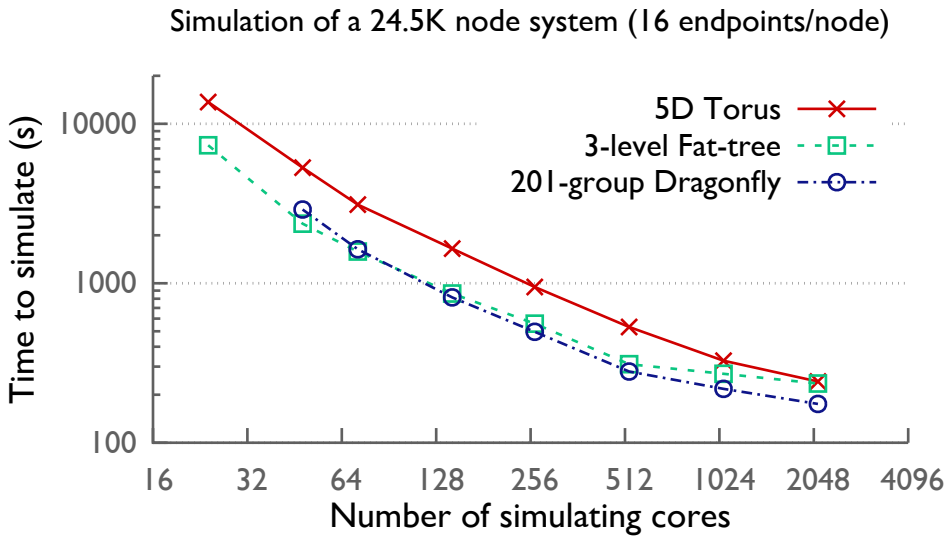
- **HPC performance is sensitive to the interconnect latency**
- **Interconnects are custom built to achieve maximum throughput**
- **Basic Components:**
  - **Network topology:** Arrangement of network nodes and links to maximize performance.
  - **Routing:** The protocol using which a packet selects its next destination. Depends on the underlying network topology.
  - **Flow control:** Allocating network links and buffers to packets as they traverse through the network. Usually independent of the underlying topology and routing protocols.

WHY HPC INTERCONNECT SIMULATION?

- HPC interconnects are complex!
- Need to look at several factors that influence network performance:
  - Interconnect Topology
  - Routing
  - Quality of Service
  - Job placement policies
  - Inter-job and intra-job interference
  - Storage placement on network
- Improved performance is contingent on an optimal combination of the above factors

ROSS: PARALLEL DISCRETE-EVENT SIMULATOR

- **Discrete event simulation (DES)**: a computer model for a system where changes in the state of the system occur at *discrete points* in simulation time
- *Parallel* DES allows execution of simulation on a parallel platform
- Rensselaer Optimistic Simulator System (ROSS) provides PDES capability for CODES
  - **Optimistically schedules events. Rollback realized via reverse computation**
  - Logical processes (LPs) model state of the system



CODES: MODELING & SIMULATION FRAMEWORK

- **Accelerate HPC system co-design by providing a detailed simulation of HPC interconnects, storage, workloads and surrounding environment**
- Couple best-of-breed parallel discrete event simulation with experts in interconnects and storage architecture design
- Incrementally develop HPC simulation capability, validating approach and components along the way
- Complement experimentation on real systems

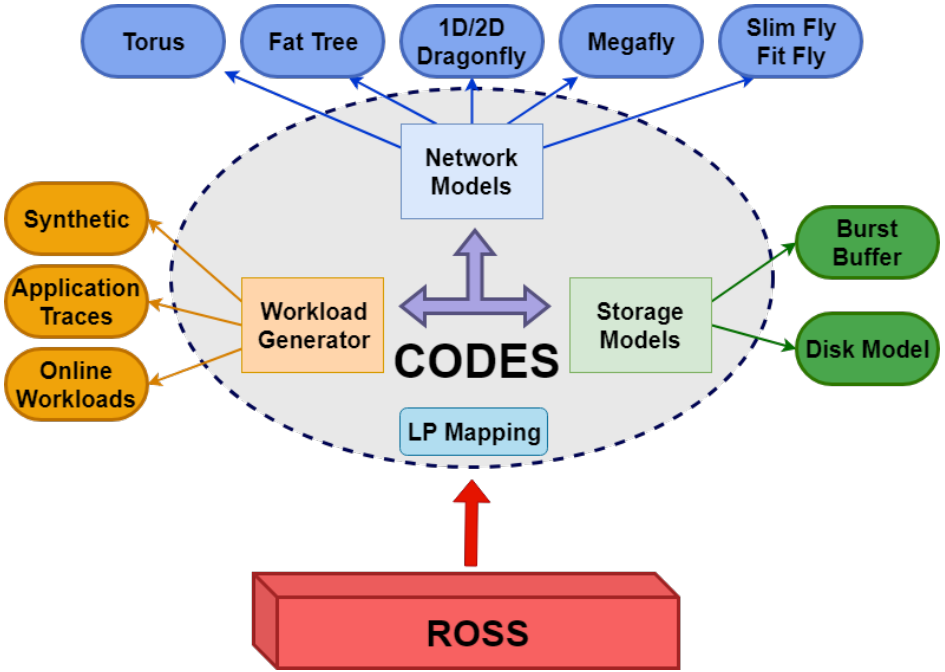


Figure: CODES Architecture Diagram

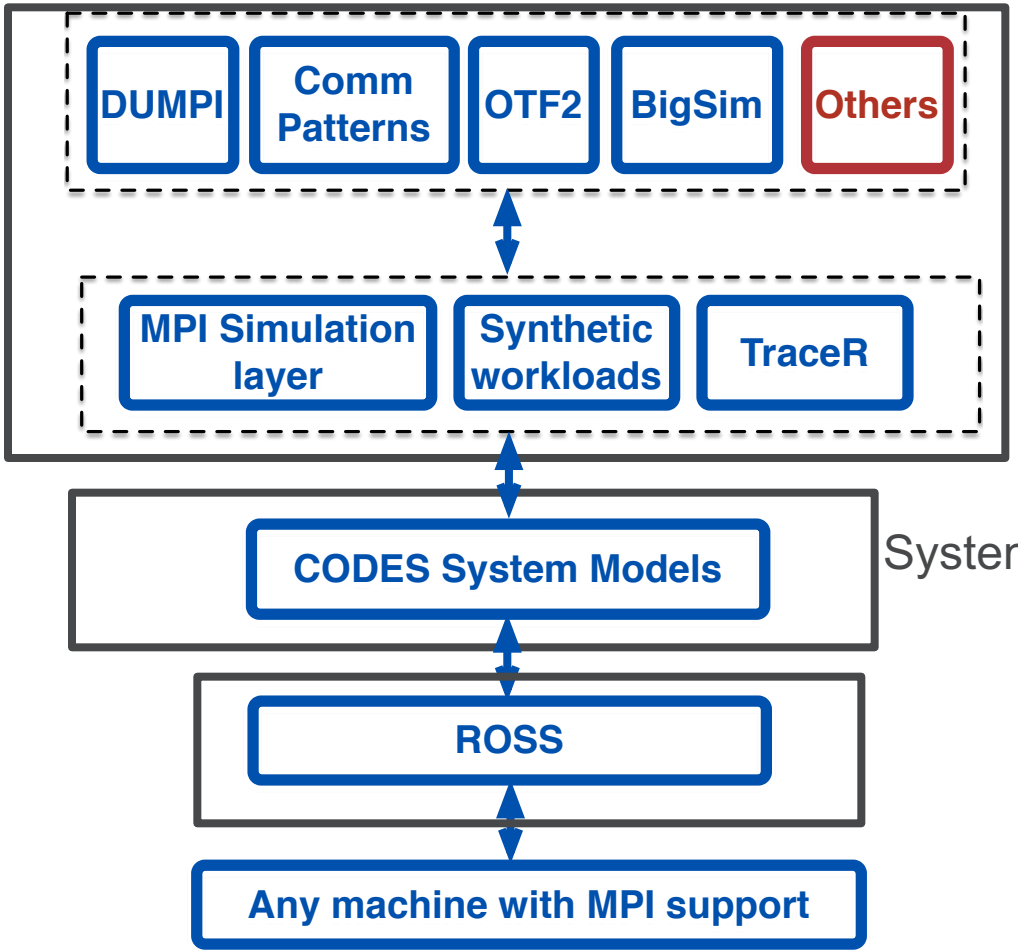
TRACER: REPLAYING MPI TRACES

- OTF2 traces for MPI
- BigSim traces for Charm++, AMPI
- Default and user-defined job placement and task mapping
- MPI point-to-point semantics and protocols
- Inbuilt collectives: tree based bcast, reduce, allreduce, and barrier; message size based algorithms for alltoall and allgather
- Simulation time scaling

FEATURES: HPC SIMULATIONS

- Packet-level simulations of HPC interconnect topologies
- Trace-driven analysis (DUMPI+OTF2), synthetic workloads
- Multiple jobs can be replayed on the network
- Different job placement schemes can be used
- Multiple ranks mapped to network nodes can be used
- Detailed statistics generation
- MPI collective operations can be simulated
- General purpose storage model that uses concurrent, pipelined RDMA read/write requests (simulating burst buffers/SSD)

SIMULATION STACK

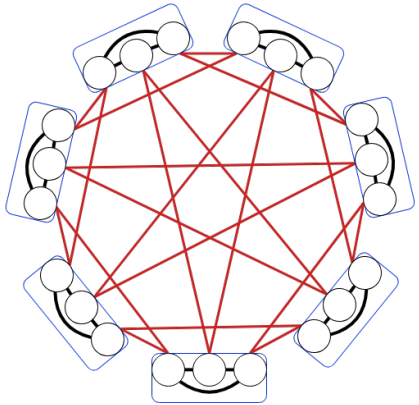


Workloads: Session II

System Models in Session I & III

HPC INTERCONNECT MODELS

STRUCTURE OF MODERN INTERCONNECT TOPOLOGIES



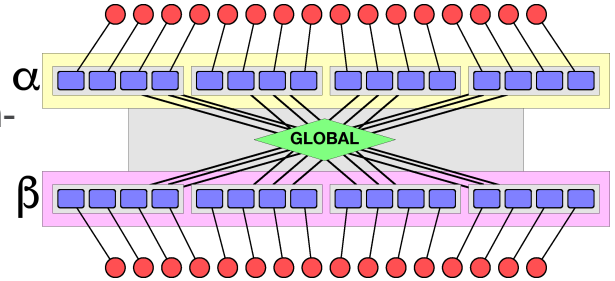
A 7 Group Dragonfly

Dragonfly Class Networks

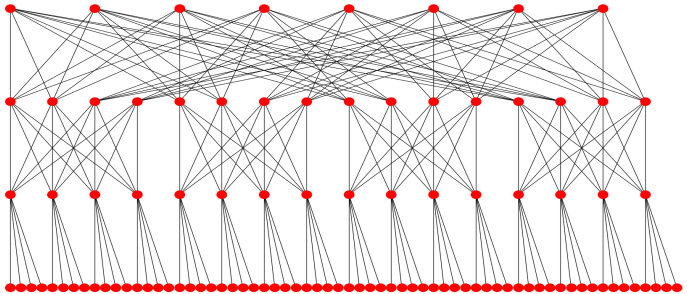
- Groups with All-To-All Global Connections
- Flexible Routing
- Strong resilience to inter-job interference

Slim Fly Network

- Cost-Effective
- Low-Diameter-High-Path-Diversity
- Complex Global Link Structure



An 8 Group Slim Fly



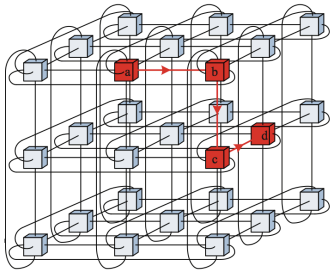
Fat tree Network

- Static Routing
- Job placement impacts inter-job interference

A 3-level pruned Fat Tree with 64 compute nodes

Torus Network

- Homogeneous structure
- Dedicated routers per compute node



A 3-ary 3-d Torus network

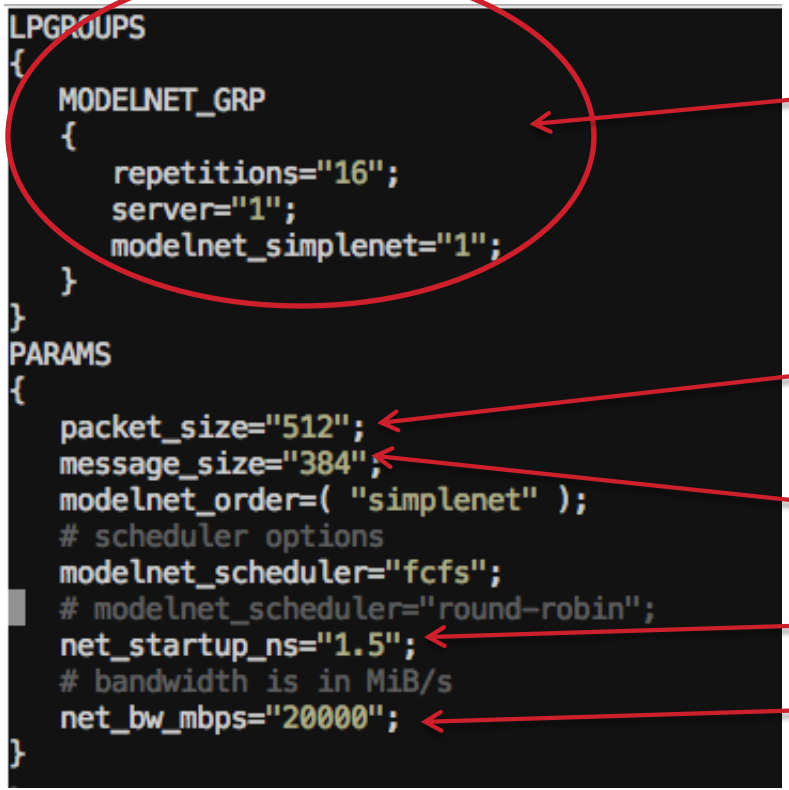
OVERVIEW OF NETWORK MODELS

- Multiple network models are supported – 1D/2D Dragonfly, Megafly, Slim Fly, Fat Tree, Express Mesh, Arbitrary graphs, Torus…
- Abstraction layer ‘model-net’ sits on top of network models
  - Breaks messages into packets
  - Offers FIFO, round robin and priority queues
- To try different networks, simply switch the network configuration files!
- Storage models, MPI simulation and workload replay layers are independent of the underlying networks

SIMPLE NET NETWORK MODEL

- A latency/bandwidth model where message is directly sent from source to destination
- Uses infinite size queuing
- Easy setup– uses a startup delay and link bandwidth for configuration
- Mostly for debugging/testing purposes-- Can be used as a starting point when replaying MPI traces
- It can be used as a baseline network model with no contention and no routing

CONFIGURING SIMPLE-NET LOGP MODEL



For mapping entities on ROSS MPI processes

Messages are broken into packets by the model-net layer

ROSS specific parameter (event size)

Startup delay in ns

Link bandwidth in MB/s (one link between each pair of nodes)

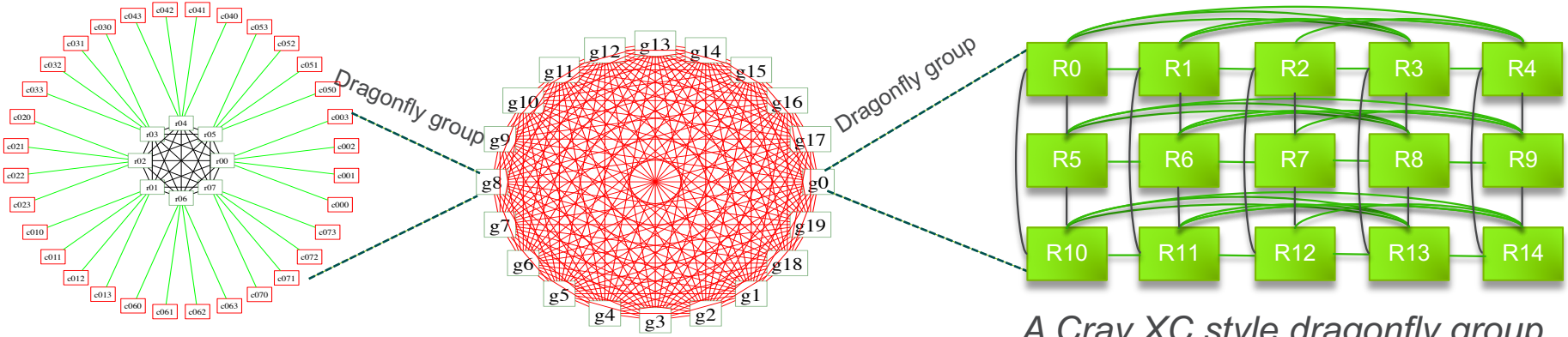
Configuration file can be found in codes/tests/conf/modelnet-test.conf

RUNNING A SIMPLE-NET LOGP MODEL

- `./tests/modelnet-test --sync=1 -- tests/conf/modelnet-test.conf`
- A simple test in which a simulated MPI rank sends message to the next rank, which replies back
- Continues until a certain number of messages is reached

DRAGONFLY NETWORK MODEL

- Multiple forms of routing are supported: minimal, adaptive, non-minimal and progressive adaptive
- Packet based simulation with credit based flow control
- Uses multiple virtual channels for deadlock prevention



Dragonfly group as proposed by Kim, Dally et al.

