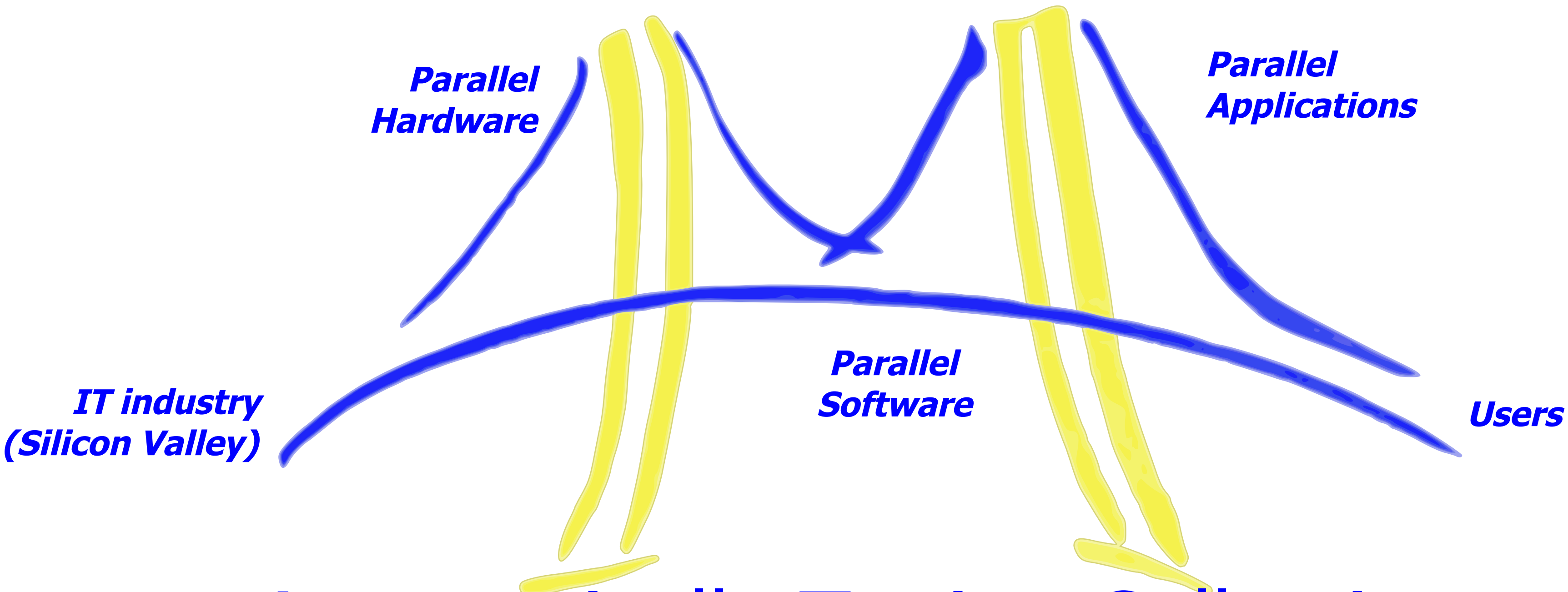


Automatically Tuning Collective Communication for One-Sided Programming Models

Rajesh Nishtala

Ph.D. Dissertation Talk

Committee: Katherine Yelick (chair), James Demmel, Panos Papadopoulos

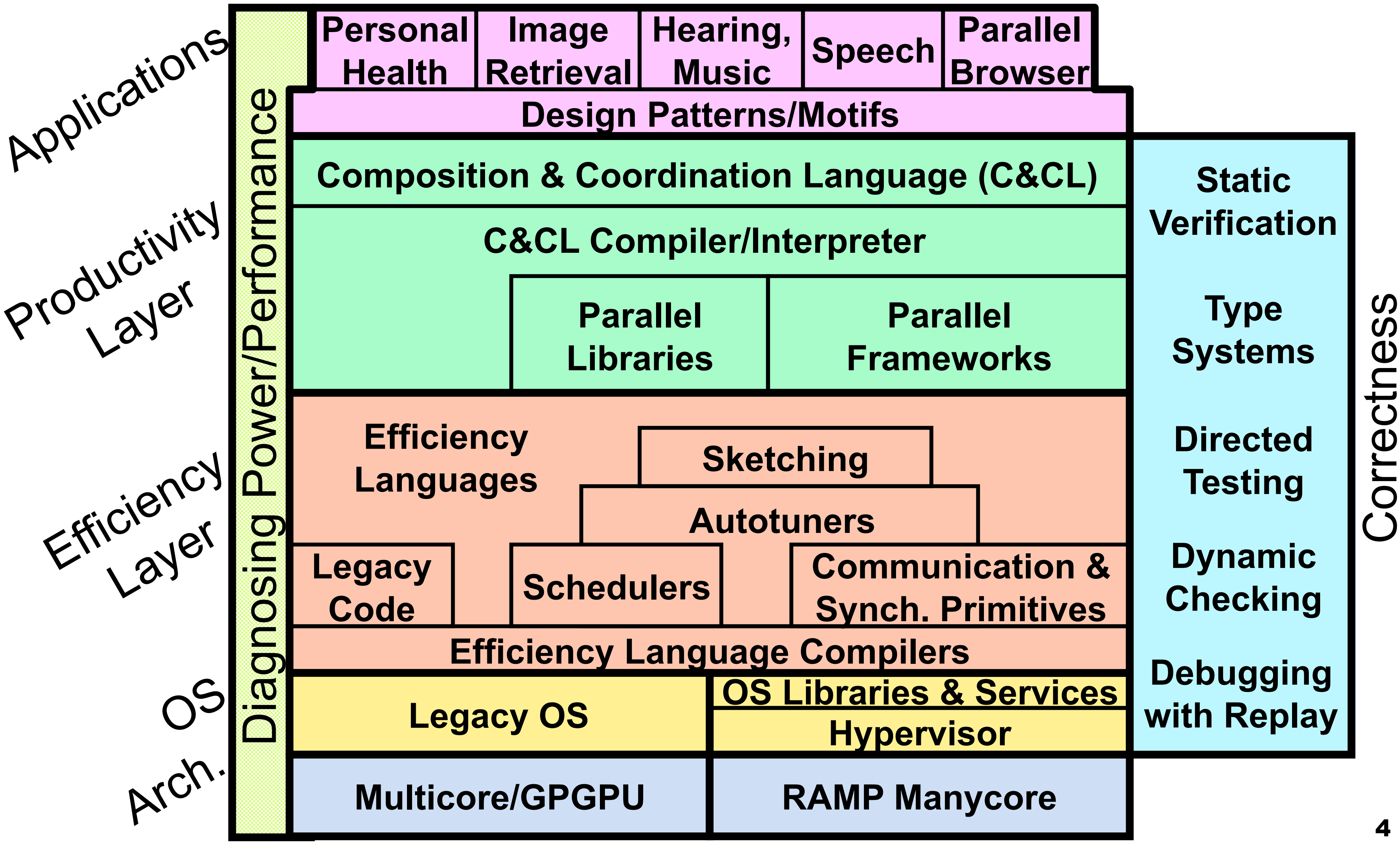
Observations

- Scientists and engineers are able to leverage large-scale systems to solve many problems important for society
  - e.g. climate simulations, genomics, cloud services, etc.
- Many interesting problems will still require orders of magnitude more computational power
- With current technological limitations (*i.e. power*) the only way to deliver the performance is by using lots of processors and relying on parallelism
  - Responsibility of efficiently using the system shifts away from the hardware and higher into the software stack

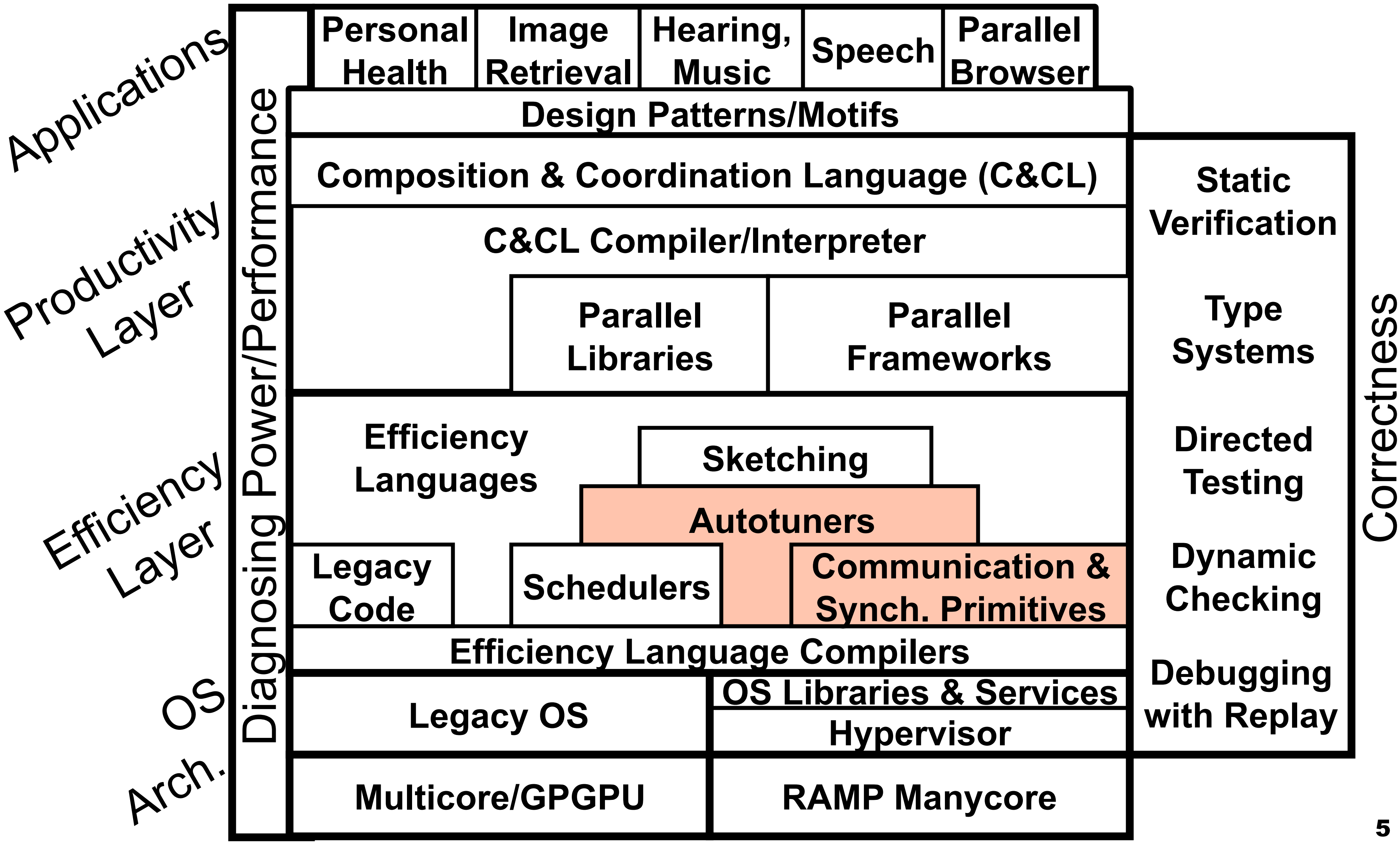
Current Processor Counts

- Large Scale Systems
  - Very common to have more than 1024 processor cores
  - Largest machines have over 128,000 processor cores
  - Millions of cores in the not-so distant future
- Desktop/Laptop/Cell Phones
  - Multicore processors are ubiquitous
  - Tens to hundreds of processors per system within the not-so distant future
    - Intel just announced 48-core processor
    - GPUs already support programming models with high levels of parallelism
- Communication is the key!
  - Must design programming models to allow processors to efficiently communicate with each other

