Hewlett Packard Enterprise

PROGRAMMING SLINGSHOT AT SCALE

Keith D. Underwood
Senior Distinguished Technologist

August 1, 2022

OVERVIEW

Interconnect Trends

- Bandwidth trends:
  - Red Storm (2005): 1.1 GB/s/direction
  - Frontier (2022): 100 GB/s/direction
- Compute trends:
  - Red Storm (2005): 4 GF node
  - Frontier (2022): 180 TF node
- Global bandwidth is expensive
  - New topologies have improved bisection bandwidth relative to injection
  - Ratios won't be changing again soon
- Transistors are cheap (but not free)
  - Networks will need to be smarter to cover gaps
  - NIC offloads and switch optimizations will continue
- Supercomputer networks are converging with datacenter networks

Implications for Applications

- Locality matters more
  - Keep the work on the node
  - Be aware of how work is placed on the system
- ***Program for Overlap***
  - Use the network all the time – not just in "communication phase" bursts
  - Use expected messaging
- Pay attention to application layouts
- Use the offloads the hardware gives you
  - Matching in hardware
  - Enabling of overlap
- The future: direct connection of instruments to supercomputers over distance

INTERCONNECT TECHNOLOGY

SLINGSHOT BRINGS HPC TO ETHERNET – AND ETHERNET TO EXASCALE

CRAY SLINGSHOT

Traditional Ethernet Networks

- Ubiquitous & interoperable
- Broad connectivity ecosystem
- Broadly converged network
- Native IP protocol
- Efficient for large payloads only
- High latency
- Limited scalability for HPC
- Limited HPC features

Standards based / interoperable
Broad connectivity
Converged network
Native IP Support
Low latency
Efficient for small to large payloads
Full set of HPC features
Very scalable for HPC & Big Data

Traditional HPC Interconnects

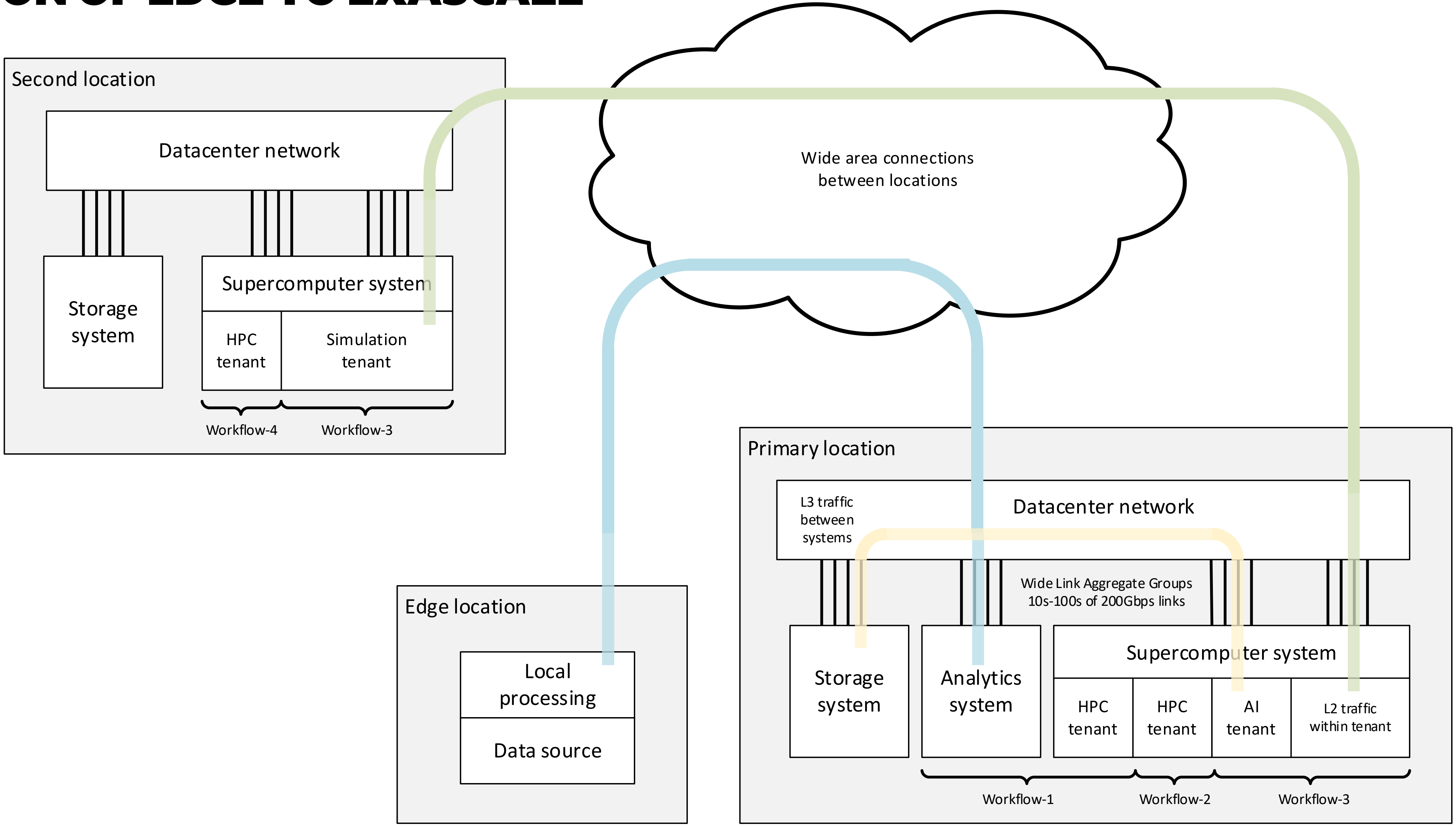
- Proprietary (single vendor)
- Limited connectivity
- HPC interconnect only
- Expensive/ slow gateways
- Low latency
- Efficient for small to large payloads
- Full set of HPC features
- Very scalable for HPC & Big Data

Consistent, predictable reliable high performance
High bandwidth + low latency, from one rack to Exascale

Excellent for emerging infrastructures
Mix tightly-coupled HPC, AI, analytics, and cloud workloads