A Cray XC style dragonfly group

CONFIGURING DRAGONFLY NETWORK MODEL

```
LPGROUPS
{
   MODELNET_GRP
   {
      repetitions="2400";
# name of this lp changes according to the model
      nw-lp="4";
# these lp names will be the same for dragonfly-custom model
      modelnet_dragonfly_custom="4";
      modelnet_dragonfly_custom_router="1";
   }
}
```

nw-lp is a simulated MPI process

A simulated dragonfly network node

A simulated dragonfly network router

- For simulating multiple MPI processes per node → nw-lp=num-procs * number of network nodes
- Self messages→ messages sent to the same network node
- Overhead for sending self message can be configured

Configuration file can be found in codes/src/network-workloads/dragonfly-custom

CONFIGURING DRAGONFLY NETWORK MODEL

```
PARAMS
{
# packet size in the network
   packet_size="4096";
   modelnet_order=( "dragonfly_custom","dragonfly_custom_router" );
   # scheduler options
   modelnet_scheduler="fcfs";
# chunk size in the network (when chunk size = packet size, packets will not be
# divided into chunks)
   chunk_size="4096";
   # modelnet_scheduler="round-robin";
   # number of routers within each group
   # this is dictated by the dragonfly configuration files
   num_router_rows="6";
   # number of router columns
   num_router_cols="16";
   # number of groups in the network
   num_groups="25";
# buffer size in bytes for local virtual channels
   local_vc_size="8192";
#buffer size in bytes for global virtual channels
   global_vc_size="16384";
#buffer size in bytes for compute node virtual channels
   cn_vc_size="8192";
#bandwidth in GiB/s for local channels
   local_bandwidth="5.25";
# bandwidth in GiB/s for global channels
   global_bandwidth="4.69";
# bandwidth in GiB/s for compute node-router channels
   cn_bandwidth="16.0";
# ROSS message size
   message_size="592";
# number of compute nodes connected to router, dictated by dragonfly config
# file
   num_cns_per_router="4";
# number of global channels per router
   num_global_channels="4";
# network config file for intra-group connections
   intra-group-connections="../src/network-workloads/conf/dragonfly-custom/intra-9K-custom";
# network config file for inter-group connections
   inter-group-connections="../src/network-workloads/conf/dragonfly-custom/inter-9K-custom";
# routing protocol to be used
   routing="prog-adaptive";
}
```

Router arrangement within a group. Should match the input network configuration

Buffer size of virtual channels can be configured

Number of compute nodes per router is configurable

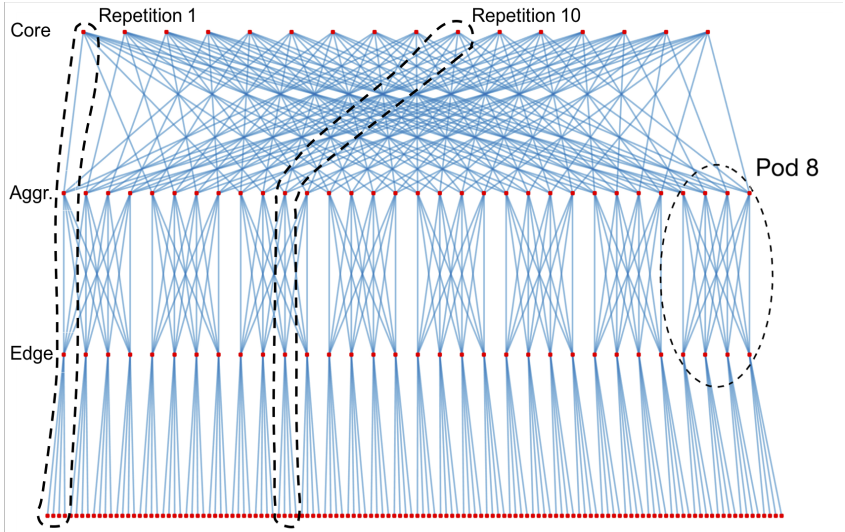
Network configuration files – can be custom generated (see scripts/gen-cray-topo/README.txt).

RUNNING A DRAGONFLY NETWORK SIMULATION

- Download the traces:
  - wget https://portal.nersc.gov/project/CAL/doe-miniapps-mpi-traces/AMG/df_AMG_n1728_dumpi.tar.gz
- Run the simulation:
  - ./src/network-workloads/model-net-mpi-replay --sync=1 --disable_compute=1 --workload_type="dumpi" --workload_file=df_AMG_n1728_dumpi/dumpi-2014.03.03.14.55.50- --num_net_traces=1728 --../src/network-workloads/conf/dragonfly-custom/modelnet-test-dragonfly-edison.conf

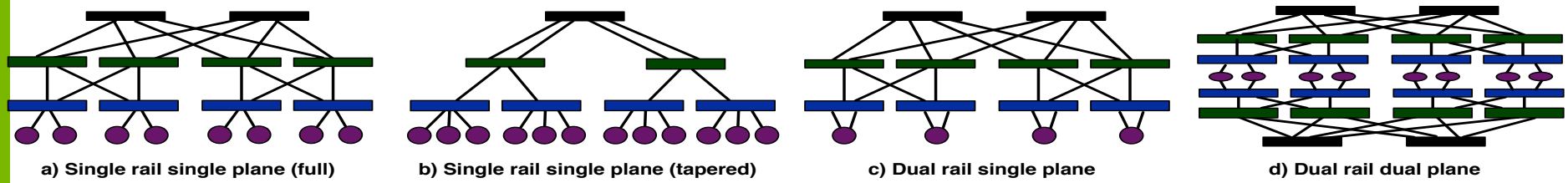
FAT TREE NETWORK MODEL

- Can simulate two and three level fat tree networks
- Width of the tree (number of pods) can also be configured
- Two forms of routing are supported:
  - static: uses destination-based look-up tables
  - adaptive: selects least congested output port
- Packet based simulation with credit based flow control



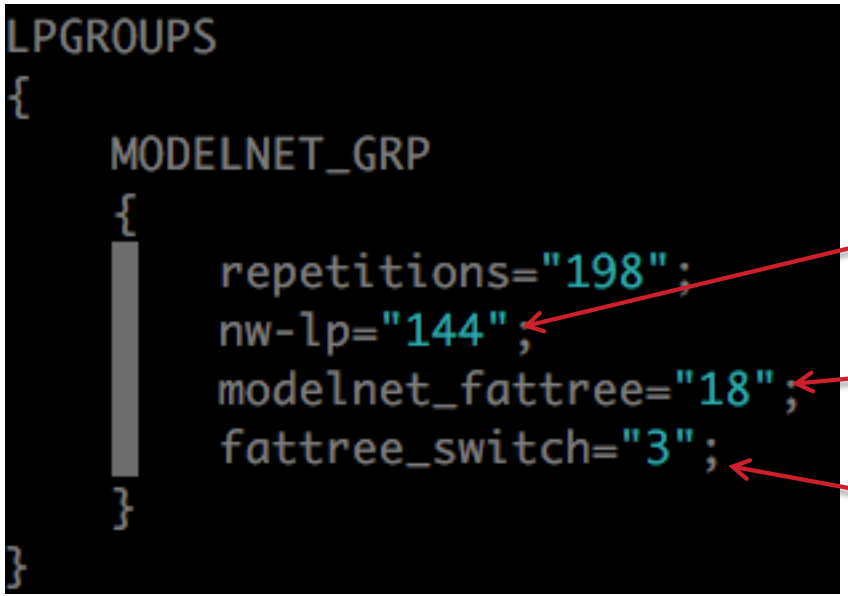
Full 3-level k=8 Fat-Tree with 128 compute nodes

FAT TREE NETWORK CONFIGURATIONS



- Tapering can be used to connect more nodes to the leaf switches
  - Reduces the bandwidth, switches and links at higher level
- To get higher bandwidth, nodes can connect to multiple ports (multi-rail) in one or more plane (multi-plane)
  - These configurations can also be tapered to reduce switches, links at higher levels
- Model supports configurations for multiple rails, multiple plane and tapering

CONFIGURING FAT TREE NETWORK MODEL



Nw-lp is a simulated MPI process

A simulated fat tree network node

Three simulated fat tree network switches (one in each level of the network)

CONFIGURING FAT TREE NETWORK MODEL

```
PARAMS
{
    packet_size="4096";
    message_size="624";
    chunk_size="4096";
    modelnet_scheduler="fcfs";
    modelnet_order=( "fattree" );
    ft_type="0";
    num_levels="3";
    switch_count="198";
    switch_radix="36";
    vc_size="65536";
    cn_vc_size="65536";
    link_bandwidth="12.5";
    cn_bandwidth="12.5";
    routing="static";
    routing_folder="/Fat-Tree/summit";
    dot_file="summit-3564";
    dump_topo="0";
}
```

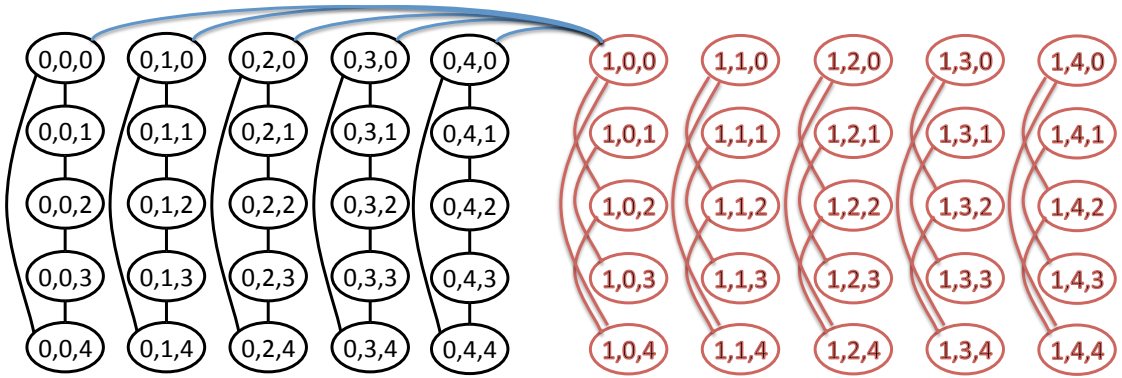
Switch arrangement should match the input network configuration

Switch radix can be configured

Static routing requires precomputed destination routing tables

See: https://github.com/codes-org/codes/wiki/codes-fattree#enabling-static-routing

SLIM FLY NETWORK MODEL



- Topology of interconnected router groups built with MMS graphs
- The max network diameter is always 2
- Packet based simulation with credit based flow control
- Multiple forms of routing are supported:
  - minimal: 2 virtual channels
  - non-minimal: 4 virtual channels
  - adaptive: 4 virtual channels

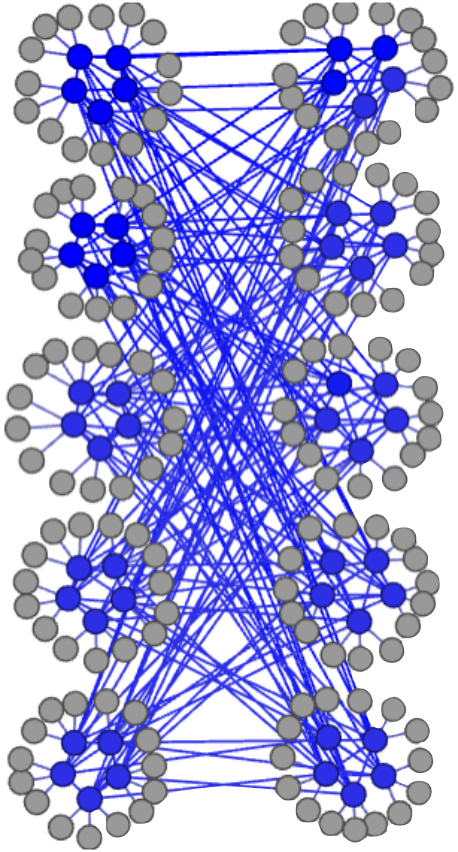


Fig. Slim Fly with q=5

CONFIGURING SLIM FLY NETWORK MODEL

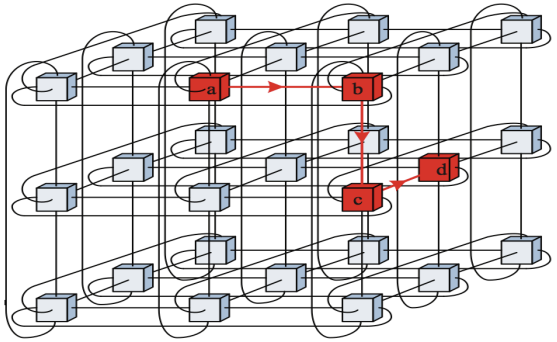
```
PARAMS
{
    packet_size="4096";
    chunk_size="4096";
    message_size="592";
    modelnet_order=( "slimfly" );
    modelnet_scheduler="fcfs";
    num_routers="13";
    num_terminals="9";
    global_channels="13";
    local_channels="6";
    generator_set_X=("1","10","9","12","3","4");
    generator_set_X_prime=("6","8","2","7","5","11");
    local_vc_size="25600";
    global_vc_size="25600";
    cn_vc_size="25600";
    local_bandwidth="12.5";
    global_bandwidth="12.5";
    cn_bandwidth="12.5";
    routing="minimal";
    num_vcs="4";
}
```

Router arrangement within a group.

Generator sets are set of indices used to calculate connections between routers in the same subgraph. They must be precomputed.