Par Lab Research Overview



Par Lab Research Overview



Contributions

- Automatically tuned collective communication library for PGAS languages
  - Collectives are common communication building blocks used by many applications
  - Understand how the one-sided communication model affects the collective tuning
  - Tuning for both shared and distributed memory systems
- Allow collectives to be overlapped with computation
- Developed performance models to better understand the performance tradeoffs
- Incorporate collectives into application benchmarks
  - Some of the largest scale runs of PGAS languages
- Software is integrated into latest release of Berkeley UPC

EXAMPLES OF MODERN SYSTEMS

Levels of Parallelism

- Many levels of parallelism
  - Each has its own implications for the communication
  - How do we manage communication at the different levels
  - Example: IBM BlueGene/P

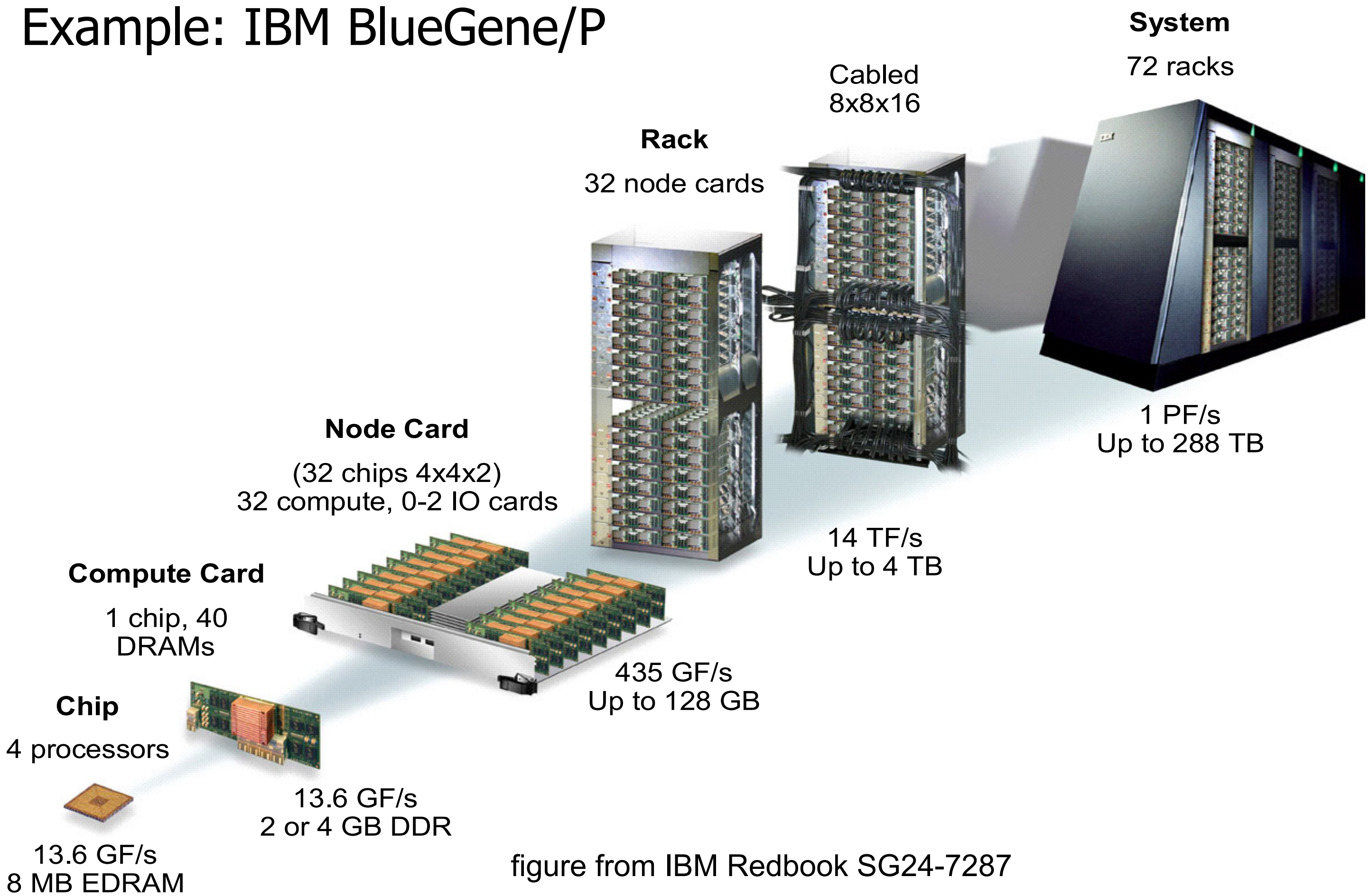
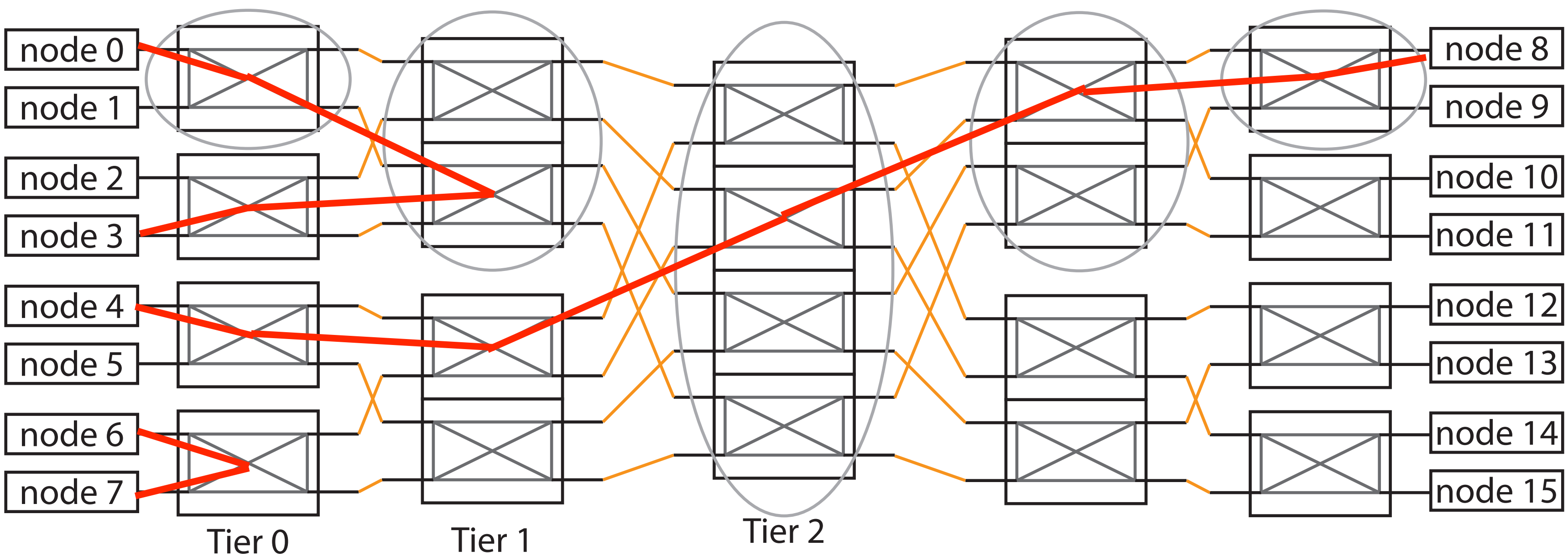


figure from IBM Redbook SG24-7287

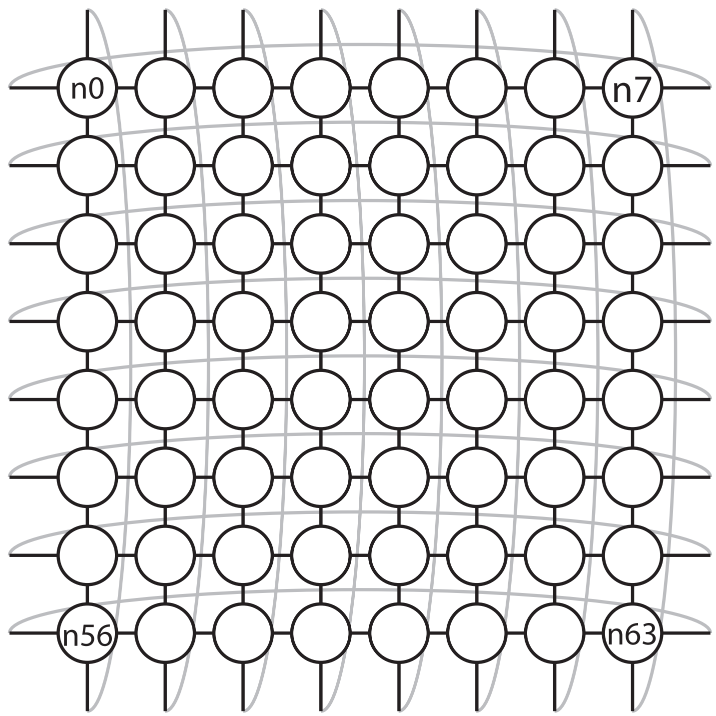
3-level Fat Tree

- Connect nodes such that there is a constant bandwidth between all nodes
  - First described by Charles Clos in 1952 for the telephone network
  - Connectivity is very similar to the butterfly found in the Fast Fourier Transform (FFT)
- Also called a "Fat Tree"
  - Switches placed into groups at every level
  - Bandwidth between child and parent groups doubles every step
  - P-port switch with T levels requires $(2T-1)(P/2)^{(T-1)}$ switches



Mesh/Torus Networks

- Fat Tree networks can be quite expensive
  - A high number of switches might be overkill
    - Tradeoff number of switches for bandwidth across network
  - A lot of applications don't need full bandwidth to *every* other node
  - Depends on target network performance and application
- In a mesh network nodes are directly connected to their neighbors
  - Unlike switched network, the network cards at the nodes need to be able to route messages
  - Messages routed through the grid
  - Bandwidth on the links is shared
  - Torus is mesh with ends wrapped
  - Example is 8x8 Torus
- What is the target network performance?
- What are the target applications?