Native connectivity to data center resources

A VISION OF EDGE TO EXASCALE



THE FIRST EXASCALE ETHERNET NETWORK

Slingshot Components

- Ethernet (IEEE 802.3cd) links with Slingshot specific additions for performance and reliability
  - Rosetta switch ASIC 64×200Gbps
  - Cassini NIC ASIC: Two NICs @200Gbps each
- Embedded management and monitoring system
  - 2368 management agents (one per switch) in Frontier
  - Management Ethernet network 1Gbps per switch
- HPE/Cray programming environment
  - HPE/Cray MPI
  - Libfabric provider for Cassini

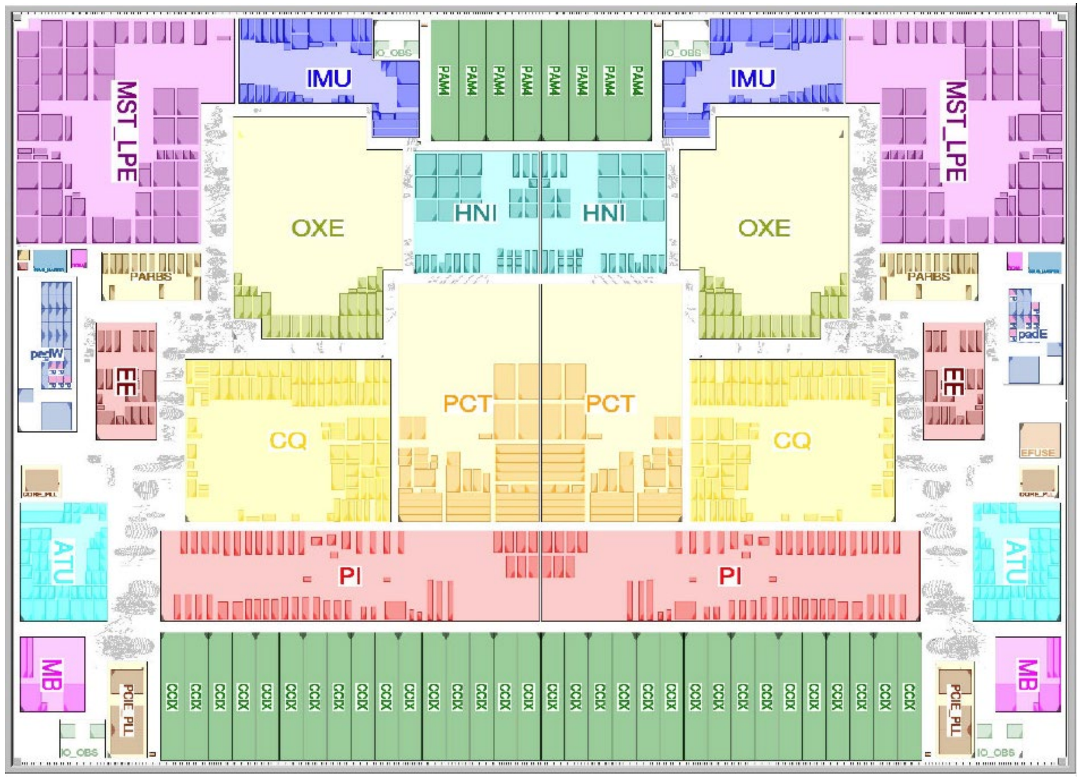
Frontier Slingshot Fabric

- Compute nodes
  - 9408 compute nodes each with 4 Cassini NICs
  - Peak bandwidth of 800Gbps/dir per node
- Dragonfly network 74 groups of 32 Rosetta switches
  - 37,632 Cassini NICs (18,816 ASICs)
  - 18,816 L0 cable assemblies (2 links each)
  - 9,472 L1 cable assemblies (4 links each)
  - 5,402 L2 active optical cables (2 links each).
  - 161,844 active 200Gbps ports
  - \+ 5 storage groups, each 292 ports
- Network properties
  - Injection bandwidth of 800 Gbps per node
  - Local link bandwidth is 200% of injection
  - Global link bandwidth is 57% of injection

The Aurora Slingshot Fabric is even bigger!

CASSINI OVERVIEW

- Dual-NIC die: two network links of 200 Gbps each
  - PCIe Gen4 x16 host interface with Extended Speed Mode
  - This is a system packaging optimization
- Provides full HPC offloads
  - MPI Tag matching for both expected and unexpected messages
  - Offloads eager and rendezvous transfers with strong progress in all cases
  - PGAS specific optimizations to reduce overheads and improve message rates
  - Includes end-to-end reliability logic
- Architecture optimized for HPC traffic
  - Integrates with switch-based collectives
  - Has large translation cache with multiple page sizes