HYPER-X, EXPRESS MESH, AND TORUS

- Express Mesh: low-diameter densely connected grids
  - Allows for specifying connection gap
  - Gap = 1 -> HyperX
- Torus: based on a n-dimensional k-ary network topology
  - Number of torus dimensions and length of each dimension can be configured
  - Supports dimension order routing
- Uses bubble escape virtual channel for deadlock prevention



ARBITRARY GRAPHS

- Can also input arbitrarily connected graphs
  - Defined using DOT format
- Static routing is required
  - Generated using OpenSM, courtesy Jens Domke

INTERPRETING SIMULATION OUTPUT

INTERPRETING SIMULATION OUTPUT

```
        Total GVT Computations                                    0
        Total All Reduce Calls                                    0
        Average Reduction / GVT                                 nan

Total bytes sent 13584368 recvd 13584368
 max runtime 449332.124035 ns avg runtime 443706.882419
 max comm time 449332.124035 avg comm time 443706.882419
 max send time 5142770.436275 avg send time 2779472.247926
 max recv time 4149449.596308 avg recv time 2335071.940672
 max wait time 432820.362362 avg wait time 430457.043452
P-IO: writing output to dragonfly-simple-33488-1499374633/
P-IO: data files:
   dragonfly-simple-33488-1499374633/dragonfly-router-traffic
   dragonfly-simple-33488-1499374633/dragonfly-router-stats
   dragonfly-simple-33488-1499374633/dragonfly-msg-stats
   dragonfly-simple-33488-1499374633/model-net-category-all
   dragonfly-simple-33488-1499374633/model-net-category-test
   dragonfly-simple-33488-1499374633/mpi-replay-stats
Average number of hops traversed 1.709869 average chunk latency 0.925252 us maximum chunk latency 9.312357 us avg message size 812.563110 bytes finished messages 16820 finished chunks 65012

ADAPTIVE ROUTING STATS: 65012 chunks routed minimally 0 chunks routed non-minimally completed packets 65012

Total packets generated 39722 finished 39722
```

Application level statistics e.g. time spent in overall execution, communication, wait operations, amount of data transferred etc.

Enabling lp-io-dir generates detailed network statistics files

- *Average and maximum times are reported for all the application runs*
- *Network statistics (hops traversed, latency, routing etc.) are reported for the entire network*
- *Detailed statistics for each MPI rank, network node, router, port are generated using lp-io-dir option*
- *--lp-io-dir=my-dir can be used to enable statistics generation (Each lp writes it statistics to a summary file)*

STATISTICS REPORTED BY LP-IO

- Dragonfly-msg-stats:
  - number of hops, packet latency, packets sent/received, link saturation time reported for each network node
- Dragonfly-router-stats
  - link saturation time each router port
- Dragonfly-router-traffic
  - Traffic sent for each router port
- Fat tree and slim fly networks have similar statistics files.
- Mpi-replay-stats (generated for any network model):
  - bytes sent/received per MPI process
  - time spent in communication per MPI process
  - Number of sends and receives per MPI process

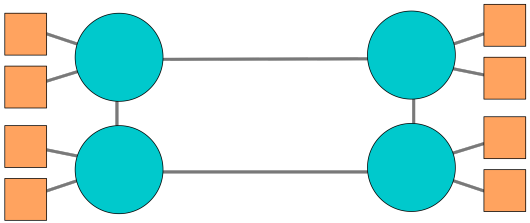
CASE STUDY

FIT FLY: INTERCONNECT INNOVATION THROUGH PARALLEL SIMULATION

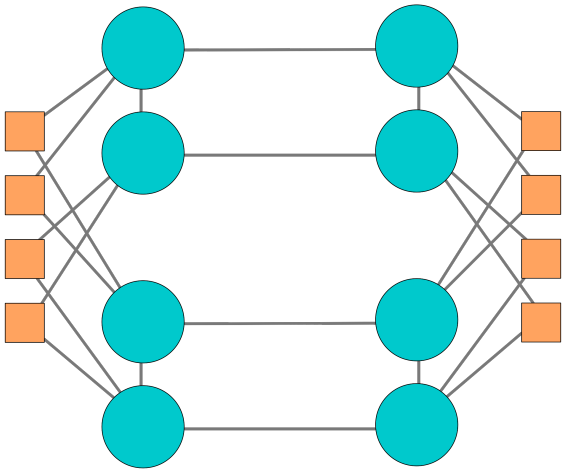
CODES is designed to help find answers to the “What If…” type questions in HPC Interconnection research

MULTI-PLANE NETWORKS

- Hand in hand with multi-rail networks
  - Additional links for packet injection
- Additional independent planes of routers
  - Often sharing terminals across planes



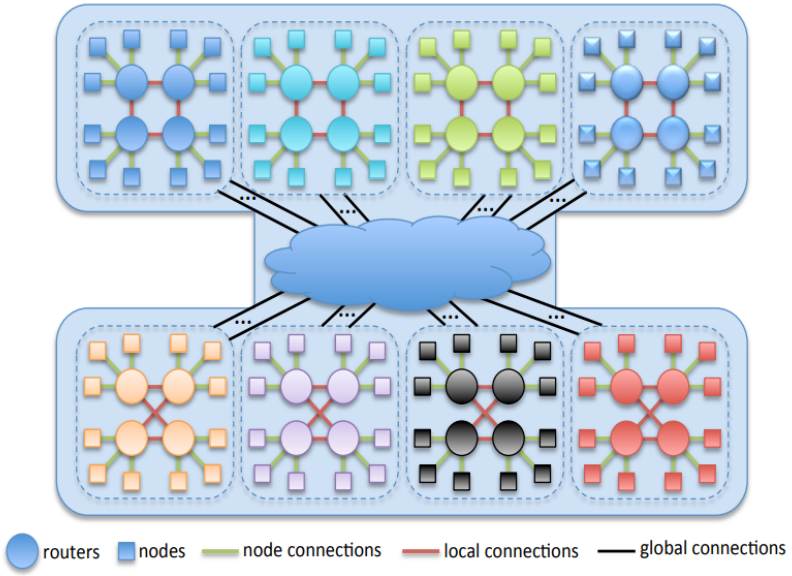
Single Plane



Dual Plane

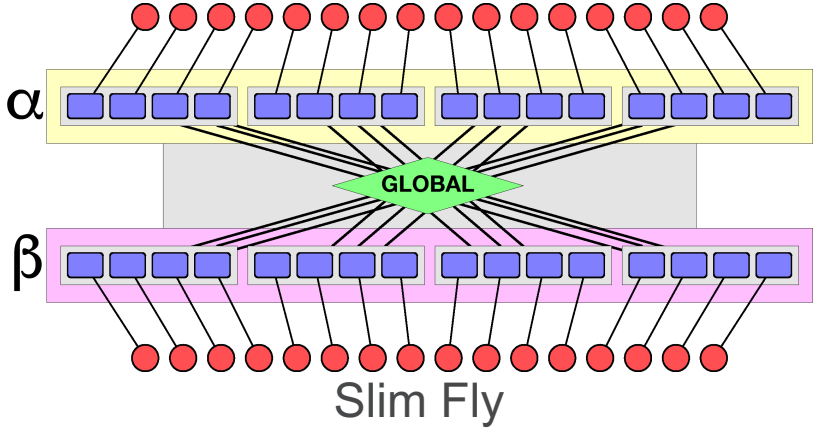
SLIM FLY NETWORK TOPOLOGY

- Routers in network are organized into groups
- Each Router
  - Some degree of Local connectivity
  - Some degree of Global connectivity
  - Some degree of Terminal connectivity
- Guaranteed Diameter-2
- Groups are divided into two subgraphs
  - No global connections between two groups within same subgraph
- Connections are determined via non-trivial generation method
  - Makes it challenging to physically build



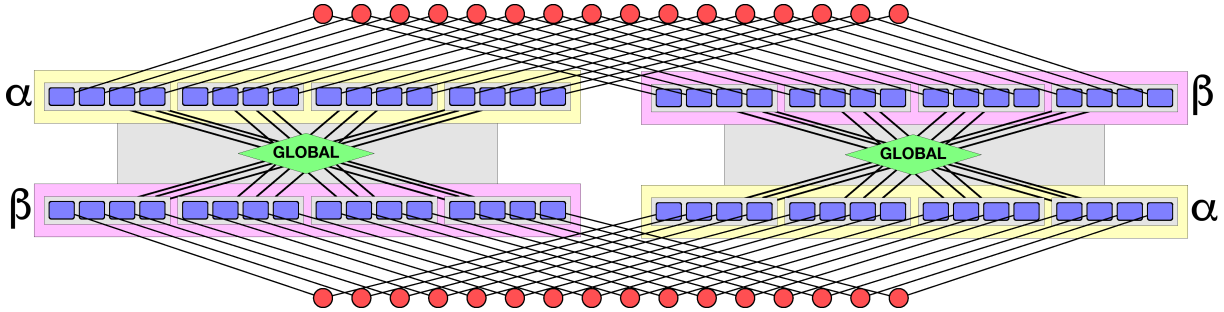
FIT FLY

- Multi-Planar Slim Fly Network
- Planes share single set of terminals
- Each plane follows same Slim Fly network generation method
- Terminal to Router mapping is alternating mirrored on each new plane
  - Increase path diversity



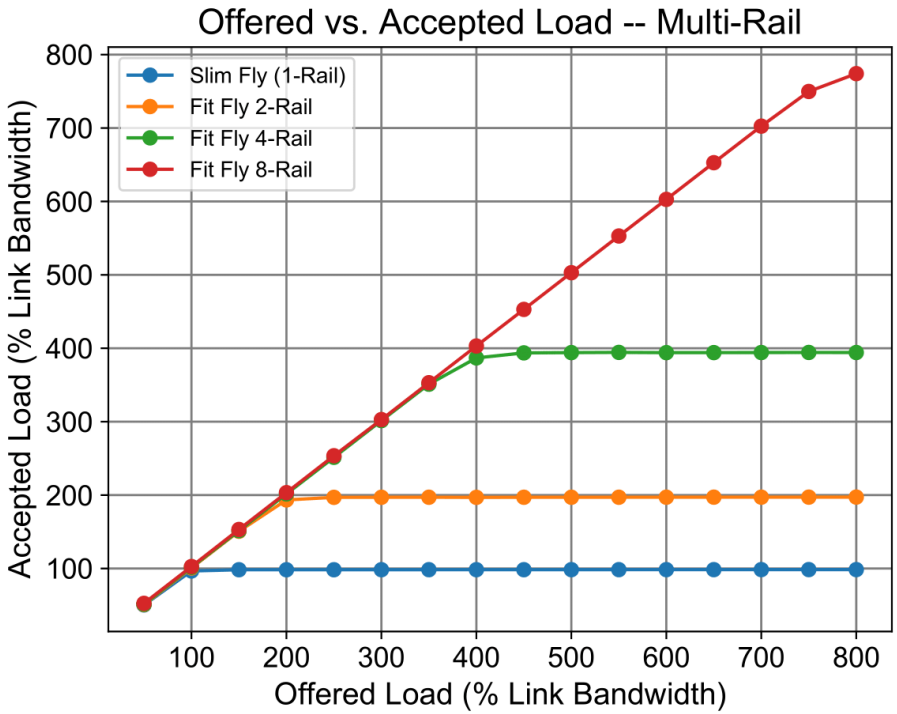
Slim Fly

Fit Fly



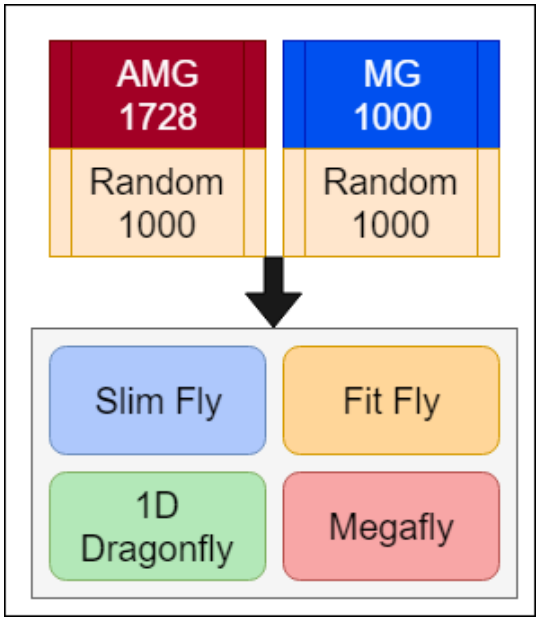
MORE RAILS = MORE THROUGHPUT

- Fit Fly Based on previously validated Slim Fly Model
- Additional planes bring additional throughput
- Observed expected increase in throughput with synthetic uniform random traffic