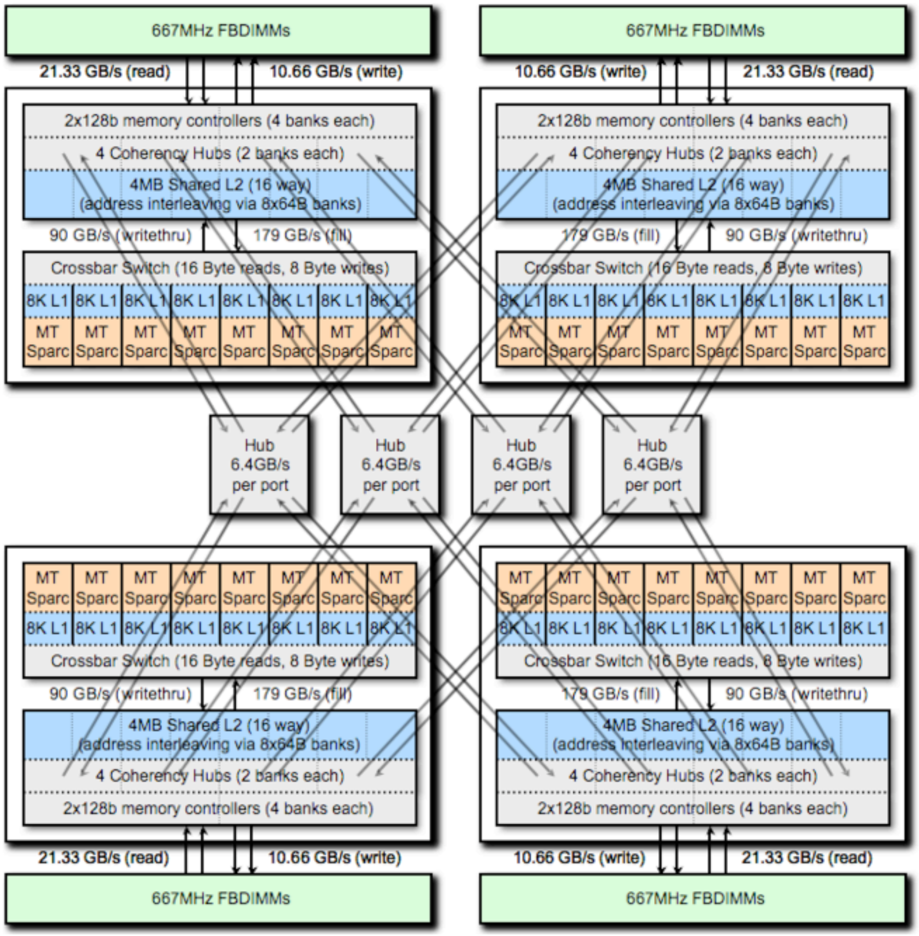


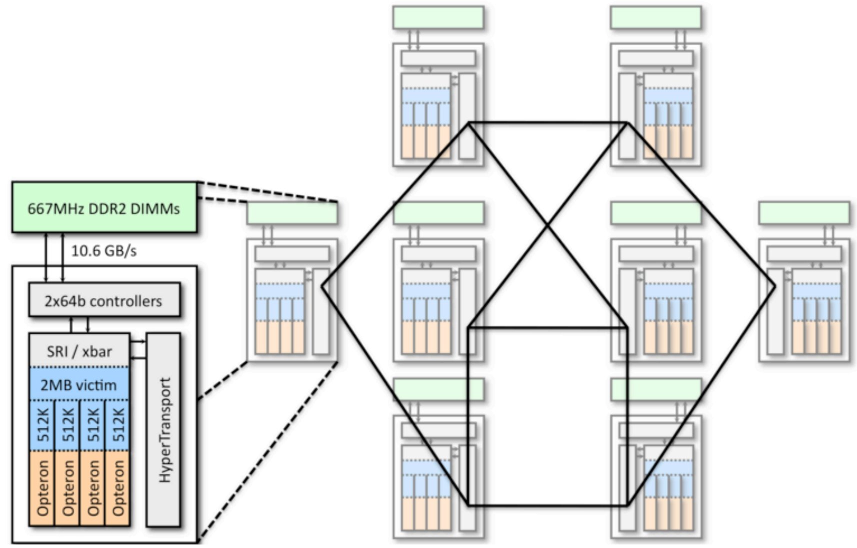
Summary Of Experimental Platforms

	Cray XT5	IBM BlueGene/P	Sun Constellation	Cray XT4
Name/Location	Jaguar/ORNL	Intrepid/ALCF	Ranger/TACC	Franklin/NERSC
Top500 Rank (Nov. 2009)	1	8	9	15
Processor Type (Revision)	AMD Opteron (Istanbul)	IBM PowerPC 450	AMD Opteron (Barcelona)	AMD Opteron (Budapest)
Processor Speed	2.6 GHz	0.85 GHz	2.3 GHz	2.3 GHz
Cores/Node	12	4	16	4
Total Cores	224,256	163,840	62,976	38,288
Interconnect	3D Torus	3D Torus	4-level Fat Tree	3D Torus

Shared Memory Systems

Sun Niagara2 (256 threads)





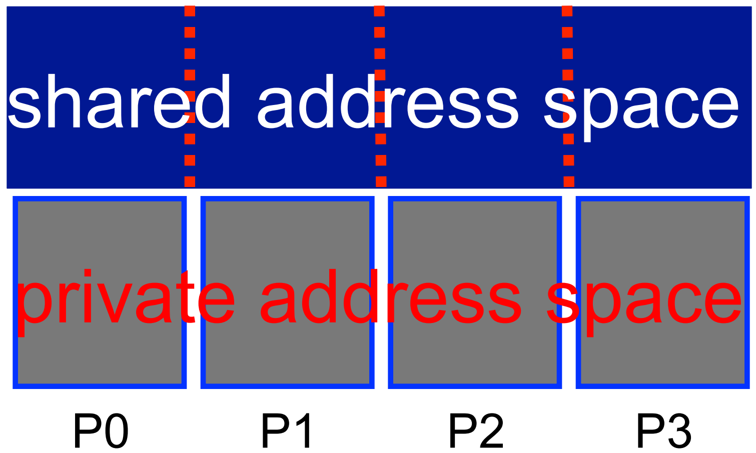
AMD Opteron (32 threads)

[Diagrams Courtesy of Sam W. Williams]

ONE-SIDED PROGRAMMING MODELS

Partitioned Global Address Space (PGAS) Languages

- Programming model suitable for both shared and distributed memory systems
- Language presents a logically shared memory
- Any thread may directly read/write data located on a remote processor
  - Can build complex distributed data structures
- Address space is partitioned so each processor has affinity to a memory region
  - Accesses to “local” memory are potentially much faster

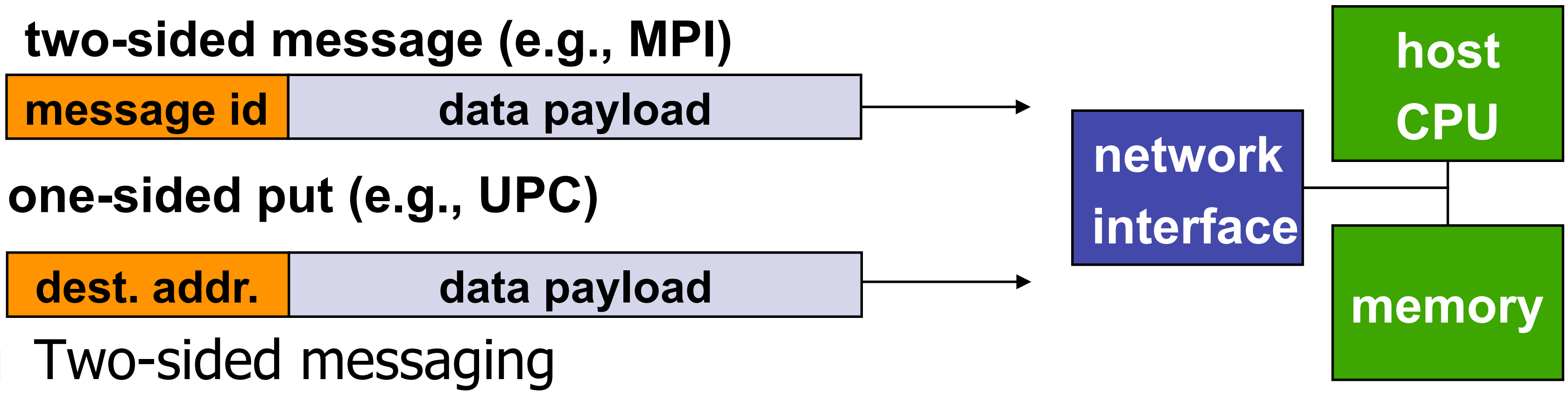


Many PGAS Languages:
UPC, Titanium, Co-Array Fortran, X10, Chapel, etc

UPC Overview

- A PGAS dialect of ISO C99
- Both private and shared data
  - **int x[10];** and **shared int y[10];**
- Support for distributed data structures
  - Distributed arrays; private and shared pointers
- One-sided shared-memory communication
  - Simple assignment statements: **x[i] = y[i];** or **t = *p;**
  - Bulk transfer operations: memcpy
- Synchronization
  - Global barriers, locks, memory fences
- Collective Communication Library
  - Broadcast, Gather, Gather-all, Scatter, Exchange, Reduce, Scan
- I/O libraries
- Implemented by multiple vendors and free-software efforts
  - Language is under active development