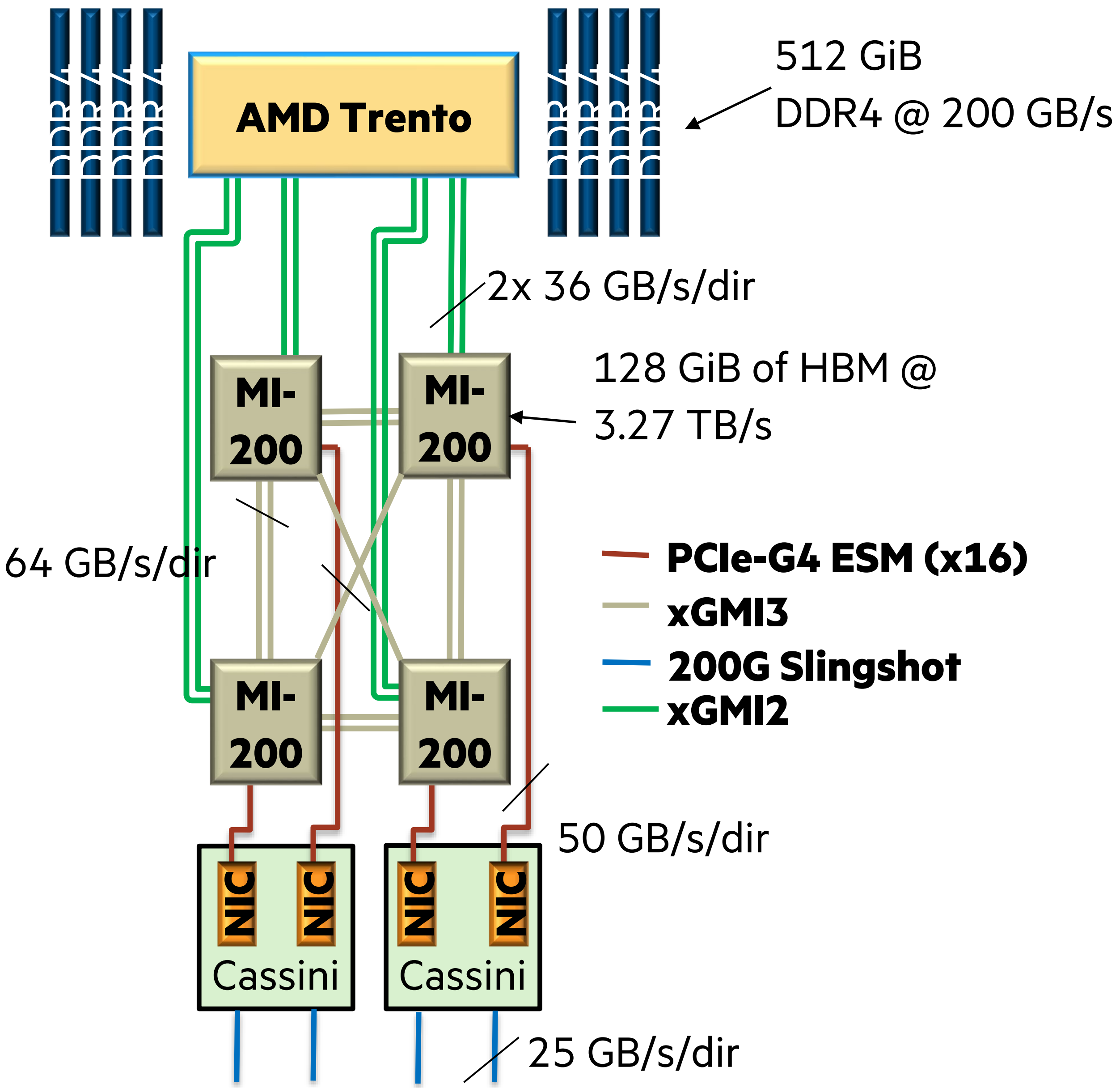
- Provides Ethernet and HPC functionality concurrently
  - Standard Ethernet Linux driver with support for NAPI, GSO, and GRO
  - Includes checksum and flow steering offloads for IP and SoftRoCE drivers
- Standard Ethernet (IEEE 802.3) frames for Ethernet
- HPC Optimized frames for HPC

LIBFABRIC: A MULTI-VENDOR NETWORK API

- The network API sits below the API the user sees (e.g., MPI, OpenSHMEM)
  - Exposes the capabilities of the network
  - Historically, this has changed with every generation of network
- Libfabric was introduced to provide a portable target for networks and middleware
  - Based on Portals 4 and influenced by PSM
  - Originally designed for OmniPath
  - Now shipping from HPE, AWS, Cornelis, others
- Includes key semantics for MPI, PGAS, and filesystems
  - One-sided operations (Put/Get)
  - Two-sided operations (send/recv) including matching
  - Atomic operations
  - Triggered operations
  - Locally managed offsets

NODE ARCHITECTURE

- A Frontier node (right) has a NIC on each GPU
  - Each GPU has significant affinity to one NIC
  - The CPU is further from all NICs
- Four NICs means 4x the bandwidth, but also:
  - 4x the amount of processing needed to drive small messages
  - 4x the concurrency needed to drive all of the NICs
- An Aurora node is different:
  - 2 CPUs
  - 8 NICs
  - 6 GPUs
- Managing locality and affinity is likely to require per-system thought
  - Not necessarily code tweaks
  - Definitely need to consider problem layout