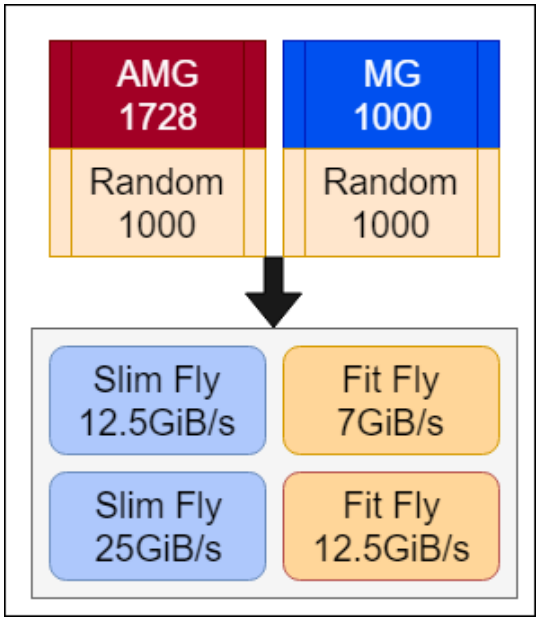


EXPERIMENTS OVERVIEW

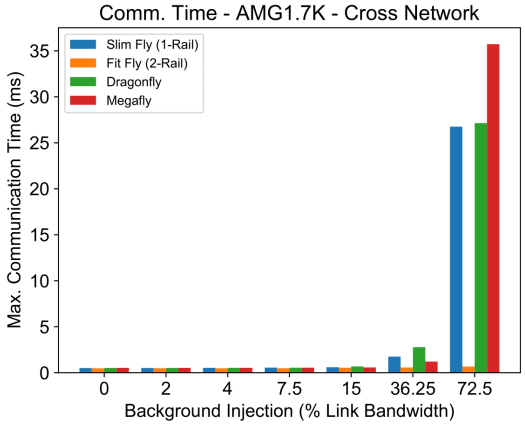
Experiment Set 1
Cross Network

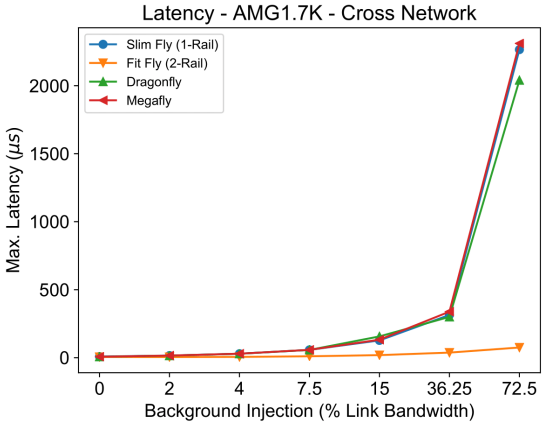


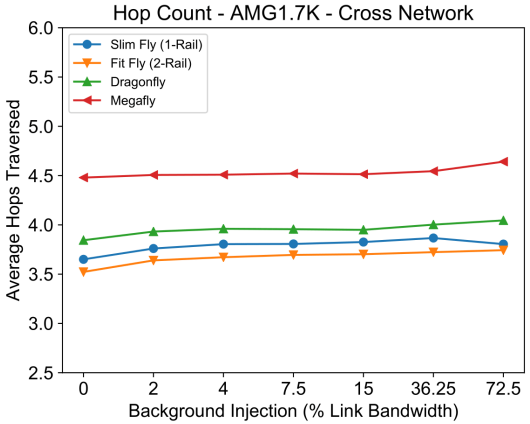
Experiment Set 2
Equalized Bandwidth



NETWORK COMPARISON: AMG 1728

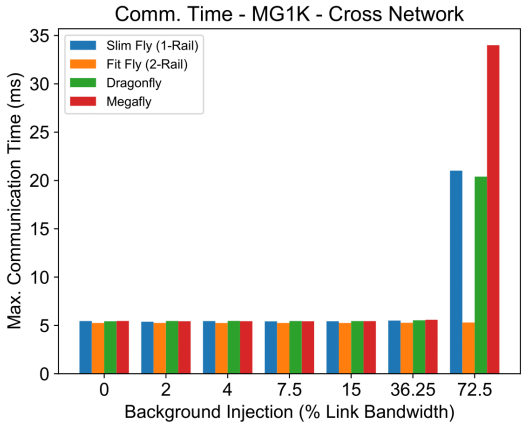


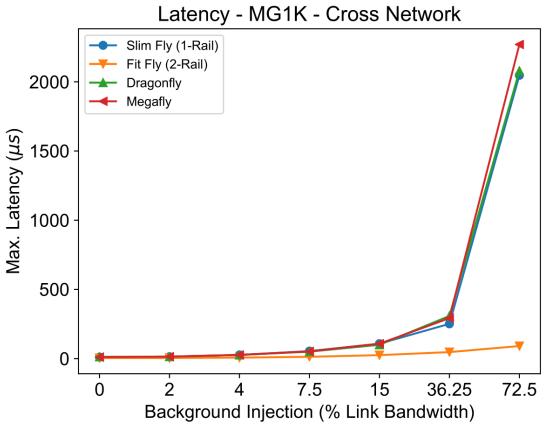


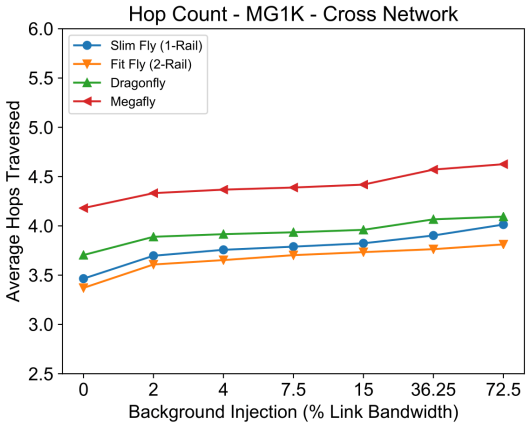


Lower is better

NETWORK COMPARISON: MG 1000







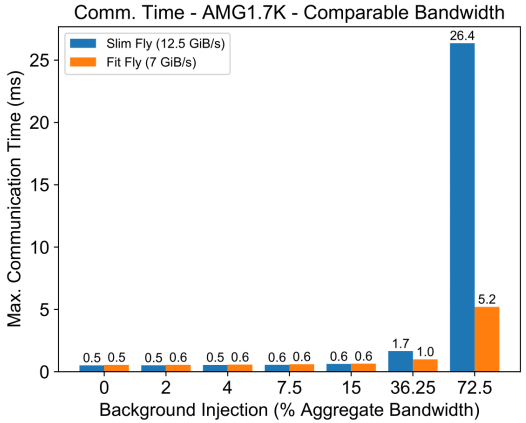
Lower is better

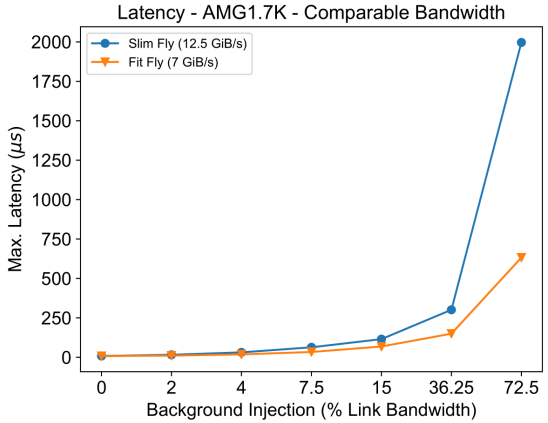
DISCUSSION: NETWORK COMPARISON

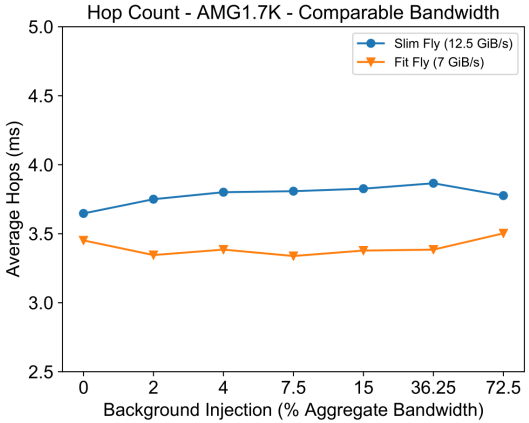
- Slim Fly performed well against state of the art Dragonfly and Megafly networks
  - Possible future exascale networks
- Fit Fly showed great resilience to high levels of interference traffic
  - Beat Slim Fly by an order of magnitude

- **Slim Fly and Fit Fly networks show great promise**
  - Low-diameter-high-path-diversity

EQUALIZED BANDWIDTH 12.5GIB/S – AMG 1728

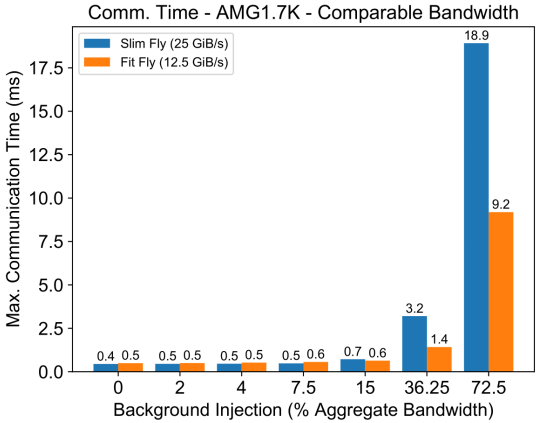


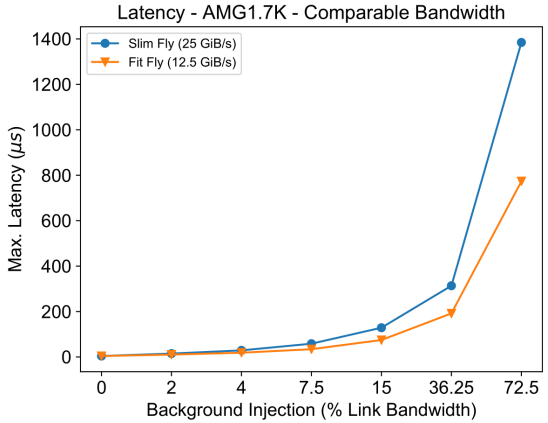


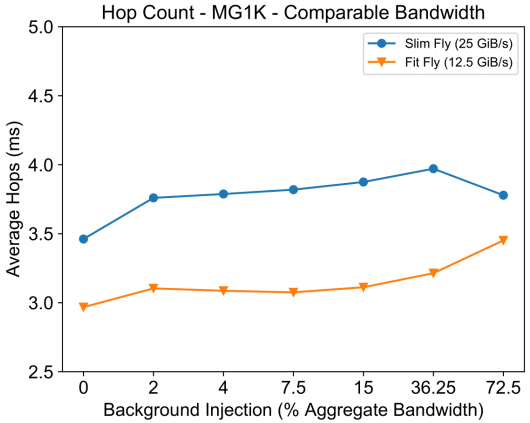


Lower is better

EQUALIZED BANDWIDTH 25GIB/S – AMG 1728







Lower is better

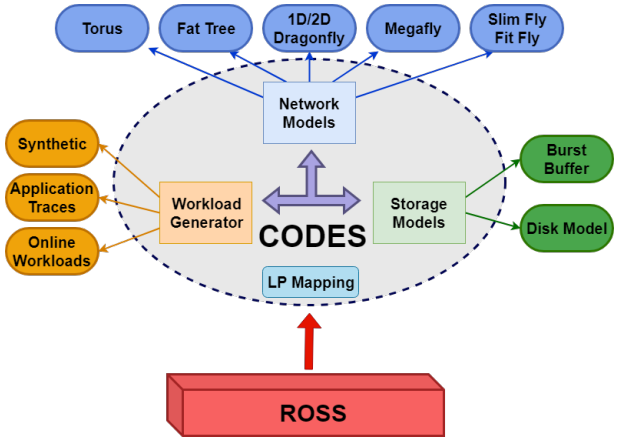
DISCUSSION: EQUALIZED BANDWIDTH

- Equalizing the aggregate bandwidth across networks slightly reduced the advantage that Fit Fly had
  - Fit Fly still pulled ahead
    - Greater interference resilience
- Additional planes of routers give less chance for any two packets to interact
  - Less interference
  - Less buffer wait time
  - Increased Application Performance

- **More planes of cheaper routers may be a better option to single-plane-high-bandwidth networks**

CASE STUDY: CONCLUSION

- Fit Fly well outperforms state-of-the-art interconnects
- More routers and planes = Less Interference
- More routers = More Cost
  - But cheaper routers and links could be used



- **CODES provides a strong environment for answering "What If…" questions and fostering future innovation in the field of HPC interconnection networks**

RUNNING INTERCONNECT SIMULATIONS

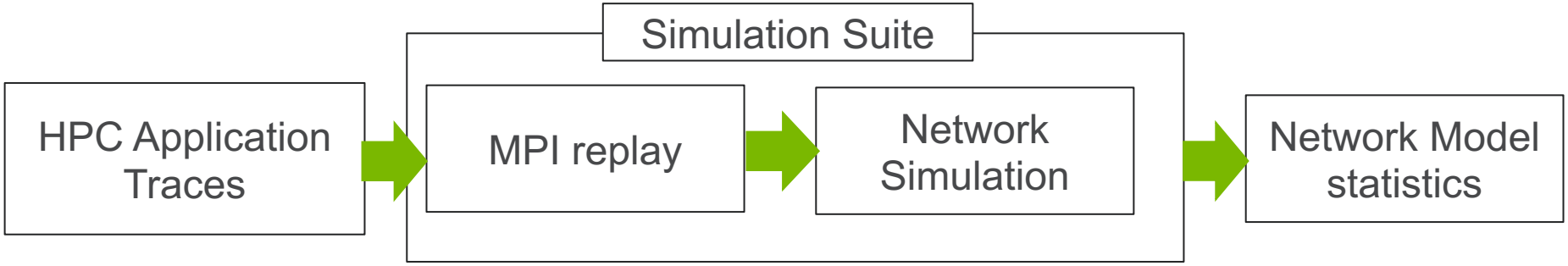
- Checkout the exercises at the wiki link:

https://github.com/codes-org/codes/wiki/quick-start-interconnects

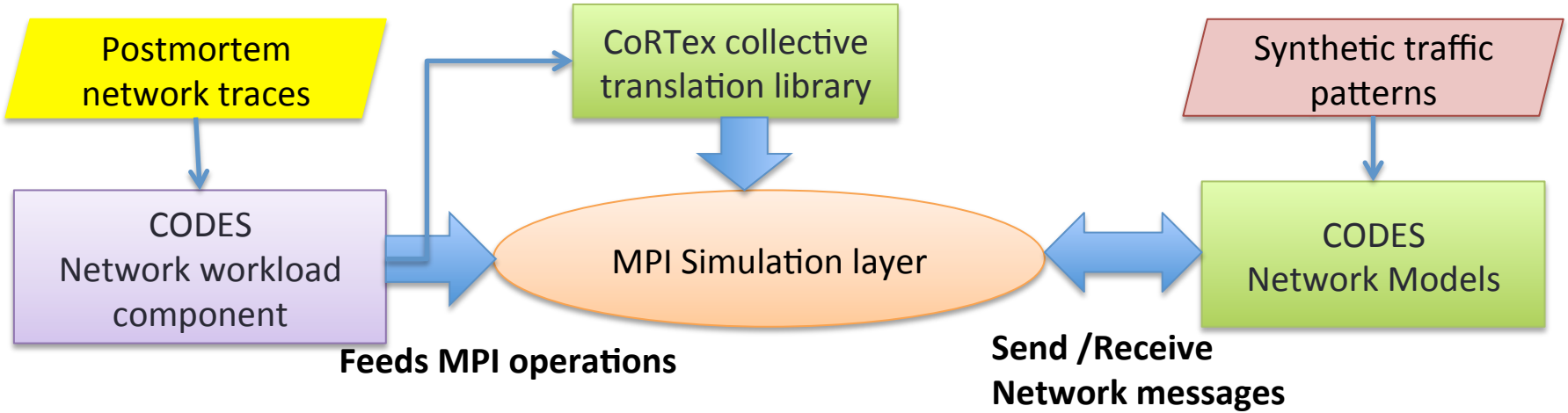
SESSION II: APPLICATION SIMULATION, WORKLOADS, MPI

FOUR STEPS TO SIMULATIONS

1. Prototype system design
   - Discussed in the previous session
   - Set up using network parameters
2. Workload selection
   - Depends on the use case
   - Application traces
   - Synthetic patterns
   - Skeletons
3. Workload creation
4. Execution



General framework for replaying traces on HPC interconnect simulation



CODES specific framework for replaying traces on HPC interconnect simulations

WORKLOADS

- ***Synthetic Workloads:***
  - Follow specific communication pattern and a constant injection rate
  - Often used to stress the network topology to identify best and worst case performance
  - Examples include uniform random, all to all, bisection pairing, bit permutation
  - Don’t require simulation of MPI operations
- ***HPC Application Traces:***
  - Useful for network performance prediction of production HPC applications
  - Trace size can be large for long running or communication intensive applications
  - Potential to capture computation-communication interplay
  - Require accurate simulation of MPI operations
  - Simulation results can be complex to analyze
- ***Intel SWM Online Workloads:***
  - Accurate workload representations
  - Decoupled from original application
  - Portable to arbitrary simulation environments
  - Generates traffic on-the-fly

DUMPI MPI TRACE LIBRARY

- Provides trace collection and replay tools for MPI based applications
- Trace collection is simple – link the MPI application with libdumpi
- Trace can be replayed using libundumpi utility
- Libundumpi provides callbacks you can use when MPI operations are replayed
- Preserves the causality order of MPI operations
- Captures detailed statistics for each MPI operation call

CAPTURING APPLICATION TRACES WITH DUMPI

- Repository can be cloned at:
  - *git clone https://github.com/sstsimulator/sst-dumpi.git*
- Configure and build using any MPI compiler
- Make sure to use '—enable-libdumpi' when configuring
- Once installed, simply add '-L$(DUMPI_INSTALL) -ldumpi' in your application
- DUMPI traces will be generated automatically with each application run
- Naming convention: dumpi-yyyy.dd.mm.hh.mm.ss-MPI-RANK-ID.bin
- More information can be found at: *https://github.com/sstsimulator/sst-dumpi*
- HPC application traces in DUMPI format: *https://portal.nersc.gov/project/CAL/designforward.htm*

GENERATING OTF2 TRACES (1/2)

- New Open Trace Format version 2 is supported by several tools
- ScoreP - Scalable Performance Measurement Infrastructure for Parallel Codes
- Tool suite with several libraries and helper tools
  - http://www.vi-hps.org/projects/score-p/
- Inside ScoreP source directory
  - CC=mpicc CFLAGS="-O2" CXX=mpicxx CXXFLAGS="-O2" FC=mpif90 F77=mpif77 ./configure --without-gui --prefix=<SCOREP_INSTALL>
  - make && make install
  - Make sure ScoreP installation's bin directory is in PATH
- Simple case: change the application linker to
  LD = scorep --user --nocompiler --noopenmp --nopomp --nocuda --noopenacc --noopencl --nomemory <your_linker>

GENERATING OTF2 TRACES (2/2)

- Before running, set the following environment variables:
  export SCOREP_ENABLE_TRACING=1
  export SCOREP_ENABLE_PROFILING=0
  export SCOREP_MPI_ENABLE_GROUPS=ENV,P2P,COLL,XNONBLOCK
- Turning tracing on/off: make sure these calls are synchronized
  - #include <scorep/SCOREP_User.h>
  - SCOREP_RECORDING_ON(); - start recoding
  - SCOREP_RECORDING_OFF(); - stop recording
- During compilation, add flags:
  -I$SCOREP_INSTALL/include -I$SCOREP_INSTALL/include/scorep -DSCOREP_USER_ENABLE
- Trace target options
  export SCOREP_TOTAL_MEMORY=256M
  export SCOREP_EXPERIMENT_DIRECTORY=/p/lscratchd/<username>/...