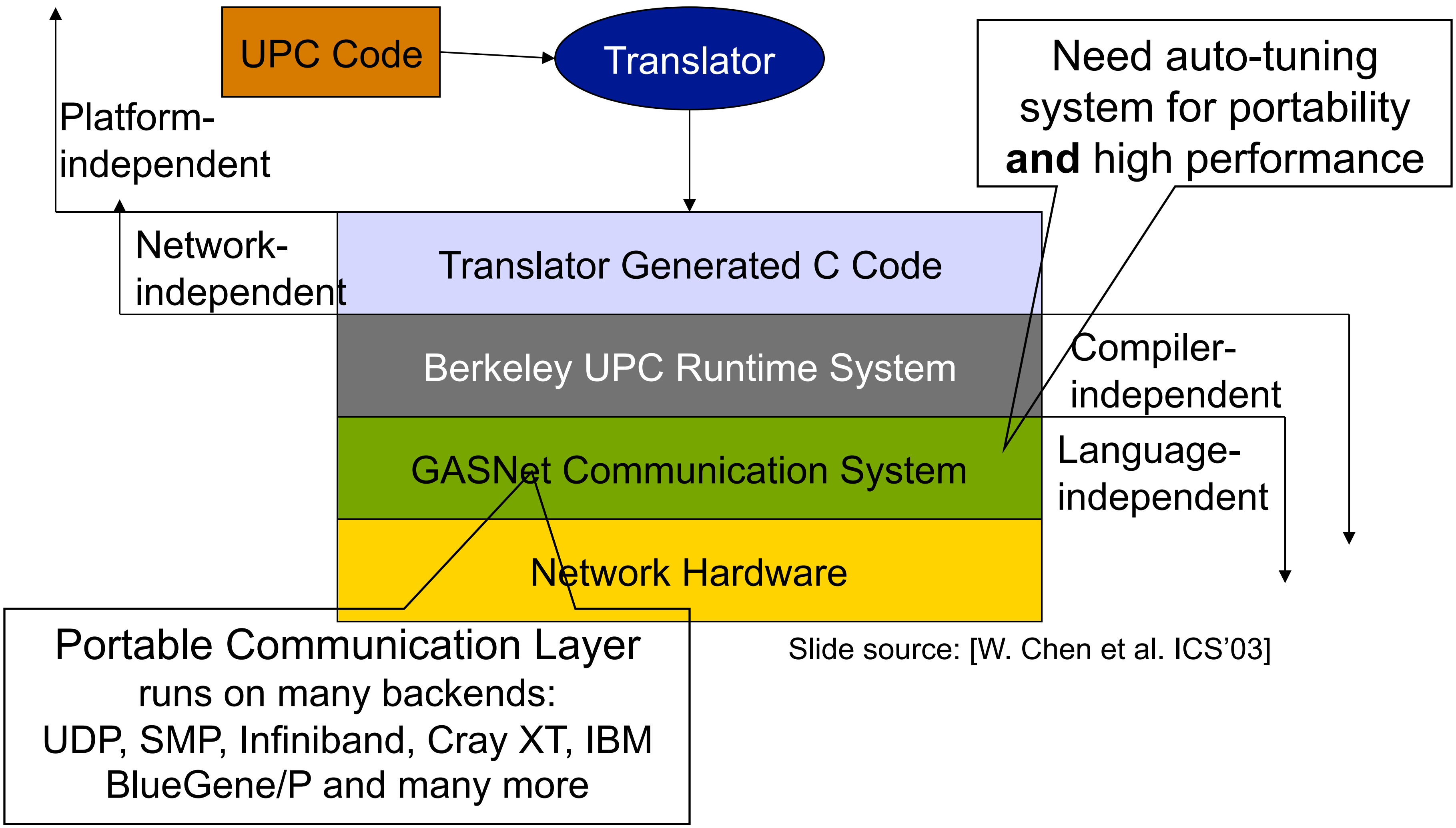
One-Sided vs. Two-Sided Messaging



- Two-sided messaging
  - Message does not contain information about final destination
  - Have to perform look up at the target or do a rendezvous
  - Point-to-point synchronization is implied with all transfers
- One-sided messaging
  - Message contains information about final destination
  - Decouple synchronization from data movement
- What does the network hardware support?
- What about when we need point-to-point sync?
  - Active Message based semaphore library to handle this efficiently (still one-sided!)

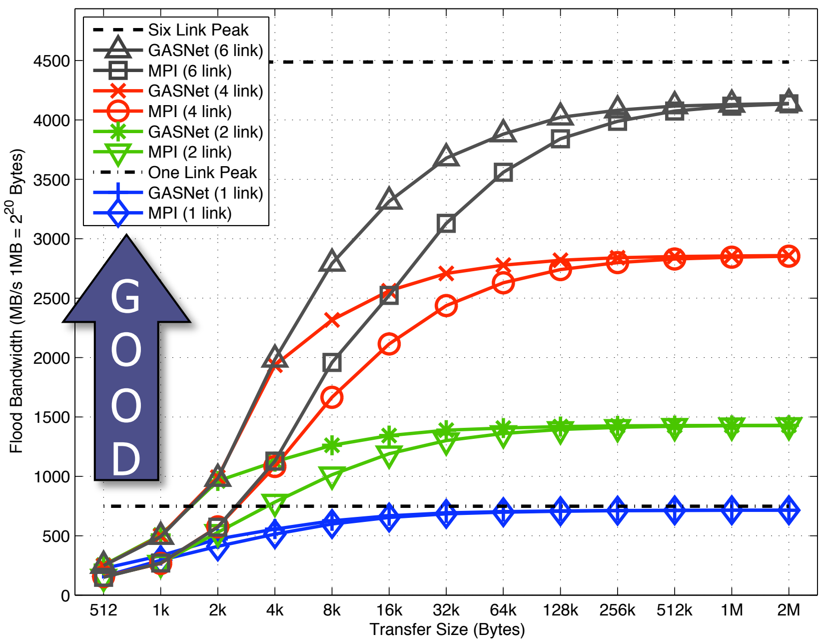
The Berkeley UPC Compiler

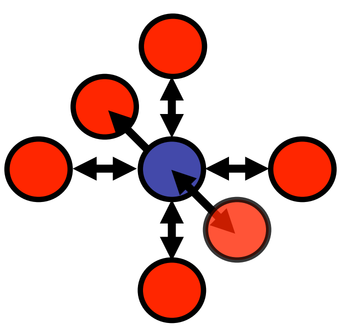
Two Goals: Portability and High-Performance



Slide source: [W. Chen et al. ICS'03]

GASNet Multilink Bandwidth



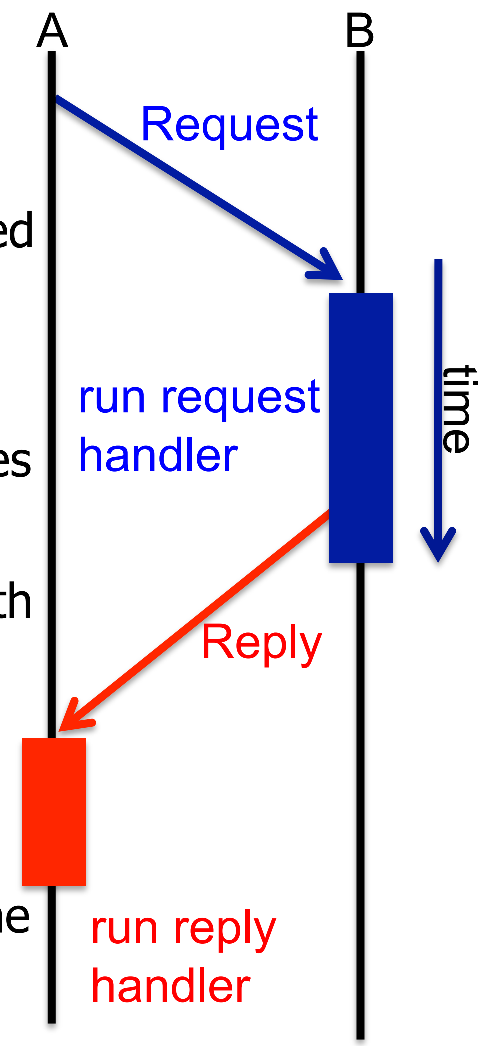


* Kumar et. al showed the maximum achievable bandwidth for DCMF transfers is 748 MB/s per link so we use this as our peak bandwidth
See "The deep computing messaging framework: generalized scalable message passing on the blue gene/P supercomputer", Kumar et al. ICS08

- Each node has six 850MB/s* bidirectional link
- Vary number of links from 1 to 6
- Initiate a series of nonblocking puts on the links (round-robin)
  - Communication/communication overlap
- Both MPI and GASNet asymptote to the same bandwidth
- GASNet outperforms MPI at midrange message sizes
  - Lower software overhead implies more efficient message injection
  - GASNet avoids rendezvous to leverage RDMA

GASNet Active Messages

- GASNet also offers rich Active Message library
  - Ability to invoke function on Remote Node
  - Important piece for collective implementation
- A request consists of an index into a function table to be invoked on the target side, arguments, and possibly payload
  - Short Request: no payload (just arguments)
  - Medium Request: small payload and arguments, source does not specify destination buffer
  - Long Request: payload and arguments, source provides both source and destination address of payload
- Replies run inside the request handler invocation
  - Can only send to the peer that sent the request
  - Have Short, Medium, and Long replies which have the same properties as their corresponding requests
  - Sending replies is optional



COLLECTIVE COMMUNICATION

What are Collectives?

- Operations that perform globally coordinated communication
- Most modern parallel programming libraries and languages have versions of these operations
- Encapsulate operations behind a library interface so that they can be tuned by runtime layer to achieve best performance and scalability

One-to-Many

- All processors communicate with a single root
  - Flat algorithm: O(T) messages
- Broadcast
- Scatter
- Gather
- Reduce-to-One

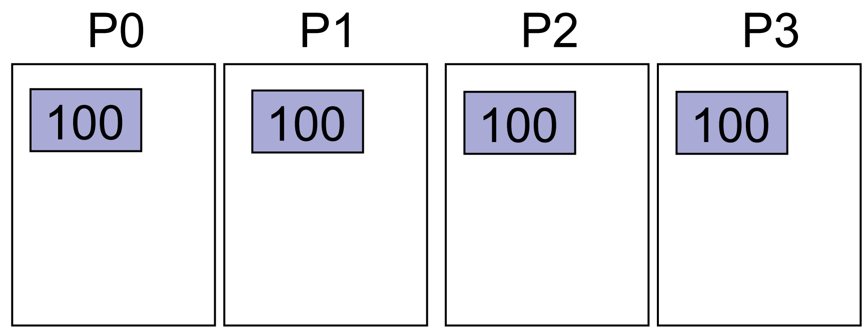
Many-to-Many

- All processors communicate with all others
  - Flat algorithm: $O(T^2)$ messages
- Barrier
- Gather-to-All
- Exchange (i.e. Transpose)
- Reduce-to-All

Rooted Collectives

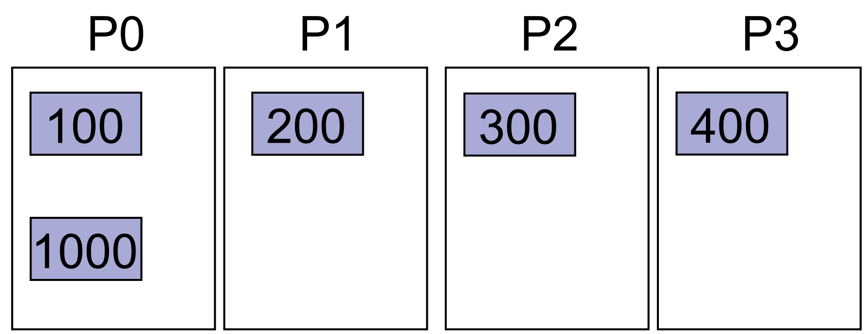
Broadcast:

send a copy of the data from root processor to all others



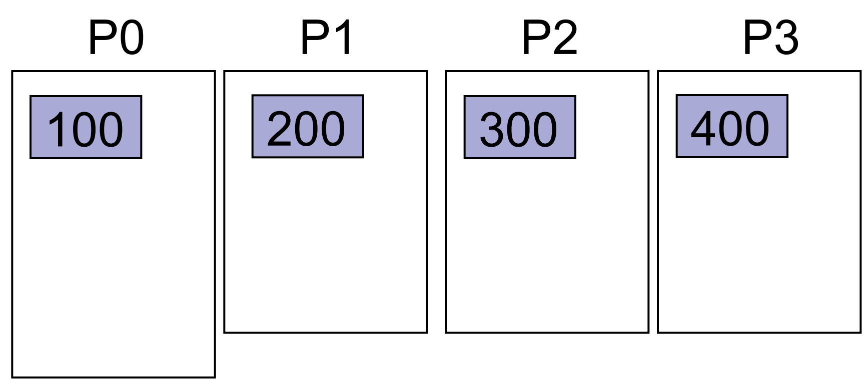
Reduce-to-One:

aggregate results from all processors



Gather:

All processors send a contribution to the root



Scatter:

inverse of Gather

Non-Rooted Collectives

Exchange (Transpose):

All processors simultaneously scatter input array

(personalized messages)