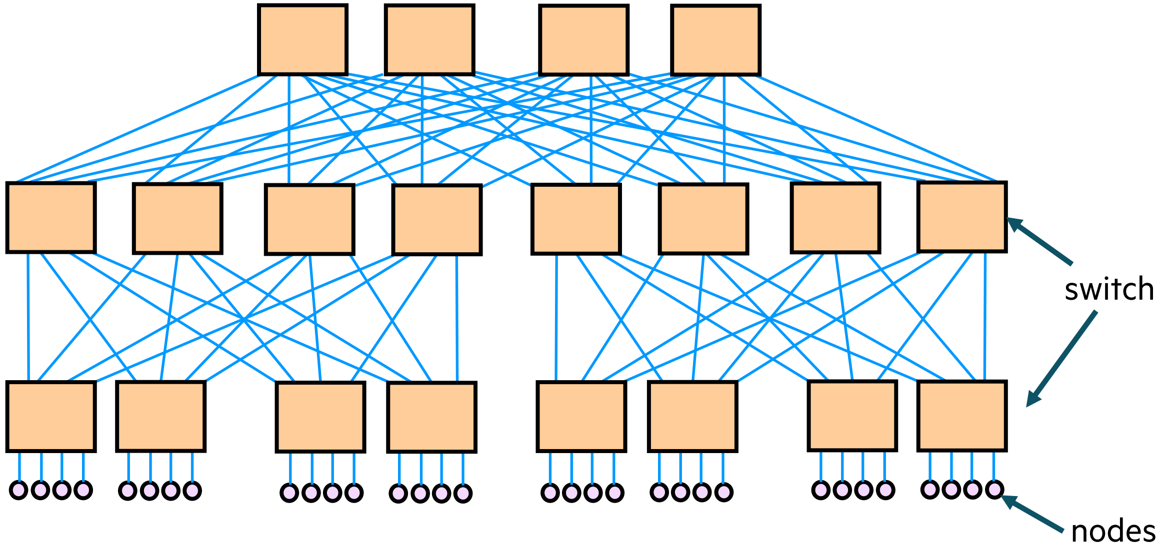
TOPOLOGIES EVOLVE WITH TECHNOLOGY AND APPLICATIONS

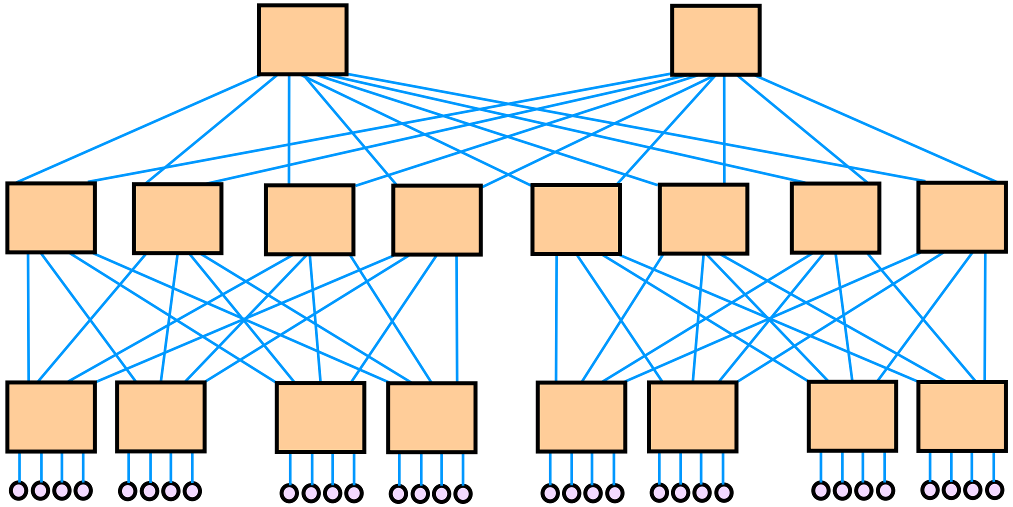
- In the old day: studies demonstrated a 3D torus was best – why?
  - Applications were regular / structured
  - Electrical wires were distance limited, but long enough
- Modern topologies optimize for bisection bandwidth – why?
  - Electrical wires cannot reach for multiple meters anymore
  - Many applications are less regular / more unstructured / more sparse
  - Many users / applications on a system, so harder to optimize placement

- Topology still matters
  - Be locality aware
  - Be mindful of your process layout
- Topologies will keep changing as technology changes
- Upcoming technology drivers:
  - Does electrical go away?
    - They told me it would at 10 Gbps…
    - It's still hanging around at 200 Gbps, but it doesn't go very far
  - Ring resonators will happen… someday… soon?
    - Lots of bandwidth in one fiber using lots of wavelengths

FAT-TREE TOPOLOGY

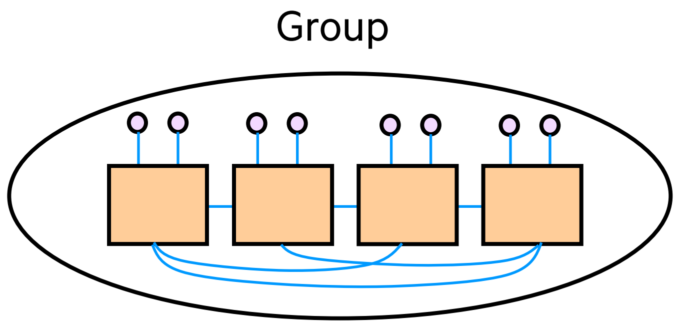
- Common topology for current systems (Sierra & Summit)
  - 3 Levels, maximum fabric hop count of 4
- Fully configured fat-tree (Summit)
  - Tree with equal bandwidth at each level
  - Non blocking between all pairs of nodes
    - Doesn't always happen in practice
  - Over provisioned bandwidth
    - Performance is quite good
    - Large percentage of fabric cables are optical
    - Can get expensive at scale
- Bandwidth tapered fat-tree
  - Unequal bandwidth at each level
  - No longer non-blocking between all pairs of nodes
  - Reduces number of cables (optical) and routers
- The number of cables and routers increases super-linearly with node count
- Scheduling a job across nodes to minimize fabric hops between hosts maximizes performance



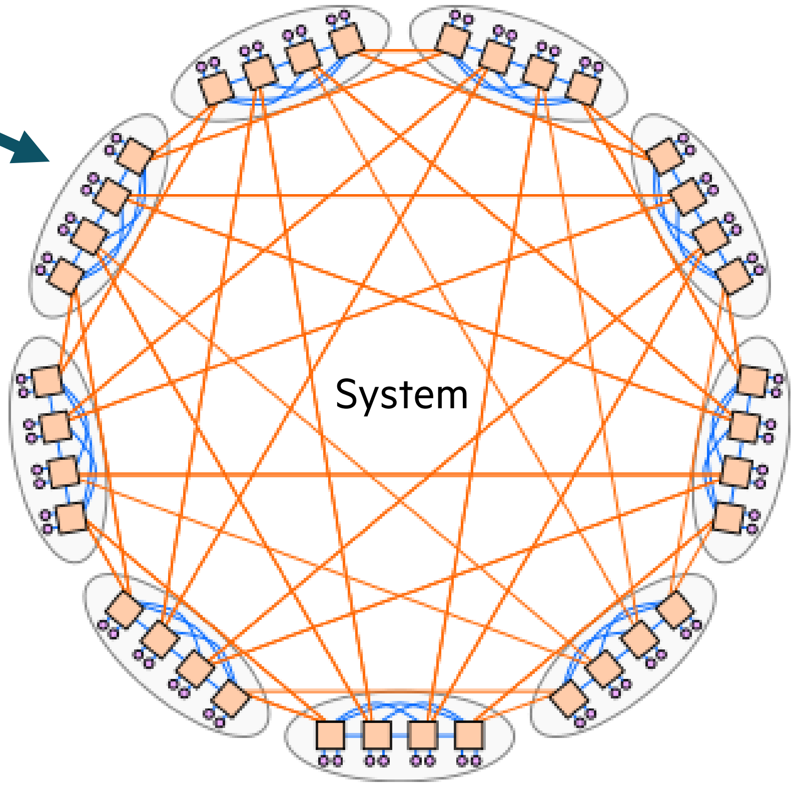


DRAGONFLY TOPOLOGY

- Every router has nodes connected
- A group contains routers that are all to all connected (1-D Dragonfly)
- All groups in the system are all to all connected