TRACING OUTPUT

- scorep-* directory generated with following content:
  scorep.cfg **traces** traces.def traces.otf2
- scorep.cfg is human readable; can be used to verify if the environment is correctly generated
- traces.otf2 is a binary meta-file
- traces is a directory that contains the details
- Use otf2-print utility in ScoreP bin to view the traces:
  otf2-print –L 0 traces.otf2

INFORMATION CAPTURED IN A TYPICAL TRACE (E.G. IN DUMPI, OTF2)

Time stamp, t (rounded off)	Operation type	Operation data (only critical information is highlighted)
t = 10	MPI_Bcast	root, size of bcast, communicator
t = 10.5	MPI_Irecv	source, tag, communicator, req ID
t = 10.51	user_computation	optional region name - “boundary updates”
t = 12.51	MPI_Isend	dest, tag, communicator, req ID
t = 12.53	user_computation	optional region name - “core updates”
t = 22.53	MPI_Waitall	req IDs
t = 25	MPI_Barrier	communicator

EXAMPLE TO SHOW THE EFFECT OF REPLAYING TRACES

Original Time stamps	Original duration	New Time stamps	New duration	Operation type
10	0.5	10	0.2	MPI_Bcast
10.5	0.01	10.2	0.01	MPI_Irecv
10.51	2	10.21	2	user_computation
12.51	0.02	12.21	0.02	MPI_Isend
12.53	10	12.23	10	user_computation
22.53	2.47	22.23	0.03	MPI_Waitall
25	1	22.26	1.7	MPI_Barrier

DUMPI VS OTF2

- Most of the information in the trace format is the same
- Different w.r.t. capturing of dynamically determined events: e.g. MPI_Waitany
- DUMPI: stores all the information passed to the MPI call
  - Simulation decides which request to fulfill: accurate resolution for target systems
  - If the control flow of the program can change significantly due to the ordering of operations, simulations are not entirely correct
- OTF2: stores only the information that is used (e.g. which request was satisfied)
  - Accurately mimics the control flow of the trace run
  - But does not accurately represent execution for the target system
- Artifact of leveraging existing tools not originally intended for PDES!

INTEL SWM WORKLOADS

- Open Source version hosted at https://github.com/codes-org/SWM-workloads
- Built separately: a CODES-SWM interface has been developed
- Includes several workloads including LAMMPS, Nekbone, Nearest Neighbor, HACC, MILC, Incast, Point-to-Point
- Each workload is configured by its own JSON configuration file
  - Specifies size
- More CODES use information:
  - https://github.com/codes-org/codes/wiki/online-workloads

```
{
    "jobs": {
            "name": "StandaloneSWM",
            "app": "dll",
            "dll_path": "apps/dll/lammps.so",
            "size": 2048,
            "time": 0,
            "cfg":
            {
                "num_x_replicas": 3,
                "num_y_replicas": 3,
                "num_z_replicas": 3,
                "num_time_steps": 30,
                "req_vc" : 0,
                "resp_vc" : 1,
                "router_freq" : 800e6,
                "cpu_freq" : 4e9,
                "cpu_sim_speedup" : 1e6
            }
        }
}
```

SIMULATING MPI

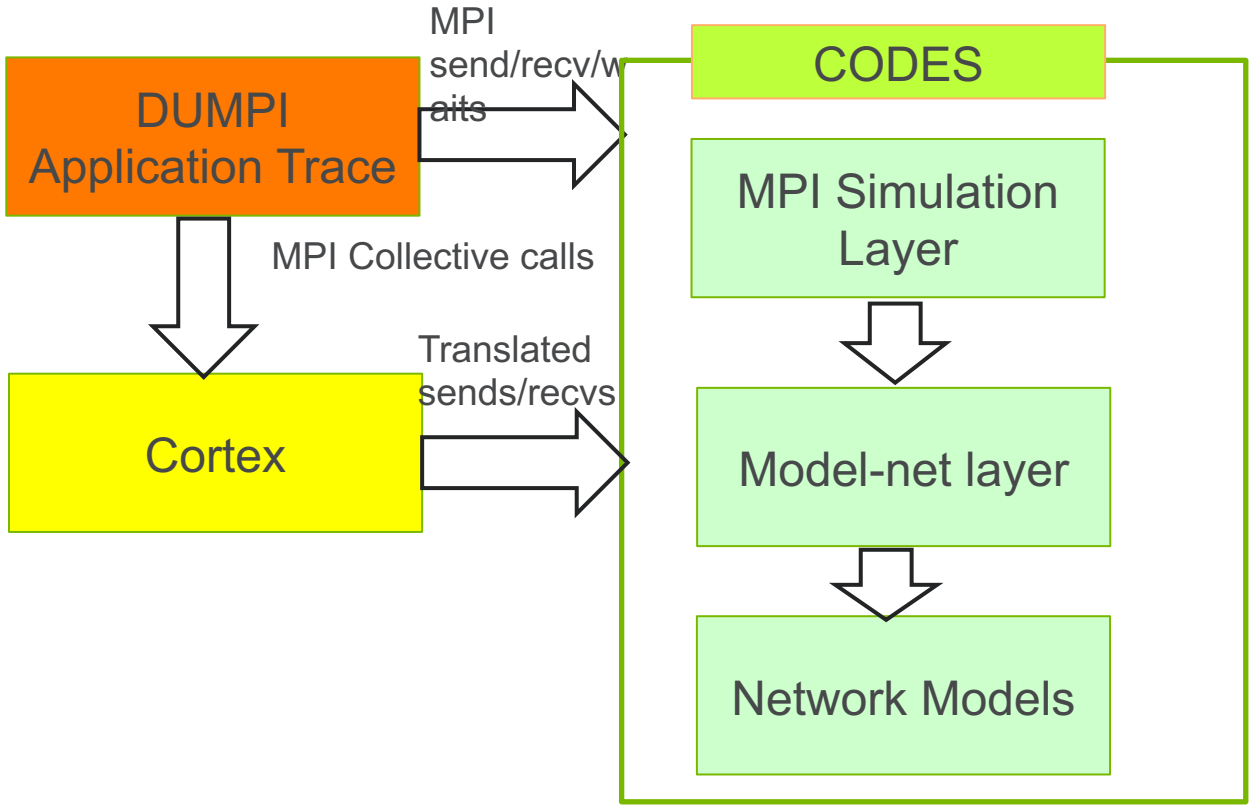
MPI SIMULATION

- Matching semantics and standard has to be followed for a correct simulation
  - So obviously done
- Eager – Rendezvous protocol
  - Cutoff can be specified in the config
- Library call overheads handled using a constant cost
- Collectives:
  - OTF2 based simulations implements them internally
  - DUMPI based simulations use Cortex

TRANSLATING MPI CALLS USING CORTEX

- Internally most MPI implementations support collectives by translating into point to point
- Cortex comes with a set of translation functions to convert collectives into point to point using MPICH algorithms
- When linked with DUMPI and CODES, Cortex translates MPI collectives into point to point sends/receives (simulated by CODES)
- Cortex can also be used to implement your own translation functions (e.g. collective algorithms)
- Cortex tutorial is available at : https://xgitlab.cels.anl.gov/mdorier/dumpi-cortex/wikis/home

CODES, CORTEX AND DUMPI INTERACTION



MPI TRANSLATION WITH CORTEX

- To enable collective translation, install Cortex and reconfigure CODES with Cortex
- Cortex available for download: git clone https://xgitlab.cels.anl.gov/mdorier/dumpi-cortex.git
- `cmake .. -G "Unix Makefiles" -DMPICH_FORWARD:BOOL=TRUE -DCMAKE_INSTALL_PREFIX=$HOME/CODES/install/cortex -DDUMPI_ROOT=$HOME/CODES/install/dumpi`
- See instructions at: https://xgitlab.cels.anl.gov/codes/codes/wikis/codes-cortex-install
- Use –with-cortex=/path/to/cortex/install option

IN A NUTSHELL: REPLAYING A SINGLE APPLICATION TRACE

```
./bin/model-net-mpi-replay --sync=1 --disable_compute=1 --
workload_type="dumpi" --workload_file=dumpi-
2014.03.03.14.55.50- --num_net_traces=1728 -- modelnet-test-
dragonfly-edison.conf
```

- Runtime options
  - --workload_type: “dumpi” or “online” for SWM
  - *--num_net_traces : Number of input network traces*
  - *--workload_file: DUMPI trace file*
  - *Network configuration file: Any of the network files (number of simulated ranks > number of ranks in trace)*
  - *--lp-io-dir (optional): Generates detailed network counters and statistics*
  - *--lp-io-use-suffix (optional): Generates a unique directory per run*
  - *--disable_compute (optional): disable any compute time between MPI events*
  - *--workload_conf_file: for SWM, specifies name of workload to be used*
- For running parallel simulations, use mpirun and –sync=3

SIMULATING MULTIPLE JOBS ON THE NETWORK

REPLAYING MULTIPLE JOBS

```
./src/network-workloads/model-net-mpi-replay --sync=1 --
disable_compute=1 --workload_type="dumpi" --
workload_conf_file=multiple-workloads.conf –alloc_fil --
modelnet-mpi-test-dragonfly.conf
```

- For multiple jobs, two of the arguments are different:
  - Workload_file: Has information on the dumpi/SWM traces for each application
  - Alloc_file: List of simulated MPI ranks to be assigned to each job

EXAMPLE WORKLOAD FILE

```
216 /path/to/AMG/df_AMG_n216_dumpi/dumpi-2014.03.03.14.55.23-
125 /path/to/Multigrid/MultiGrid_C_n125_dumpi/dumpi-2014.03.06.23.48.13-
100 /path/to/Crystal_Router/100/dumpi--2014.04.23.12.12.05-
~
```

```
2048 lammps
2197 nekbone
```

- Left column: Number of application ranks per job
- Right column: Path and prefix of DUMPI traces for each job – or name of SWM
  - Combination SWM and Dumpi slated for future
  - Currently can only combine DUMPI+DUMPI/Synthetic and SWM+SWM/Synthetic

EXAMPLE JOB ALLOCATION FILE

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 5
6 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
~
```

- List of MPI ranks for each job
- There is a job entry per line
- Example at the top shows job placement being done in a linear scheme
- Example below shows job placement in a random fashion (assumes one rank per node).

```
7022 6303 allocation_gen — vim — 189×55 7 6518 2522 3963 4851 7551 4196 2343 6285 5142 2083 7565 8172 6737
 7502 2579 6069 7474 5688 3926
834 3606 5076 7593 8038 3962 7194 2161 6692 4560 116 5981 3311 6857 5552 9 4101 7213 2022 2701
 7237 1584 4716 1978 8045 6677 3721 666 2658 4221 7758 904 4581 5872 4549 6771 4490 8015 5013
4883 3696 4955 3196 4783 2410 1570 1548 5092 5457 3958
~
```

GENERATING JOB ALLOCATIONS USING DIFFERENT SCHEMES

- Multiple schemes to map jobs onto the network
  - Randomly selected nodes
  - Contiguous or linear
  - Randomly selected switches (ranks ordered on nodes attached to a switch)
  - Clustered placement
  - …
- Scripts can be used to generate job allocation files with any of the above schemes
- CODES keeps track of the job ID and provides it in simulation output
- Some python scripts can be found in *scripts/allocation_gen*

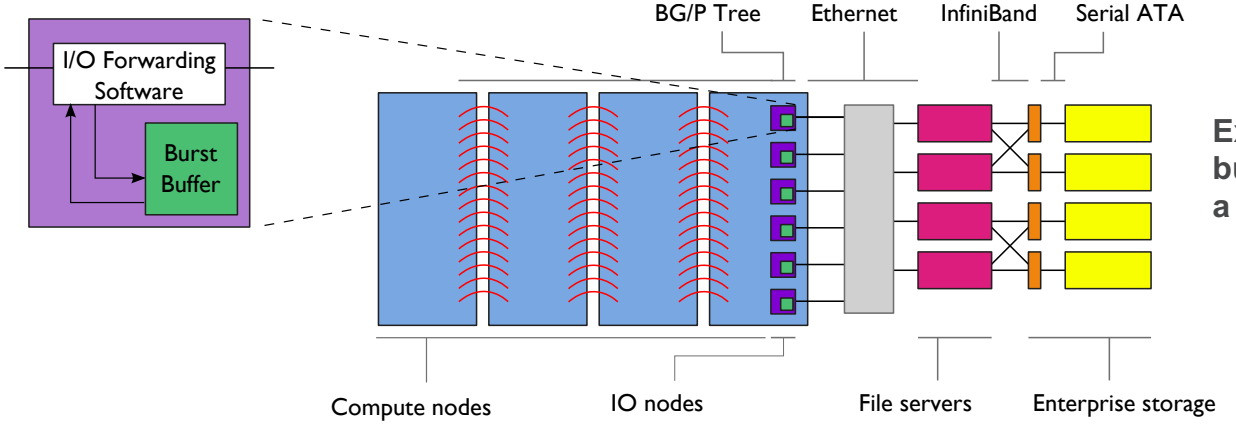
SESSION III: STORAGE MODELS AND SYNTHETIC TRAFFIC GENERATION

GOALS OF THE SESSION

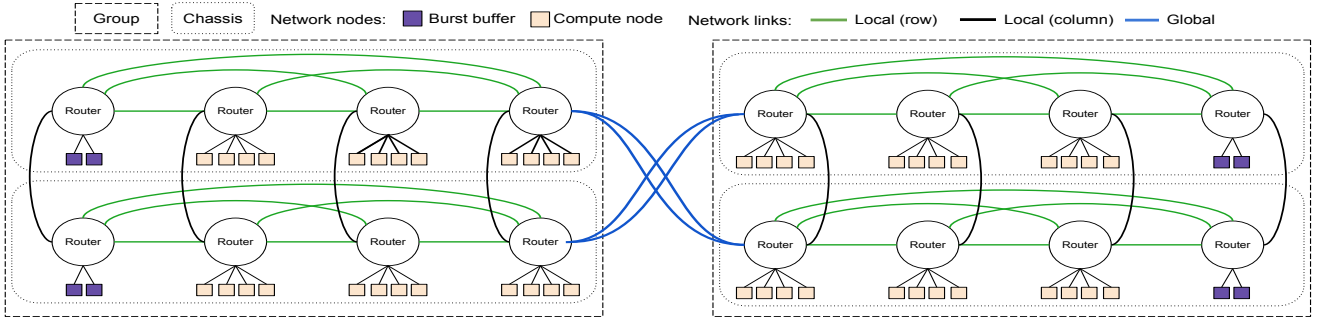
- How to do storage placement on networks?
- How to generate background network traffic?
- Using model-net API
- PDES and Networks Internal
- Continue with hands on exercises

STORAGE PLACEMENT ON INTERCONNECTS

STORAGE PLACEMENT ON HPC SYSTEMS



Example burst buffer placement on a Blue Gene system



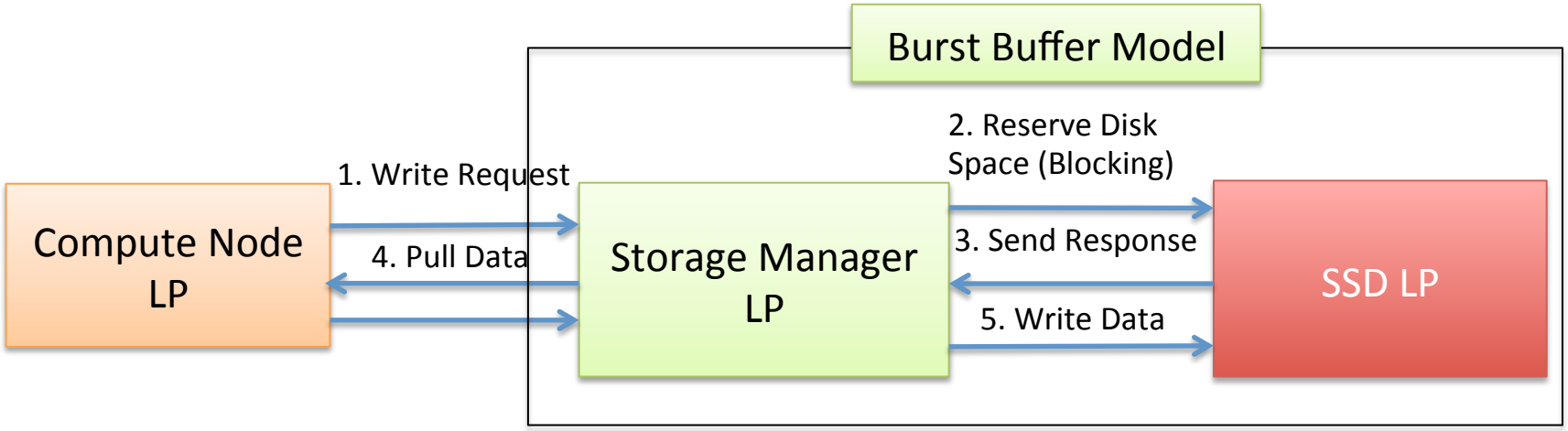
Example burst buffer placement on a dragonfly system

* Image credit: On the role of burst buffers in Leadership class storage system by N. Liu et al. in MSST 2012

MODELING BURST BUFFER WITH CODES

- General purpose model for read and write operations
- Concurrent, pipelined RDMA requests
- Comprises of the following:
  - a storage manager
  - a disk/local storage model
  - A resource tracker
- Placement of storage over the network can be modified using the network config file

PROTOCOL FOR WRITE OPERATIONS



USING THE STORAGE MODEL

- `codes_store_init_req (is_write, priority, obj_id, xfer_offset, xfer_size, codes_req)` → For initializing the request
- `codes_store_send_req(codes_req, dest_id, sender, network_id, mapping_context, ..)` → For sending the request
- `codes_store_send_req_rc` → For reverse computation

- Repo available at: `https://xgitlab.cels.anl.gov/codes/codes-storage-server`

CONFIGURING STORAGE OVER THE NETWORK