P0	A0	A1	A2	A3
P1	B0	B1	B2	B3
P2	C0	C1	C2	C3
P3	D0	D1	D2	D3

P0	A0			
P1	B0	B0	C0	D0
P2	C0	B0	C0	D0
P3	D0	B0	C0	D0

Gather-To-All:

All processors simultaneously broadcast input

(non-personalized messages)

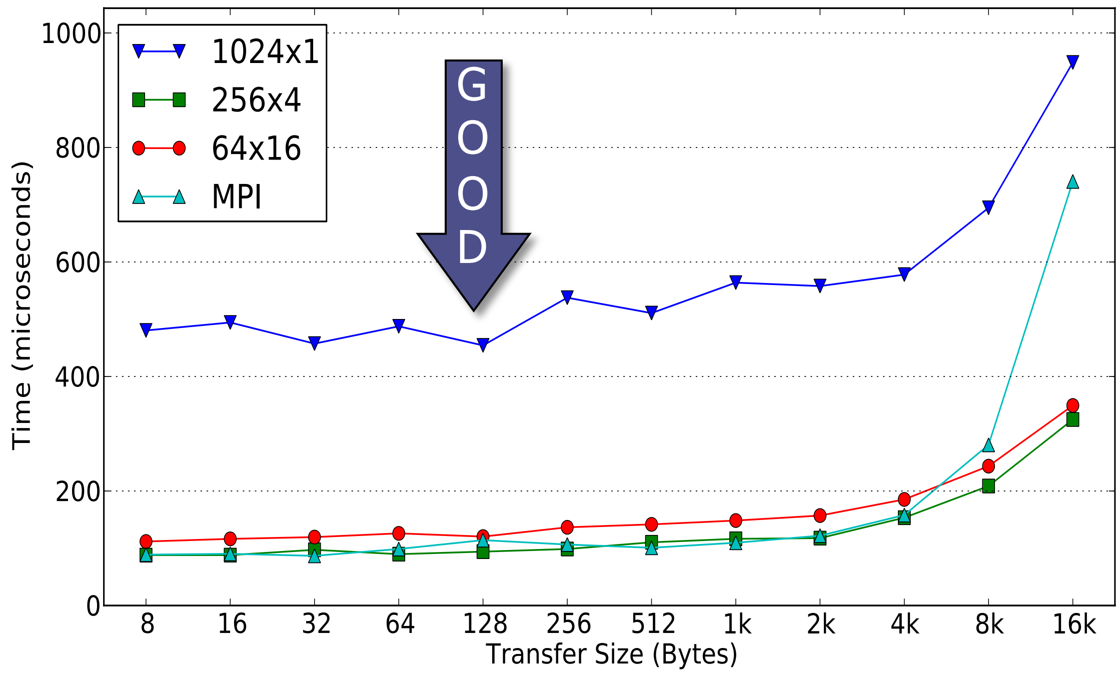
Design Goals for GASNet Collectives

- **Interface**
  - General collective interface that supports multiple PGAS languages
    - E.g. UPC and Chapel have different threading and execution models that we need to support
    - Have to support the many synchronization modes of UPC
  - Allow the collectives to be nonblocking
  - Support subset collectives (i.e. Teams)
- **Implementation**
  - Leverage shared memory whenever it's available
  - Effectively deliver the performance advantages of one-sided communication in the collectives
  - Automatically tune the collectives
    - Infrastructure should be able to include hardware collectives on platforms where applicable

TUNING COLLECTIVE COMMUNICATION FOR DISTRIBUTED MEMORY

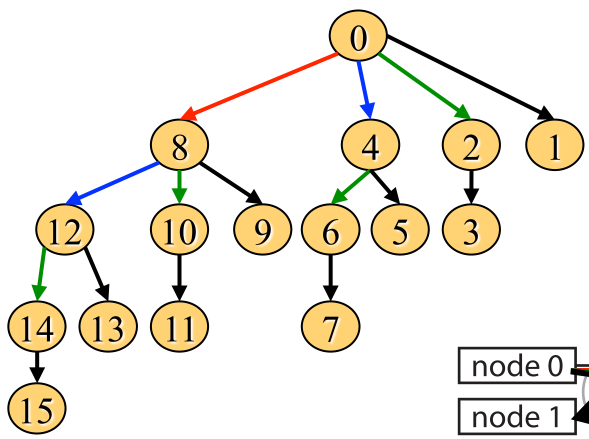
Leverage Shared Memory

- All cores within a node are part of the same shared memory domain
  - One-to-one mapping between threads and hardware cores
  - All threads within same OS process are part of same shared memory domain
- Have only one representative thread per node manages the communication
  - Responsible for packing/unpacking the data
- Experiment varies number of processes/thread grouping
  - Measures Broadcast latency of increasing sizes
  - 1024 cores of Sun Constellation (4 sockets / 4 threads per socket)
- Best performance is 4 threads per process
- Communication outside socket is expensive
  - Can incur the penalties for Non-Uniform Memory Access (NUMA)

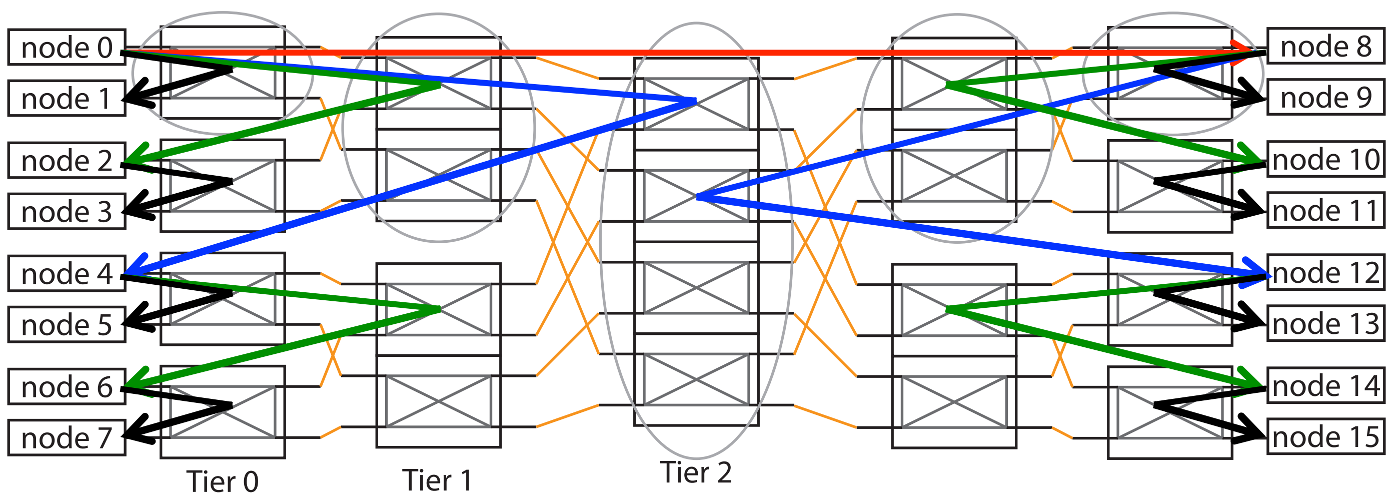


Trees

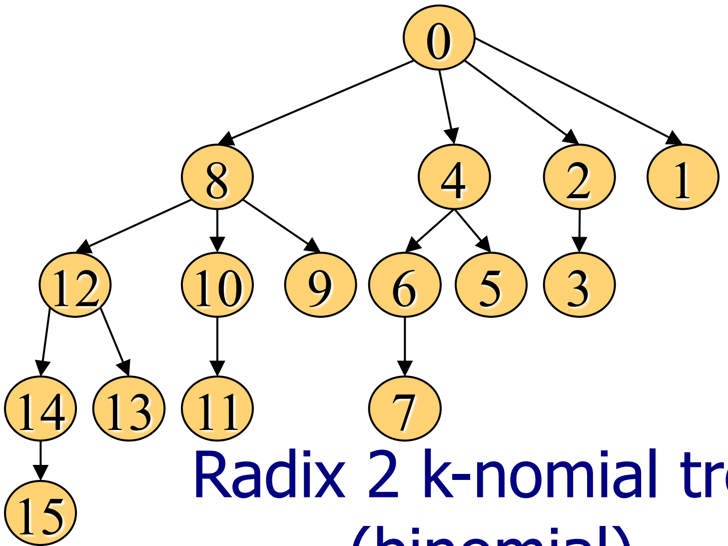
- Observation: All nodes are not directly connected together
  - Send the data through intermediaries to improve scalability
  - Nodes can communicate with O(log N) peers instead of O(n) peers
  - Tradeoff depth for the width



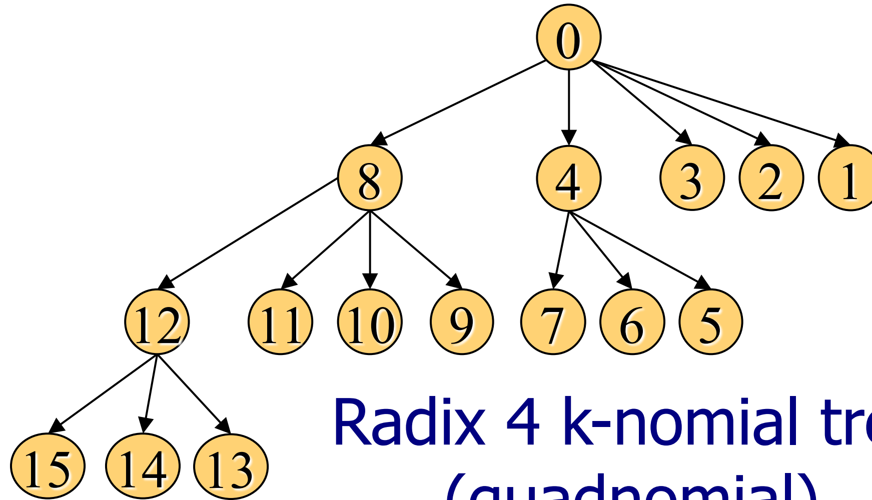
- Example: 2-nomial (Binomial) tree
  - Recursive Tree
    - Root sends to sub-trees of decreasing sizes
    - The higher the radix the shallower the tree



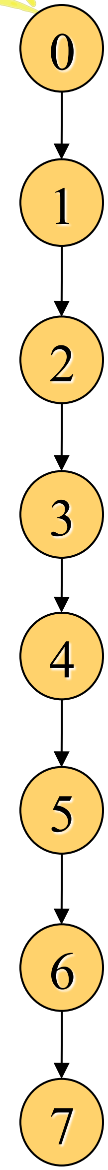
Example Tree Topologies



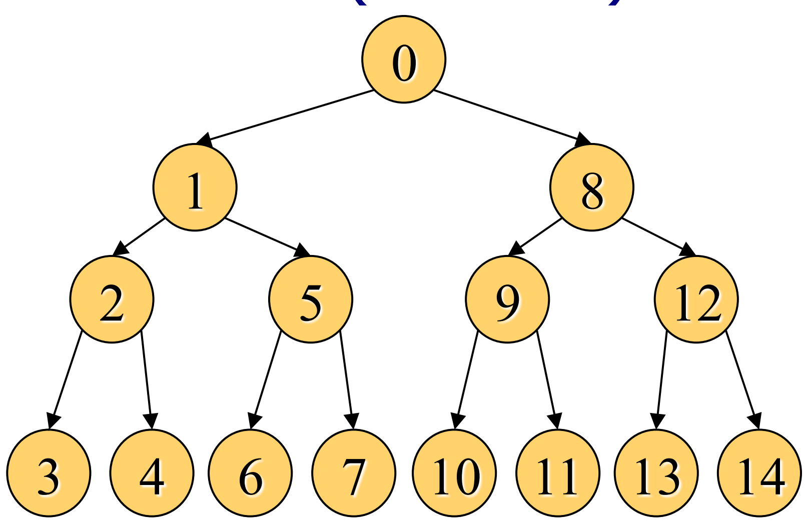
Radix 2 k-nomial tree (binomial)