- Aurora, Frontier, El Capitan will use a 1-D Dragonfly
- Primarily driven by network cost as system scale grows
  - Linear increase in the number of cables and routers with system size
  - Less than 33% of fabric cables are optical
  - Scales to 4x number of nodes as 3 level fat-tree
  - Maximum hop count of 3
- Requires sophisticated adaptive routing
- Job scheduling
  - Intra-group if job can fit in a single group (256-512 nodes/group)
  - Randomly across system if job is larger than a single group
    – Bandwidth between individual group pairs is low compared to node injection bandwidth into group (and total bandwidth out of group)



INTERCONNECT FEATURES THAT IMPROVE PERFORMANCE*

- Adaptive Routing
  - Per packet - used to choose where a packet goes
  - Targets topological based congestion (unrelated flows crossing in the network)
  - Used to route around temporal hot spots in the network
    - Used sparingly, routing via a longer path can reduce latency
    - Used excessively, routing via a longer path can increase latency and decrease bandwidth
- Quality of Service classes
  - Part of arbitration - used to choose which packet to advance
  - Tunable classes may use priority, min & max bandwidth allocation, routing biases, etc
    - Example classes: low latency, standard compute, bulk data, scavenger
  - Job can use multiple classes
  - Provides performance isolation for different classes of traffic
- Congestion management
  - Targets workload-based congestion (incast, many to few)
  - Identifies and controls causes of congestion
    - Throttles sources to prevent excess traffic from entering the network
    - Prevents highly filled buffers, congestion, contention
  - Applications much less vulnerable to other traffic on the network
  - Predictable runtimes
  - Lower mean and tail latency – a big benefit in applications with global synchronization

* Slingshot techniques described

ASSESSING CONGESTION MANAGEMENT – GPCNET ON FRONTIER

```
NetworkLoad Tests v1.3
  Test with 72000 MPI ranks (9000 nodes)
  1800 nodes running Network Tests
  7200 nodes running Congestion Tests (min 1800 nodes per congestor)
```

Isolated Network Tests			
Name	Avg	99%	Units
RR Two-sided Lat (8 B)	2.6	4.6	usec
RR Two-sided BW+Sync (131072 B)	4246.8	2962.4	MiB/s/rank
Multiple Allreduce (8 B)	51.3	54.1	usec

Behavior on 9000 nodes matches that published for 512 nodes in the GPCNeT paper

Network Tests running with Congestion Tests			
Name	Avg	99%	Units
RR Two-sided Lat (8 B)	2.6	4.6	usec
RR Two-sided BW+Sync (131072 B)	4242.2	2961.2	MiB/s/rank
Multiple Allreduce (8 B)	51.4	54.0	usec

Network Tests running with Congestion Tests - Key Results		
Name	Congestion Impact Factor	
	Avg	99%
RR Two-sided Lat (8 B)	1.0X	1.0X
RR Two-sided BW+Sync (131072 B)	1.0X	1.0X
Multiple Allreduce (8 B)	1.0X	1.0X

WHAT YOUR HARDWARE NEEDS FROM YOU

DO: FACILITATE OVERLAP

- Older applications rarely exploit computation and communication overlap
  - This meant the computer was either computing or communicating
  - Network spent a lot of time idle
- Full overlap hides the communication behind the computation
  - Express the communications, then find work you can do that is not dependent on it
  - Use nonblocking sends and recvs

This code will need Wait calls somewhere

- Real world example: recent Frontier application
  - Was not using overlap
  - Major "idle gaps" in the network
  - Carefully scheduling communication and computation enabled significant performance improvements
- ***Warning: be careful about what you wait for***
  - Ideally, call wait when the transfer is already done
  - Waitall may not be your friend

```
MPI_Irecv(PeerResultsPhaseA);
do_work_PhaseA();
MPI_Irecv(PeerResultsPhaseB);
MPI_Isend(ResultsPhaseA);
do_work_PhaseB();
MPI_Irecv(PeerResultsPhaseC);
MPI_Isend(ResultsPhaseB);
do_work_PhaseC();
MPI_Isend(ResultsPhaseB);
```

DO: USE EXPECTED MESSAGES

- A message is “expected” if the receive is “posted” before the message arrives
  - Unexpected otherwise: Unexpected messages lead to data copies
- Please post your receives ***early***
  - May need to double buffer to make this work
  - Many applications have a communication buffer that data is copied out of
- Interleave communication and computation

- Real world example: recent Frontier application
  - Encountered unexpected messages
  - Increases time spent copying data
  - Decreases achievable overlap
  - Moving the receives earlier in the code eliminated unexpected messages, and improved performance

```
MPI_Irecv(PeerResultsPhaseA[1]);
MPI_Irecv(PeerResultsPhaseB[1]);
MPI_Irecv(PeerResultsPhaseC[1]);
do_work_PhaseA(PeerResultsPhaseA[0]);
MPI_Isend(ResultsPhaseA);
do_work_PhaseB(PeerResultsPhaseB[0]);
MPI_Isend(ResultsPhaseB);
do_work_PhaseC(PeerResultsPhaseC[0]);
MPI_Isend(ResultsPhaseB);
```