```
# triton server - implements forwarding and acking protocols based on placement
# algorithm
codes-store
{
    # number of threads to multiplex transfers over
    req_threads = "4";
    # buffer size used by each thread
    thread_buf_sz = "1000000";
    # size of ram in bytes
    memory_size = "64000000000";
    # size of storage in bytes
    storage_size = "6400000000000";
    # burst buffer threshold in bytes
    bb_threshold = "1000000000000";
}

# size of LP buffer
resource
{
    available="6464000000000";
}

# params taken from triton-fault-sim configs
lsm
{
    # table of metrics based on requset size
    # here metrics will be used for any request size
    # request size in bytes
    request_sizes   = ("0");

    # write/read rates in MB/s
    write_rates     = ("5700.0");
    read_rates      = ("5700.0");

    # seek latency in microseconds
    write_seeks     = ("2500.0");
    read_seeks      = ("2500.0");

    # latency of completing the smallest I/O request, in microseconds
    write_overheads = ("20.0");
    read_overheads  = ("20.0");
}
```

Number of concurrent requests

Buffer size for each thread

Size of the Memory (RAM)

Storage size (for disk/LSM)

Aggregate memory+storage size

Disk bandwidth/seek configuration

CONFIGURING STORAGE OVER THE NETWORK

```
LPGROUPS
{
    #dragonfly_router can be in one group only!
    DRAGONFLY_GRP
    {
        repetitions = "150";
        codes-store="2";
        lsm="2";
        resource="2";
        test-dummy = "2";
        test-checkpoint-client="60";
        modelnet_simplenet="1";
        modelnet_dragonfly_custom="64";
        modelnet_dragonfly_custom_router="16";
    }
    EXTERNAL_STR_GRP
    {
       repetitions="1";
       codes-external-store="1";
       modelnet_simplenet="1";
    }
}
```

Two storage manager entities per 60 clients/compute nodes (Cray Cori configuration)

Local storage model entity (disk). One to one correspondence

Dummy nodes are to balance node to router ratio for BB routers

A total of 64 network nodes

If the data from burst buffer needs to be drained to the external storage entity

GENERATING BACKGROUND NETWORK TRAFFIC

WHY BACKGROUND TRAFFIC?

- On production HPC systems, a significant fraction of network nodes can be occupied
- How to introduce communication interference if a single application trace is being replayed on the simulation?
- Running multiple traces at a large-scale can be expensive
- One solution is to mix synthetic traffic patterns and HPC application traces

EXAMPLE SYNTHETIC PATTERNS

- Uniform Random: A network node is equally likely to send to any other network node (traffic distributed throughout the network)
- All to All: Each network node communicates with all other network nodes
- Nearest neighbor: A network node communicates with near by network nodes (or the ones that are at minimal number of hops)
- Permutation traffic: Source node sends all traffic to a single destination based on a permutation matrix
- Bisection pairing: Node 0 communicates with Node ‘n’, node 1 with ‘n-1’ and so on.
- …

SYNTHETIC TRAFFIC IN CODES

```
   /* in case of uniform random traffic, send to a random destination. */
   if(traffic == UNIFORM)
   {
    b->c1 = 1;
    local_dest = tw_rand_integer(lp->rng, 0, num_nodes - 1);
   }
   else if(traffic == NEAREST_GROUP)
   {
    local_dest = (local_id + num_nodes_per_grp) % num_nodes;
    //printf("\n LP %ld sending to %ld num nodes %d ", local_id, local_dest, num_nodes);
   }
   else if(traffic == NEAREST_NEIGHBOR)
   {
    local_dest =  (local_id + 1) % num_nodes;
//    printf("\n LP %ld sending to %ld num nodes %d ", rep_id * 2 + offset, local_dest, num_nodes);
   }
   assert(local_dest < num_nodes);
//    codes_mapping_get_lp_id(group_name, lp_type_name, anno, 1, local_dest / num_servers_per_rep, local_dest % num_servers_per_rep, &global_dest);
   global_dest = codes_mapping_get_lpid_from_relative(local_dest, group_name, lp_type_name, NULL, 0);
   ns->msg_sent_count++;
   model_net_event(net_id, "test", global_dest, PAYLOAD_SZ, 0.0, sizeof(svr_msg), (const void*)m_remote, sizeof(svr_msg), (const void*)m_local, lp);
```

*** Code snippet from synthetic workload generator**

- Typical patterns supported are uniform random and nearest neighbor.
- All to all and stencil patterns have been tested (pending integration)
- See `src/network-workloads/model-net-synthetic-custom-dfly.c and related files`

GENERATING BACKGROUND TRAFFIC WITH CODES

- Communication based on uniform random traffic
- Kicks off when the main workload starts
- A notification is sent to the background traffic node to stop generating traffic once the main workload finishes
- How to enable synthetic traffic generation?
- Simply add “synthetic” instead of DUMPI trace path in workloads config file

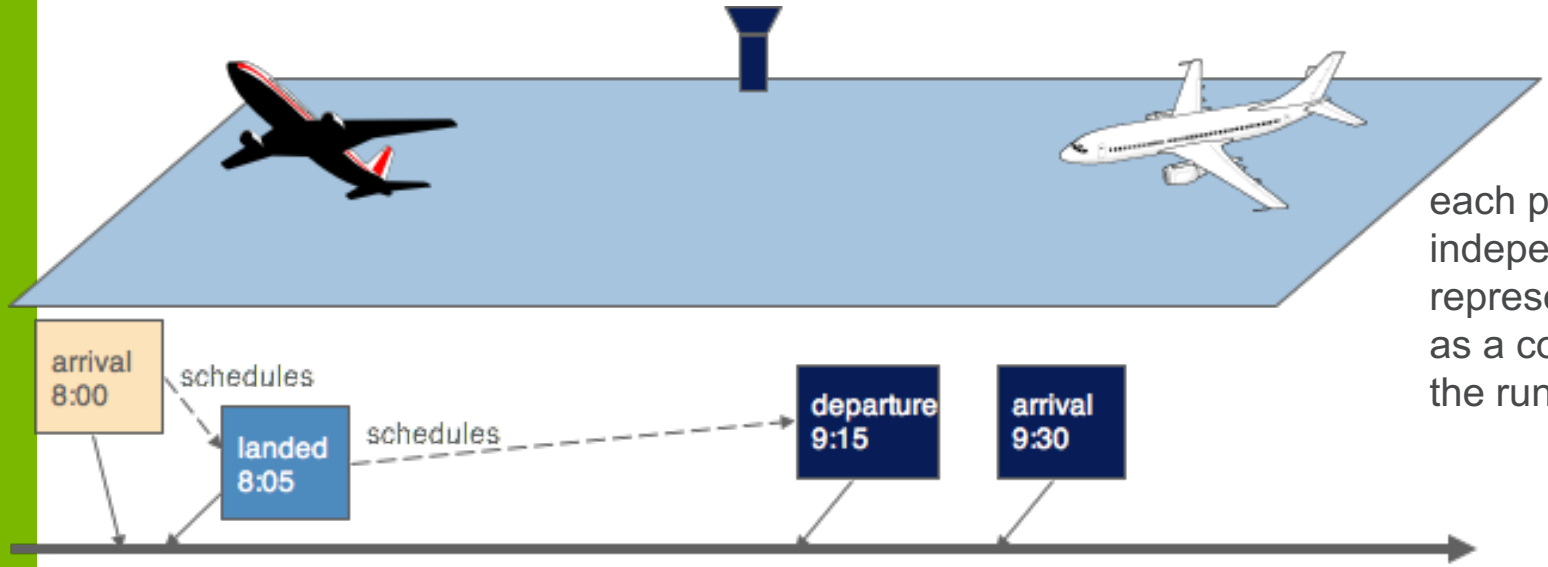
```
216 synthetic
125 /path/to/Multigrid/Multigrid_125/dumpi-2014.03.06.23.48.13-
```

PDES AND NETWORK INTERNALS

DISCRETE EVENT SIMULATION (DES)

- Computer model for a system where changes in the state of the system occur at discrete points in simulation time
- In this model, each component of the system being simulated is represented independently via their
- State variables
- Virtual time
- Events - scheduled on it and by it

DES EXAMPLE: AIR TRAFFIC

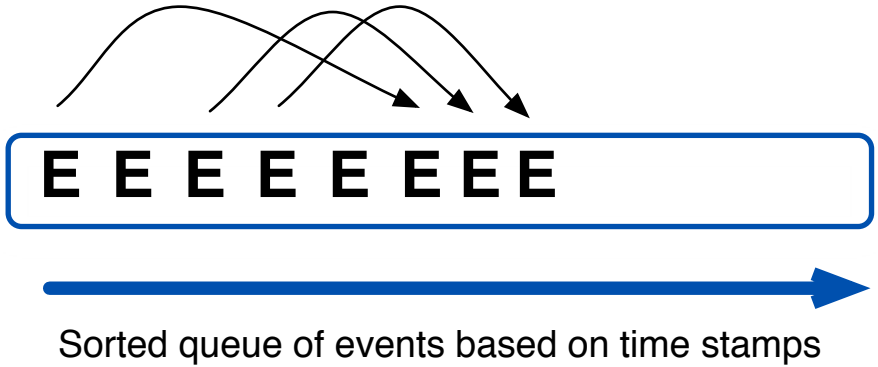


each plane is independently represented as a component, so is the runway

Event scheduling from one component to another progresses and coordinates virtual time across components

Example from slides by Prof Carothers, RPI

IMPLEMENTING DES



- ROSS lets users define LP (logical processes) on which events can be scheduled with time stamps
- Each LP can have a local state that is accessible and modified only when events are executed on it

ROSS LP

```
tw_lptype model_lps[] = {
  {
    (init_f) model_init,
    (event_f) model_event,
    (revent_f) model_event_reverse,
    (final_f) model_final,
    (map_f) model_map,
    sizeof(state)
  },
};
```

EXAMPLE OF AN EVENT FUNCTION

- Typical events act based on “type” and “content” of the message

```
static void svr_event(
    svr_state * ns,
    tw_bf * b,
    svr_msg * m,
    tw_lp * lp)
{
    (void)b;
   switch (m->svr_event_type)
    {
        case REQ:
            handle_req_event(ns, m, lp);
            break;
        case ACK:
            handle_ack_event(ns, m, lp);
            break;
        case KICKOFF:
            handle_kickoff_event(ns, m, lp);
            break;
        case LOCAL:
           handle_local_event(ns);
```