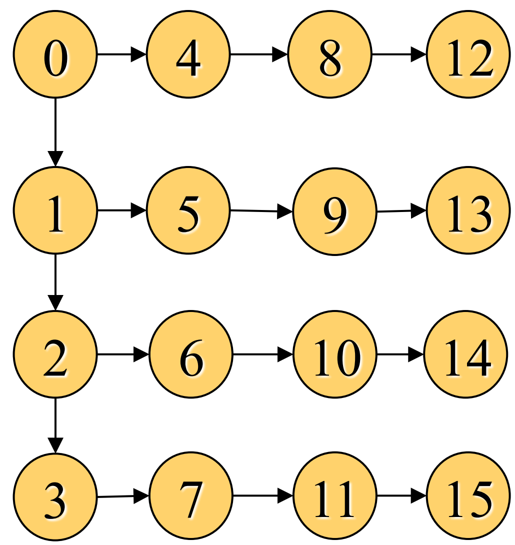
Radix 4 k-nomial tree (quadnomial)



Chain Tree



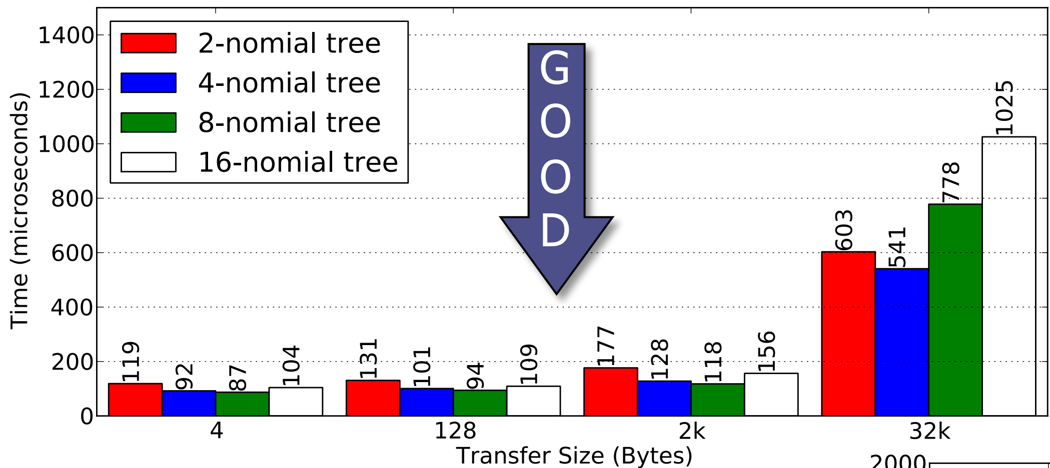
Binary Tree



Fork Tree

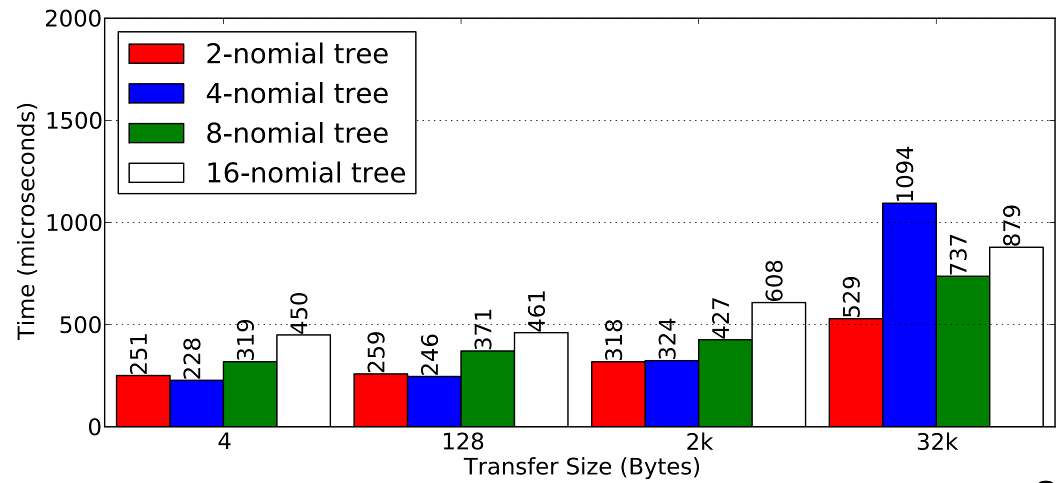
Choosing the Best Tree

- Optimal tree depends on many factors such as network latency and bandwidth and network connectivity
  - Best tree changes based on platform and collective



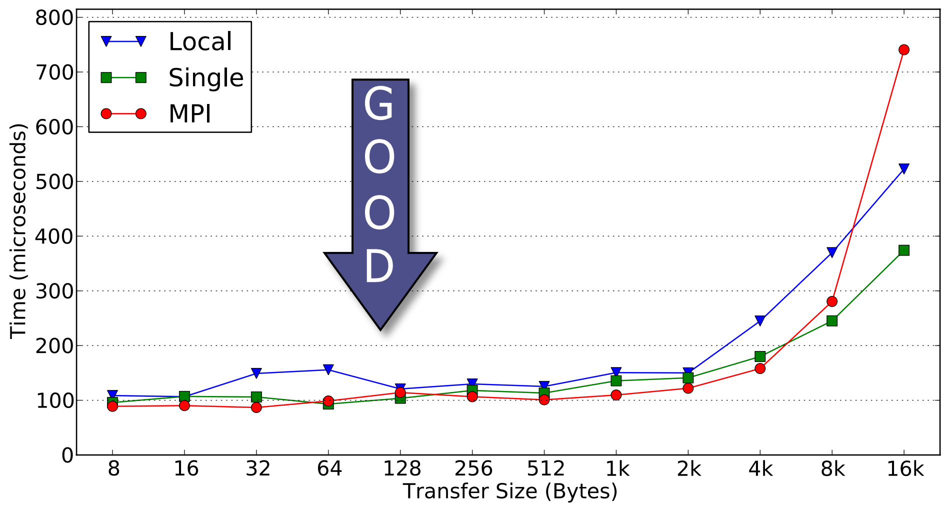
- Broadcast on Sun Constellation (1024 cores)
  - 4-nomial is consistently a "good" performer
  - 8-nomial is best at < 2k bytes

- Broadcast on Cray XT4 (2048 cores)
  - 4-nomial is best < 2k
  - choosing 4-nomial at 32k leads to 2x degradation in performance



Address Modes

- In Global Address Space every thread knows directly where to put the data
  - How do we specify the arguments to the collective?
- Two Options:
  - Single: All nodes provide address for all the other nodes
  - Local: Nodes only provide one address

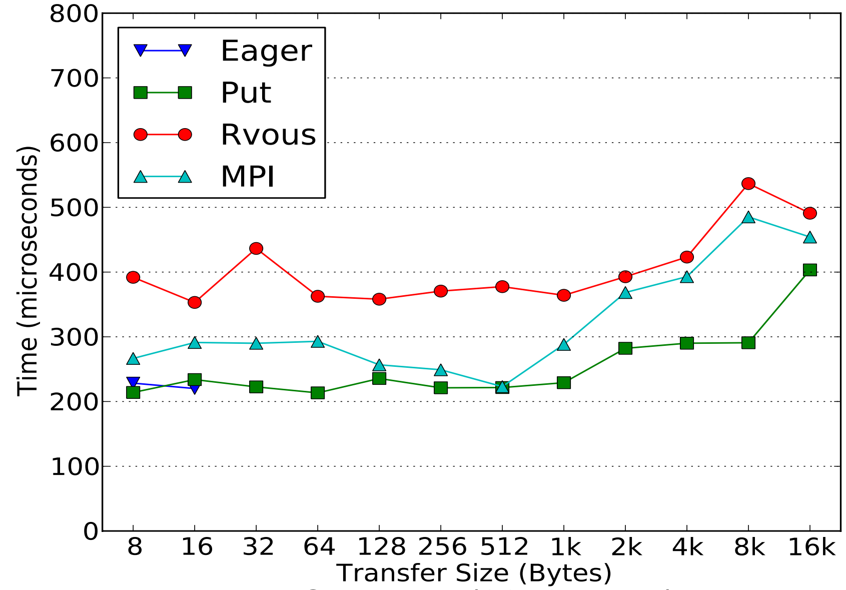


Broadcast on Sun Constellation (1024 cores)

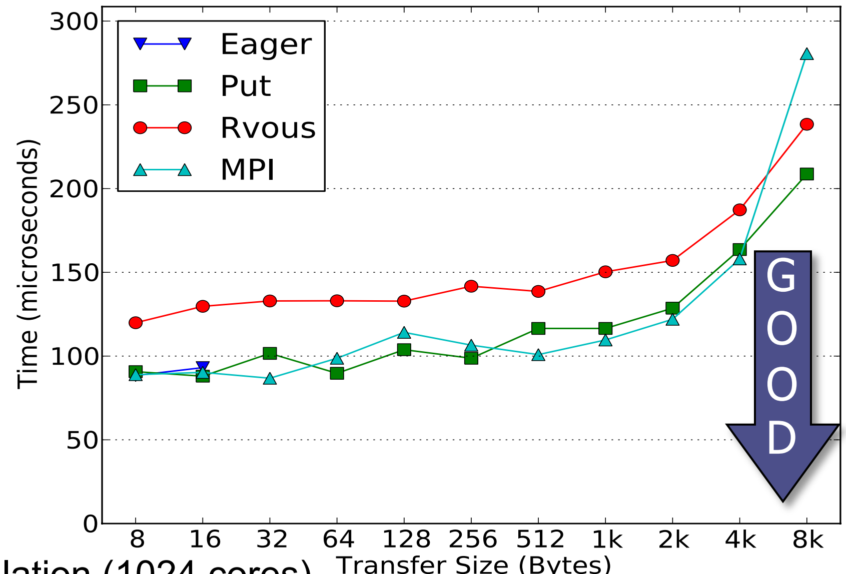
- Single Address Mode
  - Pros: can directly leverage puts/gets without additional overhead
  - Cons: overhead of generating and storing all the addresses
    - In PGAS languages however this is not that high
- Local Address Mode
  - Pros: easy to generate addresses and no meta-data overhead
  - Cons: have to spend time to discover addresses before data can be sent
- Broadcast on 1024 cores of Sun Constellation shows that the cost of address discovery is high at large messages
  - Time spent communicating addresses wastes bandwidth