DO: EXPRESS CONCURRENCY

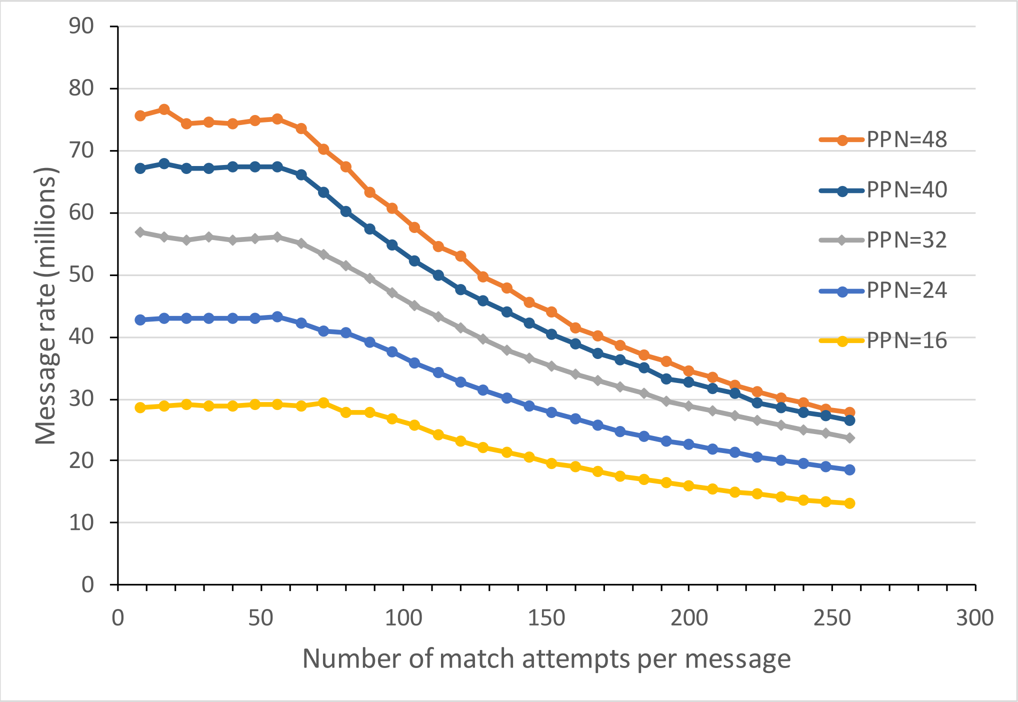
- You have 4 NICs... or 8... use them!
  - One message is not going to be split across multiple NICs
    - Locality matters a lot, so it would not be beneficial
  - Need to make sure there is enough concurrent messages to drive all of the NICs
- Potentially hard choices to make
  - If the messages are "too small", you will need more cores to drive them
  - Assuming 3 Mmsgs/s/core, need 8KB messages for one core to drive one direction of one NIC
  - Same computation yields 4 cores for 4 NIC
  - 64 cores may be able to drive 4 NICs with 512B messages – in one direction
- How many ranks will you have per node?
- Alternate approach: tell the implementation more
- Example: partitioned communications
  - Facilitates exposing even more concurrency to the NIC
  - Per-"message" overheads are amortized by pre-setup operations
  - Implementation can use various strategies to match the capabilities of the hardware
- Caution: Newest MPI features tend to have a chicken and egg problem
  - Work with your vendor to focus on optimizing the right pieces

DON'T: OVERWHELM YOUR NIC RESOURCES

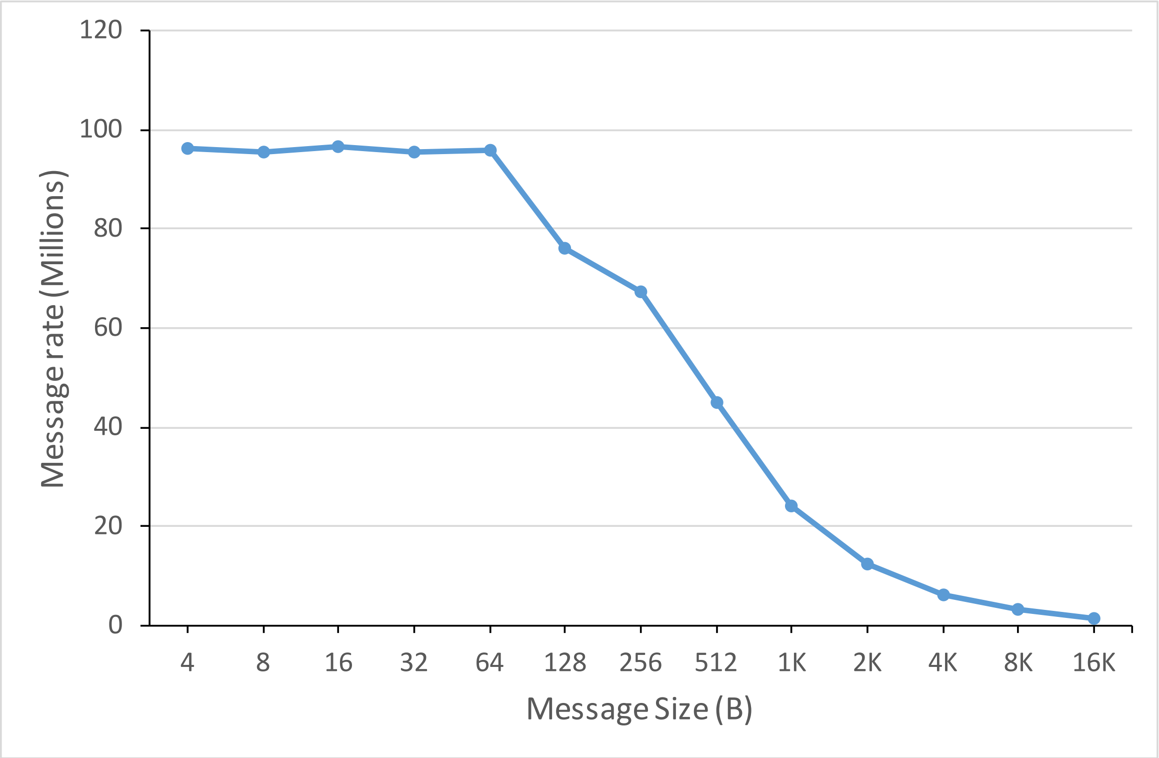
- Most NICs try to offload MPI matching now
  - MPI Matching typically uses a "posted receive queue" of Irecv operations the user has issued
  - Typically uses a linked list, but sometimes a hash table can be used
  - Unexpected messages form their own list
- Long lists have hidden costs
  - New message searches posted receive queue
  - New Irecv searches unexpected messages
- Most NICs have a limited number of places to hold entries
  - That's ok: traversing enormous lists is a huge waste of time.
  - Falling back to software erodes your:
    - Overlap: need software to make progress
    - Network performance: software matching isn't as fast as benchmarks tell you

CASSINI REAL-WORLD MESSAGE MATCHING

- Target: two searches with an average of 64 match attempts for every message at peak rate
  - Small number of unexpected messages
  - Posted receives for each neighbour (maximum of 16K)
- MPI message matching rate
  - 100 million messages per second
  - ~6.4 billion match attempts/sec