```
static void handle_req_event(
    svr_state * ns,
    svr_msg * m,
    tw_lp * lp)
{
    assert(!do_pull);
    svr_msg * m_local = malloc(sizeof(svr_msg));
    svr_msg * m_remote = malloc(sizeof(svr_msg));

    m_local->svr_event_type = LOCAL;
    m_local->src = lp->gid;

    memcpy(m_remote, m_local, sizeof(svr_msg));
    m_remote->svr_event_type = ACK;

    ns->msg_recvd_count++;

    m->ret = model_net_event(net_id, "test", m->src, PAYLOAD_SZ,
                0.0, sizeof(svr_msg), (const void*)m_remote,
                sizeof(svr_msg), (const void*)m_local, lp);
    return;
}
```

ROSS'S LAW OF OPTIMISTIC EXECUTION: FOR EVERY FORWARD ACTION, YOU MUST TELL ROSS HOW TO GO BACKWARDS

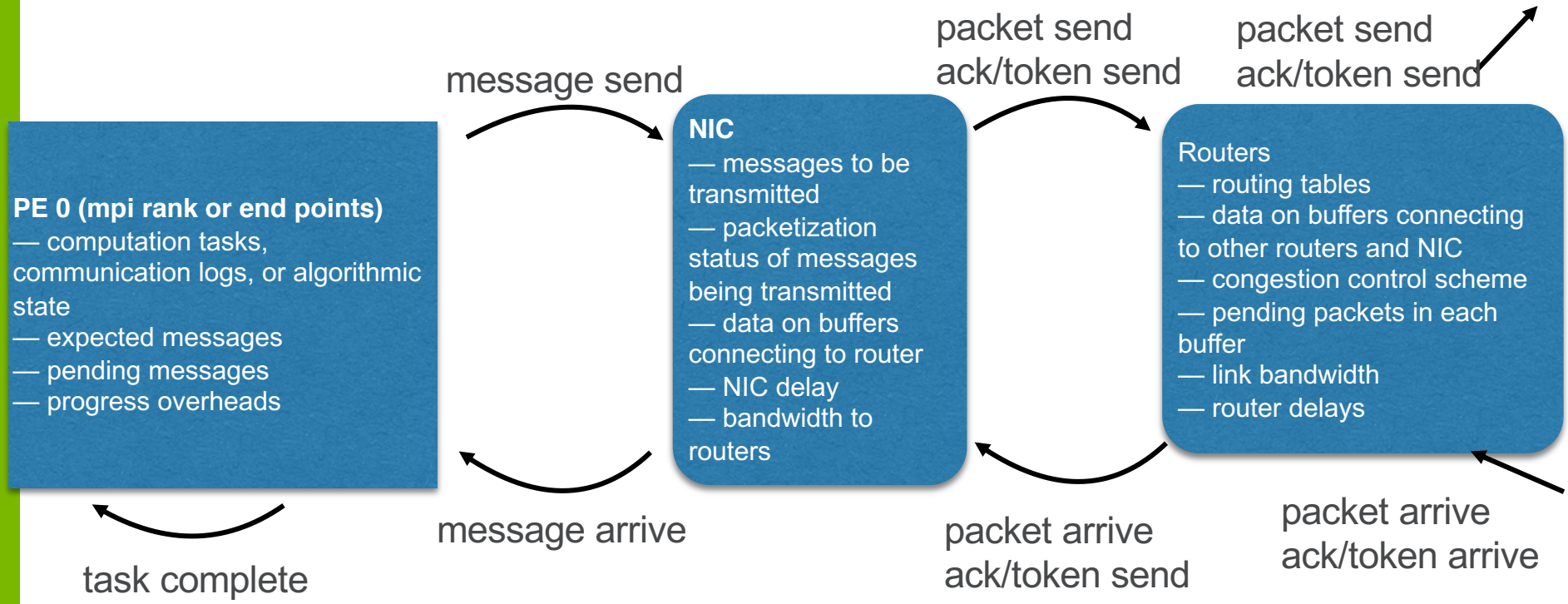
```
static void svr_event(
    svr_state * ns,
    tw_bf * b,
    svr_msg * m,
    tw_lp * lp)
{
    (void)b;
   switch (m->svr_event_type)
    {
        case REQ:
            handle_req_event(ns, m, lp);
            break;
        case ACK:
            handle_ack_event(ns, m, lp);
            break;
        case KICKOFF:
            handle_kickoff_event(ns, m, lp);
            break;
        case LOCAL:
           handle_local_event(ns);
```

```
static void svr_rev_event(
    svr_state * ns,
    tw_bf * b,
    svr_msg * m,
    tw_lp * lp)
{
    (void)b;
    switch (m->svr_event_type)
    {
        case REQ:
            handle_req_rev_event(ns, m, lp);
            break;
        case ACK:
            handle_ack_rev_event(ns, m, lp);
            break;
        case KICKOFF:
            handle_kickoff_rev_event(ns, m, lp);
            break;
        case LOCAL:
            handle_local_rev_event(ns);
            break;
```

```
static void handle_req_event(
    svr_state * ns,
    svr_msg * m,
    tw_lp * lp)
{
    assert(!do_pull);
    svr_msg * m_local = malloc(sizeof(svr_msg));
    svr_msg * m_remote = malloc(sizeof(svr_msg));

    m_local->svr_event_type = LOCAL;
    m_local->src = lp->gid;

    memcpy(m_remote, m_local, sizeof(svr_msg));
    m_remote->svr_event_type = ACK;

    ns->msg_recvd_count++;

    m->ret = model_net_event(net_id, "test", m->src, PAYLOAD_SZ,
                0.0, sizeof(svr_msg), (const void*)m_remote,
                sizeof(svr_msg), (const void*)m_local, lp);
    return;
}
```

```
static void handle_req_rev_event(
    svr_state * ns,
    svr_msg * m,
    tw_lp * lp)
{
    ns->msg_recvd_count--;
    if (do_pull){
        model_net_event_rc2(lp, &m->ret);
    }
    else{
        model_net_event_rc2(lp, &m->ret);
    }
```

APPLICATION SIMULATION IN CODES



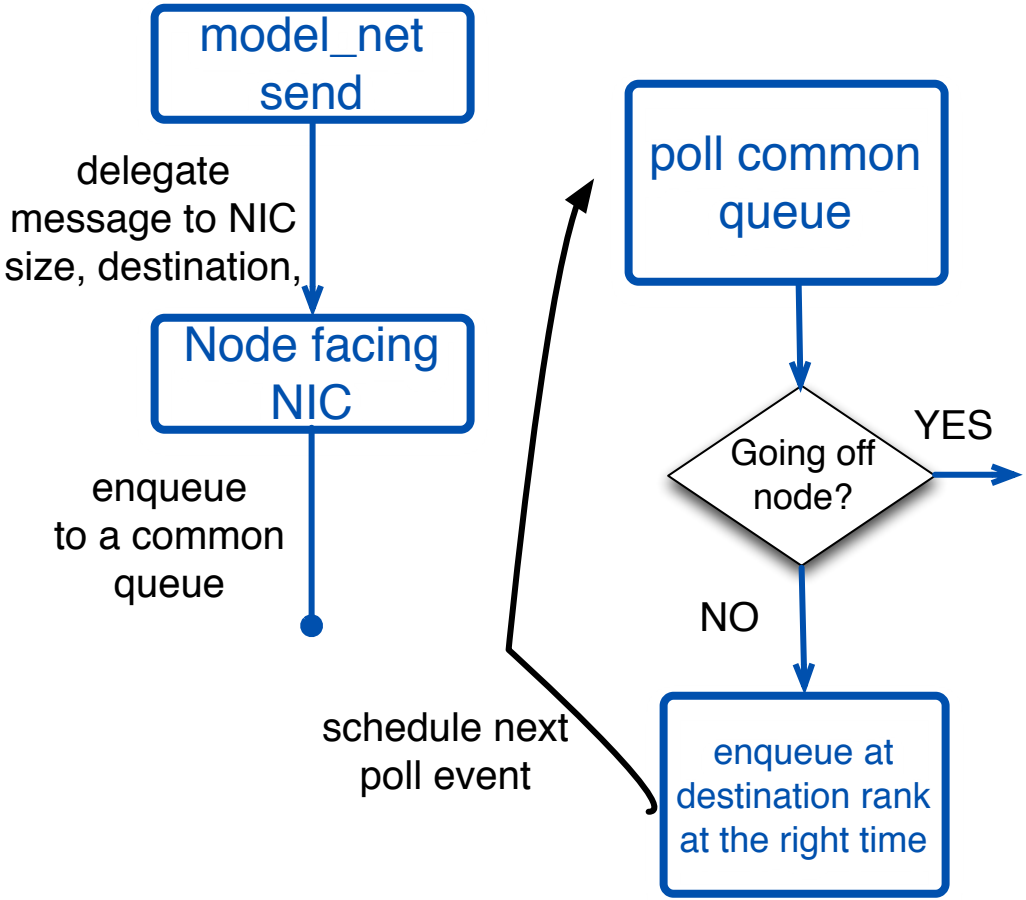
AVAILABLE MODELS, FEATURES, AND ADDING A NEW NETWORK MODELS

- Available: simple-net model, torus, dragonfly-(custom), fat-tree, slim fly, express-mesh/hyperX
- Typical model consist of NIC (terminals) and switches/routers
- NICs
  - Common code available for within-node, message ordering, etc
  - Plugin code for individual network
- Switch/routers
  - Entirely within a network model
- But, a significant fraction of node is similar for NIC plugin and switch!

NIC COMMON

- Types of queues:
- fifo
- round-robin
- priority

- Other params
- intra_bandwidth (10)
- node_copy_queues (4)

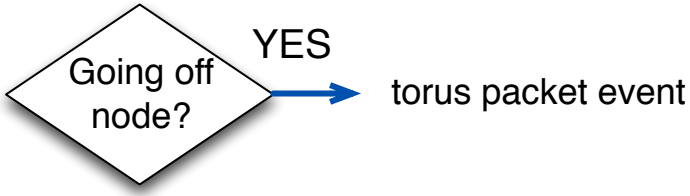


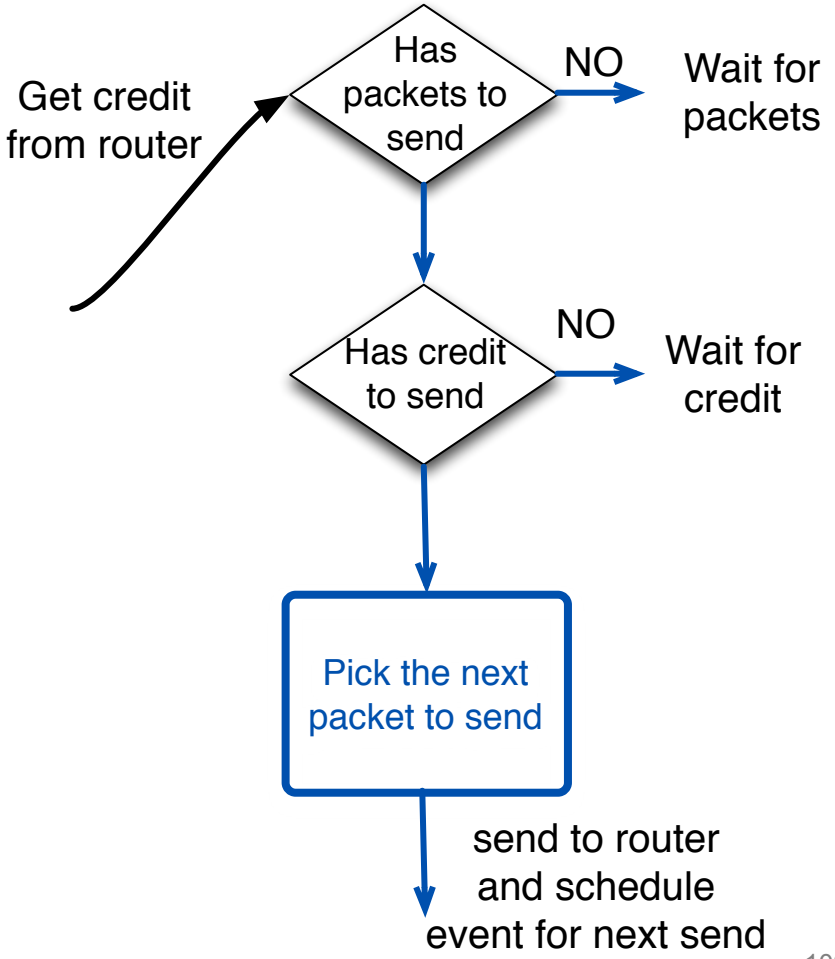
NIC NETWORK SPECIFIC

```
struct model_net_method torus_method =
{
    .mn_configure = torus_configure,
    .mn_register = NULL,
    .model_net_method_packet_event = torus_packet_event,
    .model_net_method_packet_event_rc = torus_packet_event_rc,
    .model_net_method_recv_msg_event = NULL,
    .model_net_method_recv_msg_event_rc = NULL,
    .mn_get_lp_type = torus_get_lp_type,
    .mn_get_msg_sz = torus_get_msg_sz,
    .mn_report_stats = torus_report_stats,
    .mn_collective_call = NULL,
    .mn_collective_call_rc = NULL,
    .mn_sample_fn = NULL,
    .mn_sample_rc_fn = NULL,
    .mn_sample_init_fn = NULL,
    .mn_sample_fini_fn = NULL
};
```

```
tw_lptype torus_lp =
{
     (init_f) torus_init,
     (pre_run_f) NULL,
     (event_f) event_handler,
     (revent_f) node_rc_handler,
     (commit_f) NULL,
     (final_f) final,
     (map_f) codes_mapping,
     sizeof(nodes_state),
};
```

NIC NETWORK SPECIFIC





CONTRIBUTING:

- Fork off on the github repository
- Add new features
- Submit a pull request!

THANK-YOU

ADDITIONAL MATERIAL

CODES INSTALLATION

INSTALLATION & SETUP 1/2

- **ROSS INSTALLATION**
  - Download ROSS repo: ***git clone https://github.com/ROSS-org/ROSS***
  - Configure by making a build directory*: `cd build`*
  - *`ARCH=x86_64 CC=mpicc CXX=mpicxx cmake -DCMAKE_INSTALL_PREFIX=../install ../`*
  - *make –j 3 && make install*
- **CODES INSTALLATION**
  - Download CODES repo: git clone https://github.com/codes-org/codes
  - `./prepare.sh`
  - Configure in build directory : `cd build`

`../configure --prefix=/path/to/install CC=mpicc CXX=mpicxx PKG_CONFIG_PATH=/path/to/ross/install/lib/pkgconfig`

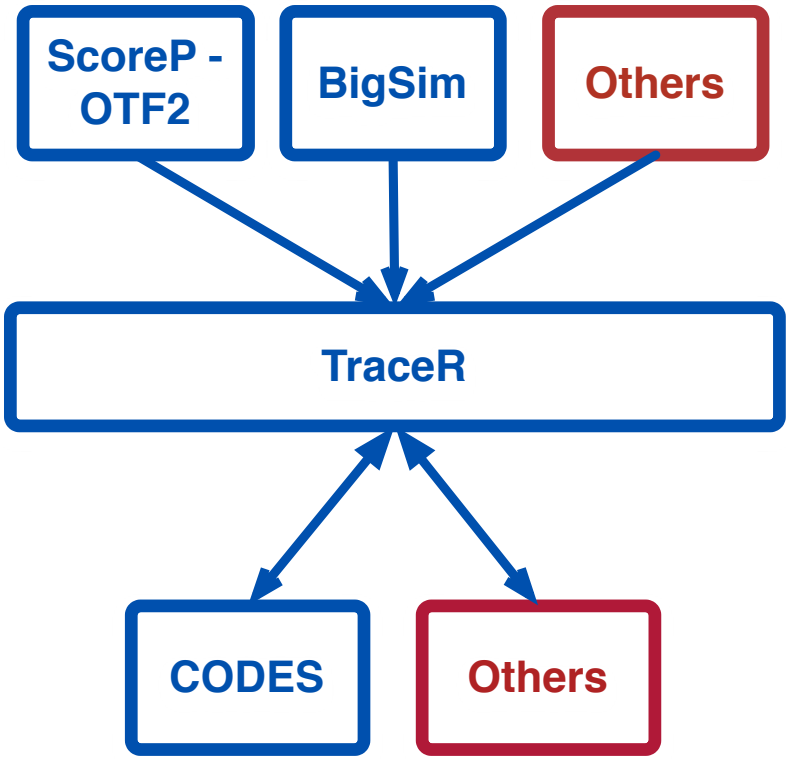
  - *Do both make && make tests*

INSTALLATION & SETUP 2/2

- **DUMPI INSTALLATION**
  - *git clone https://github.com/sstsimulator/sst-dumpi*
  - *CFLAGS="-DMPICH_SUPPRESS_PROTOTYPES=1 -DHAVE_PRAGMA_HP_SEC_DEF=1"*
  - *./bootstrap.sh*
  - *./configure --enable-libundumpi CC=mpicc --prefix=$INSTALL_PATH*
  - *Use –with-dumpi=/path/to/dumpi/install option to enable DUMPI with CODES*
- **OTF2 and BigSim based Tracing**
  - **ScoreP/OTF2:** http://www.vi-hps.org/projects/score-p/
  - **Charm++/BigSim:**http://charm.cs.illinois.edu/manuals/html/bigsim/manual.html
  - **TraceR**: https://github.com/LLNL/tracer/
- *For installation details and documentation see:*
  - *https://xgitlab.cels.anl.gov/codes/codes/wikis/home*
  - *https://xgitlab.cels.anl.gov/codes/codes/wikis/installation*

MORE ON TRACER

TRACER – A LAYER FOR CONFIGURABLE REPLAY OF APPLICATION TRACES



Capture application behavior by tracing runs on existing systems

Reproducing the execution: applications' behavior, job placement and mapping, job scheduling, MPI/Charm++, etc.

Simulation of traffic flow on NICs and networks

DOCUMENTATION

- Distributed with TraceR source code
- README.md – getting started
- README.OTF - OTF2 installation and usage
- docs/UserWriteUp.txt – detailed workflow and usage
- utils/README – job placement and task mapping

INSTALLING TRACER (1/4)

- Hosted on github: https://github.com/LLNL/tracer/
- git clone and follow README.md
- Download and install ROSS
  - Last verified commit provided
- Download and install CODES
  - Last verified commit provided

INSTALLING TRACER (2/4)

- Choose a trace format: BigSim or OTF2
- For BigSim, download Charm++
  - git clone http://charm.cs.uiuc.edu/gerrit/charm
- Assuming MPI is available, install two flavors of Charm++
  - For compiling codes for trace generation
    ./build bgampi mpi-linux-x86_64 bigemulator –O2
  - For compiling TraceR
    ./build charm++ mpi-linux-x86_64 bigemulator --with-production

INSTALLING TRACER (3/4)

- For OTF2, download ScoreP
  - http://www.vi-hps.org/projects/score-p/
- Inside ScoreP source directory
  - CC=mpicc CFLAGS="-O2" CXX=mpicxx CXXFLAGS="-O2" FC=mpif90 F77=mpif77 ./configure --without-gui --prefix=<SCOREP_INSTALL>
  - make && make install
- Make sure ScoreP installation's bin directory is in PATH

INSTALLING TRACER (4/4)

- In tracer/Makefile.common
- Set ROSS to ROSS's installation directory
- Set CODES to CODES's installation directory
- If using BigSim,
  - Set CHARMPATH
  - SELECT_TRACE = -DTRACER_BIGSIM_TRACES=1
- If using OTF2,
  - Make sure ScoreP installation's bin directory is in PATH
  - SELECT_TRACE = -DTRACER_OTF_TRACES=1
- make: generates traceR executable

MORE ON GENERATING OTF2 TRACES

- ScoreP macros can be used to mark special regions
  - SCOREP_USER_REGION_BY_NAME_BEGIN(regionname, SCOREP_USER_REGION_TYPE_COMMON)
  - SCOREP_USER_REGION_BY_NAME_END(regionname)
- Printing simulation time at locations of interest:
- Region name with prefix TRACER_WallTime_<any_name> prints current time during simulation with tag <any_name>.

MORE ON GENERATING OTF2 TRACES

- Simulation time looping:
- Region name TRACER_Loop can be used to mark beginning and ending of a code loop (currently once)

- In future, region names will be used for
  - Targeted kernel time modifications
  - Targeted message size modifications

CODE EXAMPLE FOR TRACING WITH OTF2