Data Transfer Mechanisms

- Eager Put
  - Send data to anonymous buffer on target node
  - Uses Medium AM
- Signaling Put
  - Send data and signal target once it has arrived
    - Still one-sided!
  - Needs to know where the data goes
  - Uses Long AM
  - Single-Mode Only
- Rendez-Vous
  - Send child a short message indicating data is read
  - Child does get and sends a short message indicating data is complete
  - AMs for synchronization only



Broadcast on Cray XT4 (2048 cores)

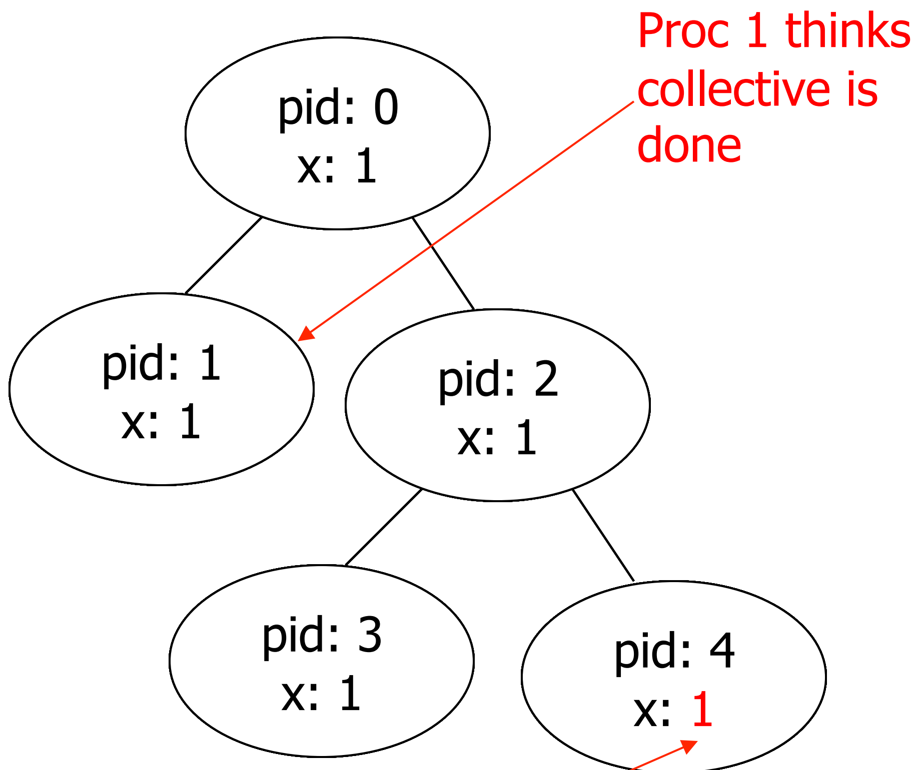


Broadcast on Sun Constellation (1024 cores)

Potential Synchronization Problem

1. Broadcast variable x from root
2. Have proc 1 set a new value for x on proc 4

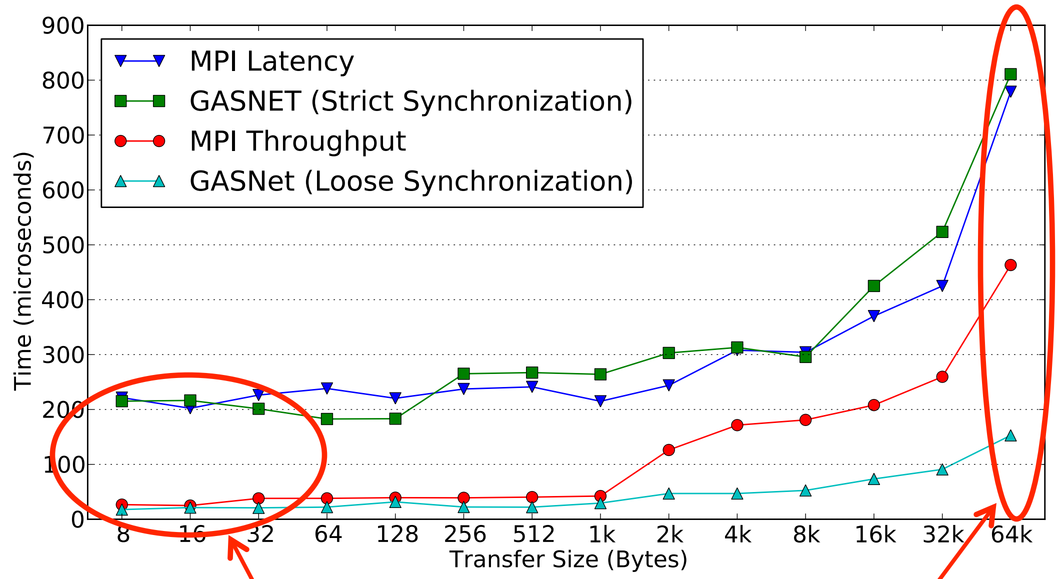
```
broadcast x=1 from proc 0
if(myid==1) {
   put x=5 to proc 4
} else {
   /* do nothing*/
}
```



Put of x=5 by proc 1 has been lost
Proc 1 observes locally complete but globally incomplete collective

Strict v. Loose Synchronization

- A fix to the problem
  - Use synchronization before/after the collective
  - Enforce global ordering of the operations
- Is there a problem?
  - We want to decouple synchronization from data movement
  - Let user specify the synchronization requirements
    - Potential to aggregate synchronization
    - Done by the user or a smart compiler



Cray XT4 Broadcast Performance (1024 Cores)

> 12x faster at small message sizes and > 5x faster at large message sizes!

Nonblocking Collectives

- Relaxing Synchronization still requires at least one processor inside collective
- Overlapping communication w/ computation is a good idea for 1-sided programming models [Nishtala et al. IPDPS'09, Nishtala UCBMS'06]
- How to overlap collectives w/ computation?
  - Two Questions:
    - Can the applications support overlap?
    - Can the hardware support overlap?
- Related work being pursued by MPI community [Hoeffler et al. and Brightwell et al]

… initialize X …

start broadcast of X

… computation unrelated to X…

… unsafe to modify X …

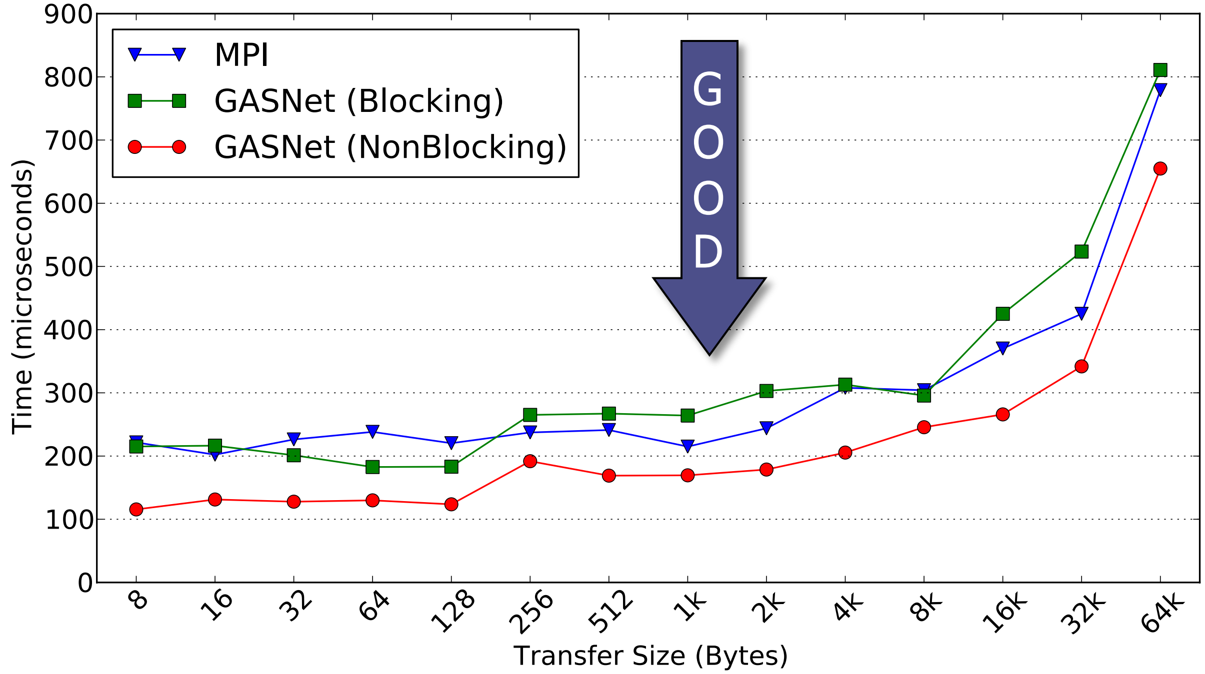
wait for broadcast to complete

…. X can be safely modified …

Code for Root Processor

Performance of Nonblocking Collectives

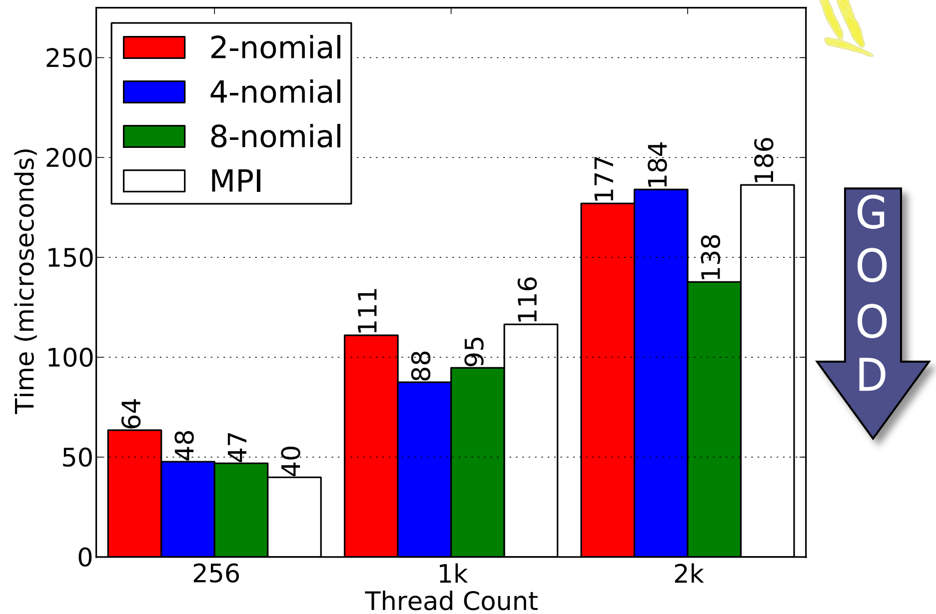
- Benchmark overlaps collectives with each other
  - Collectives pipelined so that the network resources are more effectively used
  - 100-200 microsecond difference
  - We show later how this can be incorporated into a real application
  - All collectives built as state machines
    - State machines make progress on network interrupts or polling depending on platform