Message rate for increasing number of match attempts

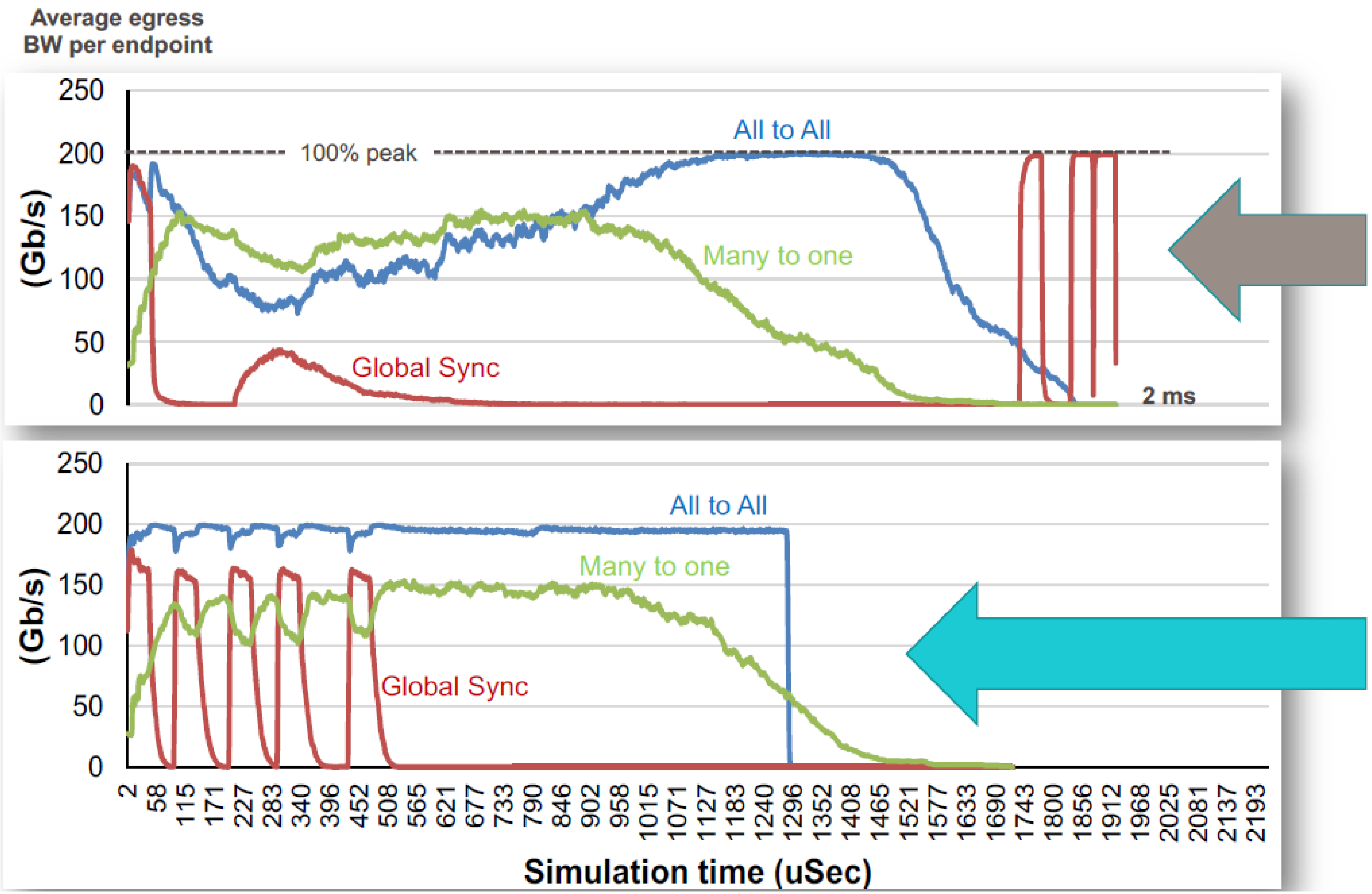


Message rate for increasing size, 64 processes per node

DON'T: PROGRAM AN INCAST

- Congestion management is designed to reduce collateral damage
  - Stops your app from hurting other people's apps
  - It is really good, but it is not magic
- An app performing an incast is still bound by the laws of physics
  - Messages cannot complete faster than the target can absorb them
  - Most networks (and network operators) would be willing to punish an app with an incast to protect others
- Unfortunately, people routinely do this
  - Example: every rank checks in with the root
    - Rank 0 winds up posting 100,000 to 1,000,000 receives
    - What order do those arrive in?
- Remember: MPI (typically) uses linked lists
  - If you post 1,000,000 receives and hit the end, that goes badly
- Side note: MPI_ANY_TAG and MPI_ANY_SOURCE
  - Use them ***always*** or ***never***
  - Mixing the two makes a mess of things (i.e., it gets hard to make a hash table)