```
int main(int argc, char **argv) {
  MPI_Init(&argc, &argv);
  SCOREP_RECORDING_OFF();
//initializaton code
  MPI_Barrier(MPI_COMM_WORLD);
  SCOREP_RECORDING_ON();
  SCOREP_USER_REGION_BY_NAME_BEGIN
("TRACER_Loop", SCOREP_USER_REGION_TYPE_COMMON);
  if(!myRank)
    SCOREP_USER_REGION_BY_NAME_BEGIN
("TRACER_WallTime_Total", SCOREP_USER_REGION_TYPE_COMMON);
  startTime = MPI_Wtime();
  while(iterations < MAX_ITER) {
    if(myRank == 0)
      SCOREP_USER_REGION_BY_NAME_BEGIN
("TRACER_WallTime_InLoop",
SCOREP_USER_REGION_TYPE_COMMON);
//kernel and other code
  }
  SCOREP_USER_REGION_BY_NAME_END
("TRACER_Loop");
  MPI_Barrier(MPI_COMM_WORLD);
  endTime = MPI_Wtime();
  if(!myRank)
    SCOREP_USER_REGION_BY_NAME_END
("TRACER_WallTime_Total");
  SCOREP_RECORDING_OFF();
```

RUNNING TRACER

- A typical run command:
  mpirun -np 8 ./traceR --sync=3 --nkp=16 --extramem=100000 --max-opt-lookahead=1000000 --timer-frequency=1000 –lp-io-dir=stats-dir -- torus.conf tracer_config
- In green, ROSS options
  - --nkp : how many KPs to create per PE = total LPs/<-np>
  - --extramem : how many ROSS messages to allocate = 100K should work for most cases
  - --max-opt-lookahead : optimistic leash = 1 millisecond is a good number

RUNNING TRACER

- A typical run command:
  mpirun -np 8 ./traceR --sync=3 --nkp=16 --extramem=100000 --max-opt-lookahead=1000000 --timer-frequency=1000 –lp-io-dir=stats-dir -- torus.conf tracer_config
- TraceR-CODES options
  - --timer-frequency : how frequently to print progress of task completion; optional, default = 5000
  - --lp-io-dir : where to write output stats; optional; code fails if the directory exists to avoid over-writing
  - torus.conf : network config file
  - tracer_config : TraceR config file

TRACER PARAMETER IN NETWORK FILE

- server in MODELNET_GRP
  - Number of processes associated with a switch
  - Assigned in a round-robin manner to nodes
- soft_delay in PARAMS
  - Approximate overhead of making an MPI/runtime call
  - In nanoseconds
- rdma_delay in PARAMS
  - Overhead of using RDMA call in rzv protocol, in nanoseconds
- eager_limit in PARAMS
  - Switch over point between eager and rzv protocols, in bytes
- copy_per_byte in PARAMS
  - Copy cost for a byte, in nanoseconds per byte

TRACER CONFIG FILE (1/2)

Format:

<global map file> or NA

#jobs

<path to job traces> <task mapping file or NA> <#ranks> <loop iterations>

Example:

global_map.bin

2

traces-64/traces.otf2 job0 64 1

traces-32/traces.otf2 job1 32 1

TRACER CONFIG FILE (2/2)

- At the end of file,
- E <job id> scale_all <scale factor>
  - Inverse scales computation time by the given factor
  - E.g. : E 0 scale_all 40
- S <job id> <msg size> <replace by>
  - Change the size of message
  - Under review, to be merged

SAMPLE OUTPUT

PE0 - LP_GID:0 : START SIMULATION, TASKS COUNT: 245611, FIRST TASK: 0, RUN TIME TILL NOW=70.000000 s, CURRENT SIM TIME 1.005877

[0 0 : time at task 0/245611 0.000000]

[0 0 : Begin TRACER_WallTime_MainLoop 0.000001]

[0 0 : time at task 100/245611 0.000663]

[0 0 : Begin TRACER_WallTime_NextTrajec 0.001175]

[0 0 : time at task 200/245611 0.003104]

….

….

[0 0 : time at task 245600/245611 1.485123]

[0 0 : End TRACER_WallTime_NextTrajec 1.485264]

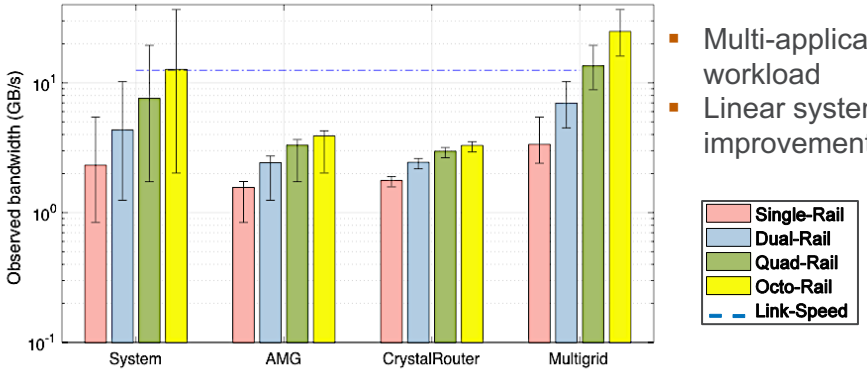
[0 0 : End TRACER_WallTime_MainLoop 1.485265]

CASE STUDIES WITH CODES-TRACER

Evaluating Methods for Effective Fat-tree Utilization

- Quantify the effectiveness of multiple network rails in increasing network performance
- Evaluate job placement methodologies and routing choices to maximize the throughput of a multi-rail fat-tree network
- Investigate network performance gains offered by multiple rails in response to increased compute performance of high-density compute nodes
- Improve routing, injection and job allocation design choices of multi-rail fat-tree networks to maximize individual application and system wide performance

Fat-tree strong scaling rail performance

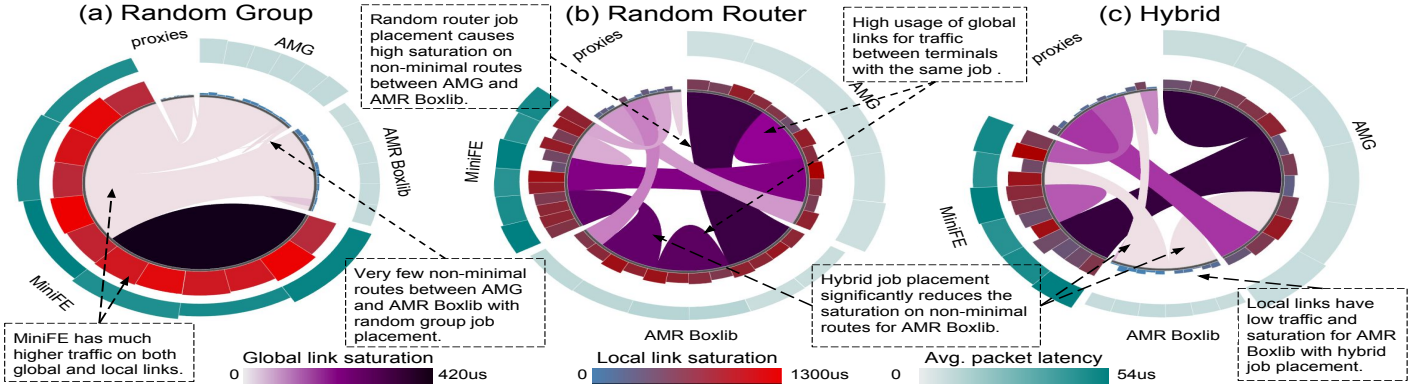


- Multi-application workload
- Linear system improvement

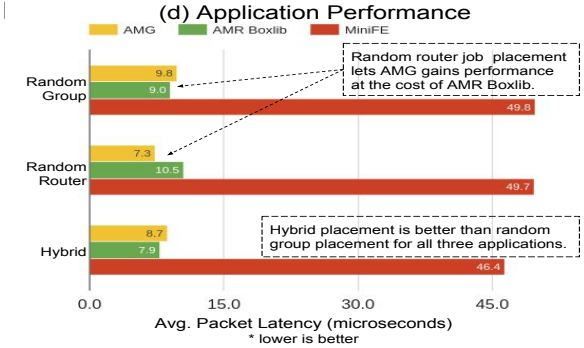
Related papers

- Jain et al. Predicting the Performance Impact of Different Fat-Tree Configurations (To appear in Supercomputing 2017)
- Wolfe et al. Preliminary Performance Analysis of Multi-Rail Fat-tree Networks (CCGRID 2017)

VISUALIZING HPC INTERCONNECT PERFORMANCE



Comparing job placement schemes using the CODES simulation framework of the high-radix dragonfly network running the Algebraic Multigrid Solver (AMG), AMR BoxLib and MiniFE applications. The visualizations show the aggregated [illegible]twork links.

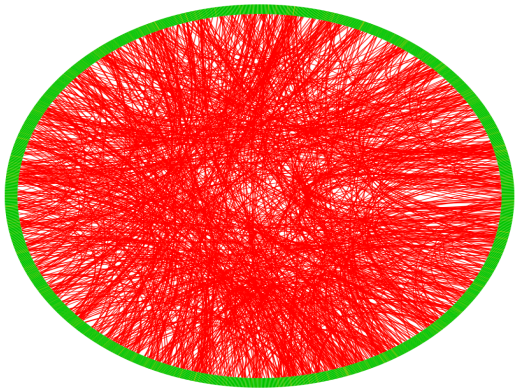


Application	Ranks	Data	Comm. Pattern
AMG	1728	1.2GB	3D nearest neighbor
AMR Boxlib	1728	2.2GB	Irregular and sparse
MiniFE	1152	147GB	Many-to-many

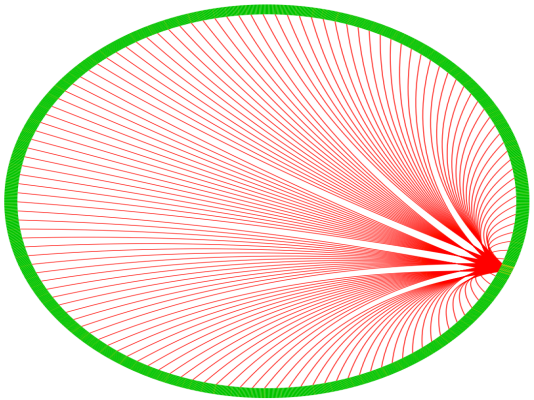
- *K. Li, M. Mubarak, R. Ross et al.* ***"Visual Analytics Techniques for Exploring the Design-space of large-scale high-radix networks"****, to appear in IEEE Cluster 2017*

ACCELERATING COLLECTIVE COMMUNICATION ON DRAGONFLY NETWORKS

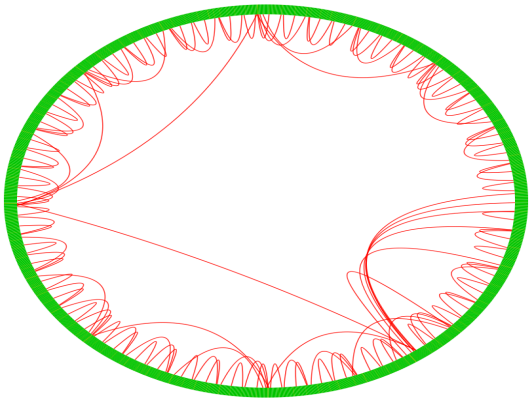
- Need to design new scalable collective algorithms for high-radix interconnects
- Investigate the parameter space of these algorithms, and the effects of cross-application communication interference
- Designed/evaluated several collective algorithms for Dragonfly networks



(a) Tree (Non topology aware)



(b) Local Links First (topology-aware)



(c) Global Links First

M. Dorier et al. ***"Evaluation of Topology-Aware Broadcast Algorithms for Dragonfly Networks"****, IEEE Cluster 2016*

USING MODEL-NET API

CONFIGURATION

- Model-net– An abstraction layer on top of network models – topology details are specified through the config files
- A valid network configuration file – examples can be found in the repo
- Network model must be registered – `model_net_register`
- CODES mapping must be setup – `codes_mapping_setup`
- Use model-net function calls – `model_net_event(network id, source, destination, message size,…)`
- Example of using model-net – tests/model-net-test.c