CONGESTION MANAGEMENT PROVIDES PERFORMANCE ISOLATION

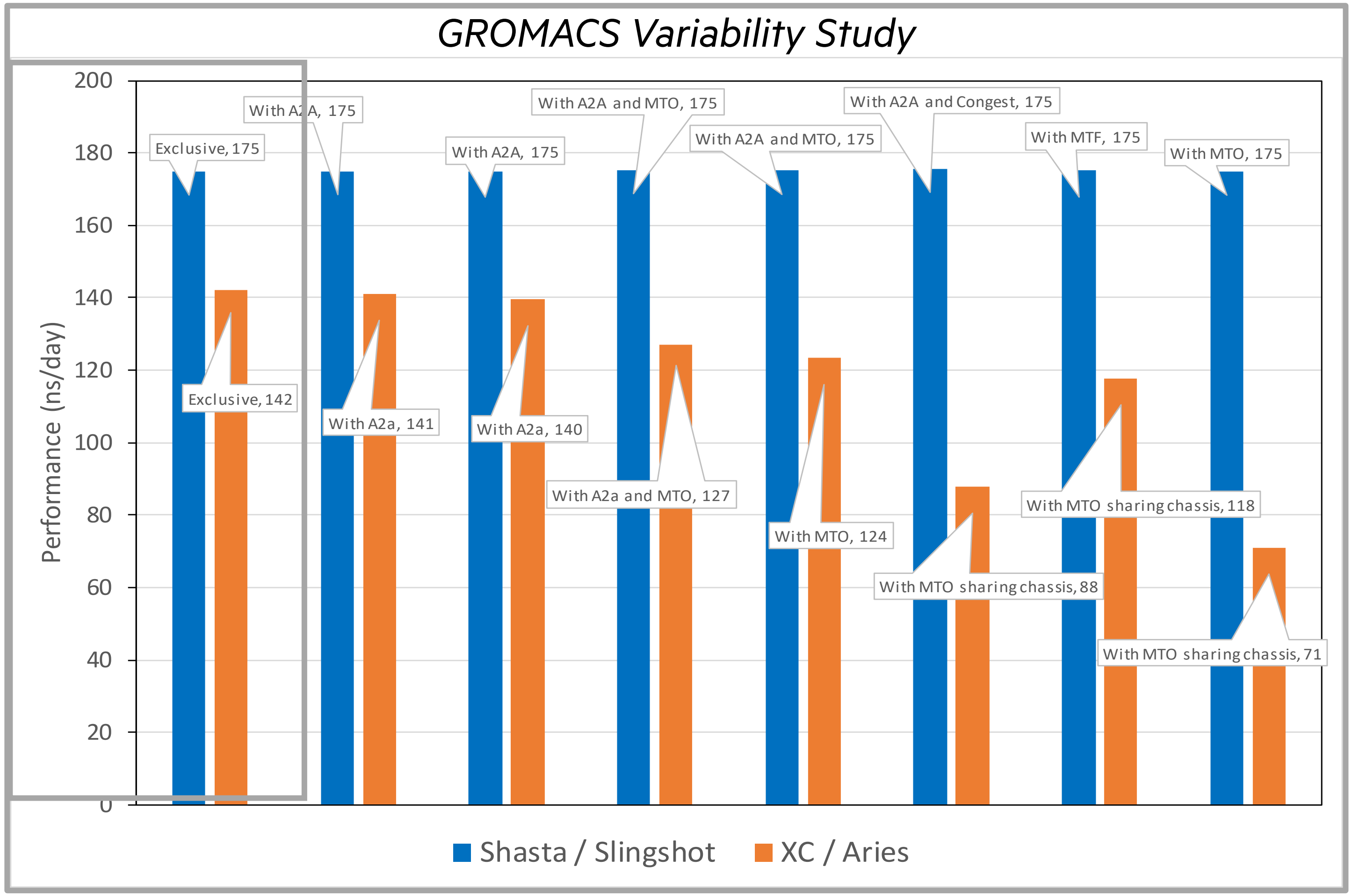


Job Interference in today's networks

Congesting (green) traffic hurts well-behaved (blue) traffic, and really hurts latency sensitive, synchronized (red) traffic.

With Slingshot Congestion Management

MINIMIZE OR ELIMINATE RUN-TO-RUN VARIABILITY



"At the same time that HPC centers are getting increasingly in need of congestion control is precisely the moment when Cray-now-HPE has a new switch that is doing congestion control in a new fashion…

The congestion control features in HPE Slingshot seem to be working like a charm."

Timothy Prickett Morgan – The Next Platform

Slingshot controls tail latency under load

DON'T: USE NON-SCALABLE CONSTRUCTS

- Historical application example 1: MPI_Gather()
  - Monte Carlo application computed a lot of independent results
  - Gathered them to the root and summed the results
  - Scaled poorly because the total amount of data gathered at the root grew linearly with the number of nodes
    - Root node was receiving gigabytes
    - Then it had to sum those results
- Right answer (for that application): MPI_Reduce()
  - Summation happened in parallel
  - Logarithmic scaling of reduction of data
- Your mileage may vary
  - Maybe there is not a built-in operator for what you need to do
  - Do the work to think about the scaling implications
- Another painful example: Alltoallv

DO: USE FASTER, MORE SCALABLE CONSTRUCTS

MPI-3 RMA and OpenSHMEM

- Lower overhead
  - No tag matching
  - No unexpected messages
- More scalable operations
  - No lists or linked list traversal
  - Directly access peer memory
- Also, not a panacea
  - Two-sided operations have inherent management of buffer access
  - Synchronization adds overhead
- MPI-3 also has a non-scalable resource: the window
  - Limited number of “fastest” windows
  - Implementation can be forced to track a surprising amount of state per peer

Partitioned Communications

- Leverages persistent communication infrastructure
  - A lot of the “expensive parts” can be setup when the communication is created
  - Numerous optimizations available to implementations
- Enables underlying implementation to leverage operations similar to RMA operations
  - Sometimes lower overhead
  - Sometimes higher message rates
- Originally intended to facilitate threading
  - May later be extensible to GPU operations

COMMENTARY ON OTHER NETWORK TRENDS – AND FADS

- Optical interconnects are a critical enabling technology
  - For the near term, it is just wires using light
  - Glass is lower loss for photons than copper is for EM waves
- Longer term: will we see optical switching?
  - Interesting, exciting, and harder than it sounds
  - What will it take beyond technology development?
    - Can you use a network where bandwidth is semi-statically partitioned between long term “connections”?
- Machine Learning “super pods”
  - Interesting for problems that fit in a super pod
    - Includes (at least) 3 tiers of locality inside – does anybody want to add yet-another-tier of locality?
- CXL: the future of NIC to host interconnects
  - Until PCIe subsumes the capabilities…
  - In the near term, hard to use for a NIC
    - Even harder to use portably
- SmartNICs: everything old is new again
  - HPC has used programmable NICs before (e.g., Quadrics, Myricom)
  - Two differences this time:
    - The processors are less connected to the datapath
    - There are commercial customers
  - What AWS, Google, and Microsoft do with a SmartNIC is not typical for HPC

SUMMARY

- Low ratio of bandwidth to compute means we have to be smart about using the bandwidth we have
  - Network hardware is adding features to help
    - Enabling overlap
    - Advanced adaptive routing
    - Advanced congestion control
- We still need help from the programmers
  - Operations must be organized in order to enable overlap with expected messages
  - Hardware will continue to deliver features, but many will depend on developers to use them
- Not every interface available in software can be optimized
  - Choose wisely!
  - Benchmark what you do and engage with your network vendor

THANK YOU

Keith D. Underwood
keith.underwood@hpe.com