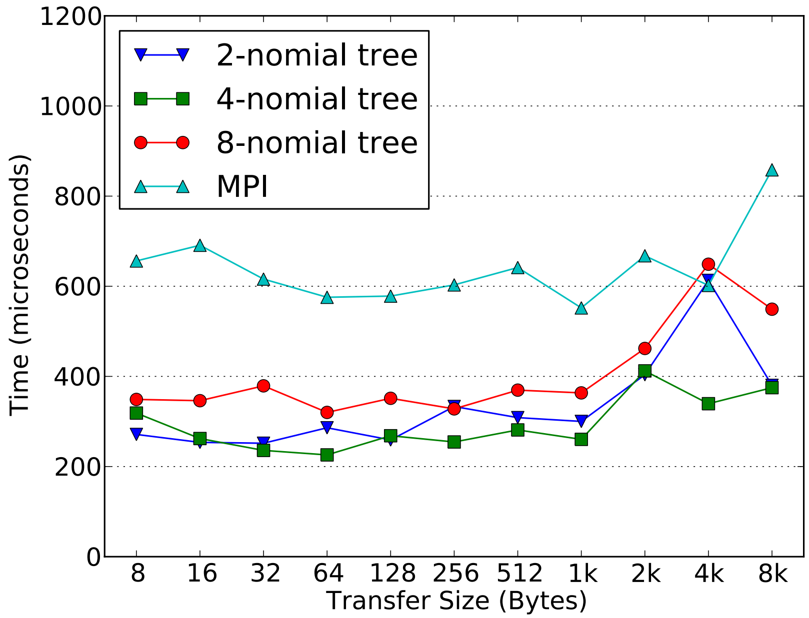


Cray XT4 Nonblocking Broadcast Performance (1024 Cores)

Reduce

- 8-byte Reduce on Sun Constellation
  - 8-nomial tree delivers best or close to optimal performance
  - GASNet outperforms vendor-MPI by 18% at 1k cores and 25% at 2k cores

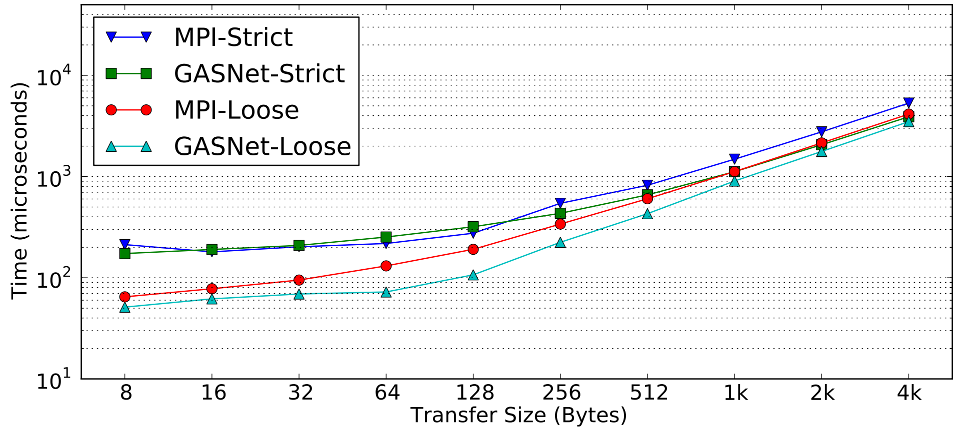




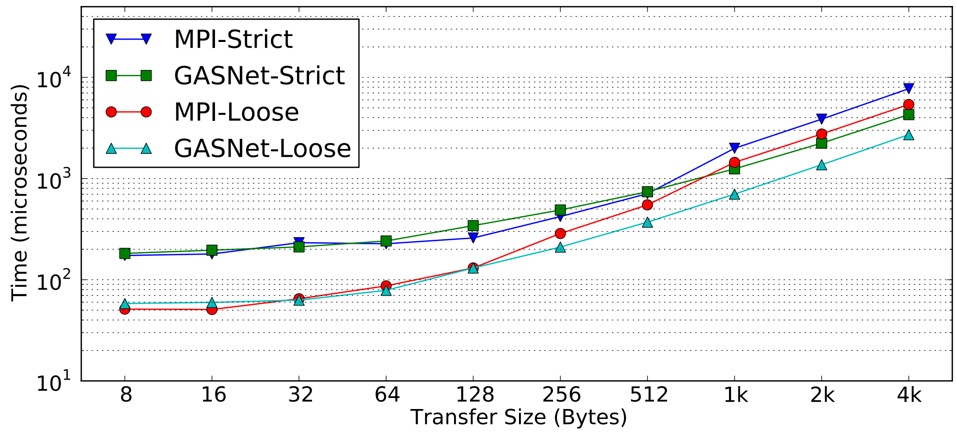
- Reduce on Cray XT4
  - 4-nomial consistently gives a good algorithm
    - Average of 25% better performance over 8-nomial
  - GASNet out performs MPI by > factor of 2x in most cases

Scatter/Gather Performance

- Scatter on 1536 cores of Cray XT5
  - Loose synch. offers 4x performance improvement at low sizes
  - Difference decreases at higher message sizes
  - GASNet is able to deliver better performance for both modes compared to vendor MPI library

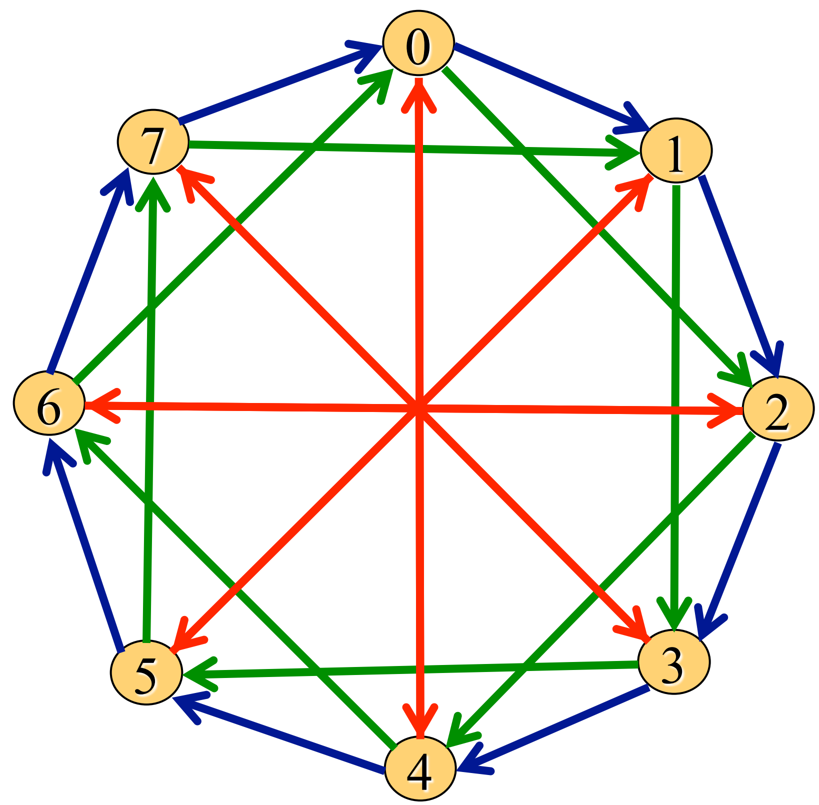


- Gather on 1536 cores of Cray XT5
  - Similar results as Scatter
    - Looser synchronization continues to deliver good performance upto 4k bytes
  - GASNet is able to consistently outperform vendor MPI library



Dissemination for Non-rooted Collectives

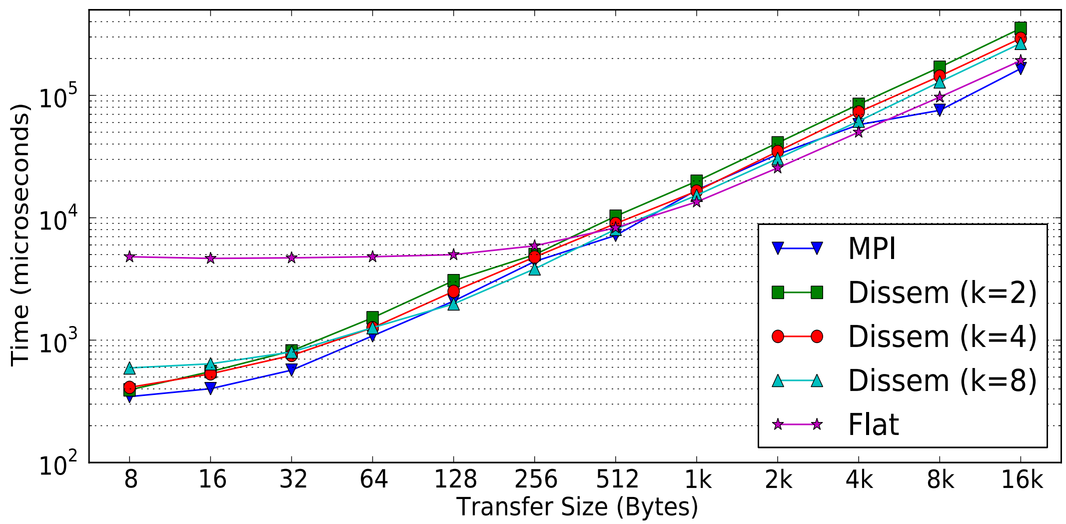
- Flat algorithm: every processor sends to every other processor
  - $O(n^2)$ messages
  - Can we do better by sending through intermediaries?
- Idea: send the data multiple times in the network but communicate with a fewer number of peers
- Collect data from double the number of peers each stage
- Dissemination required all threads to be active all the time
  - O(T log T) "messages"
  - Time: L*(log T) (L = latency)



View from Thread 0	T0	T1	T2	T3	T4	T5	T6	T7
Who knows about T0	✔	✔	✔	✔	✔	✔	✔	✔
Who T0 knows about	✔	✔	✔	✔	✔	✔	✔	✔

Exchange

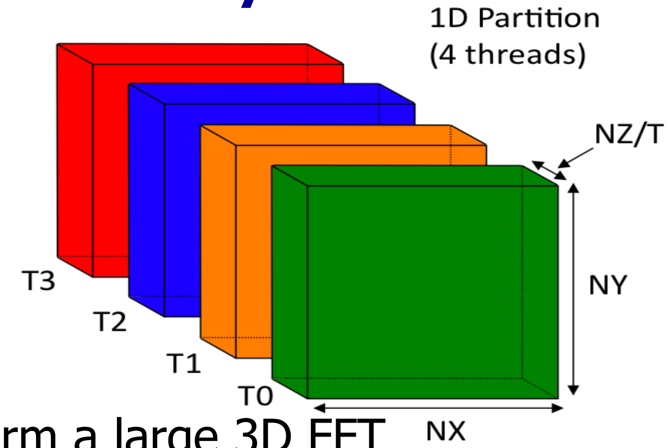
- Dissemination algorithm by Bruck et al. (1997)
  - Send the data multiple times through the network before it reaches the final destination
  - Uses less messages at the cost of more bandwidth
- Highlights a tradeoff between algorithmic choice
  - Intuition suggests there is a crossover point between the algorithms
- Finding the best algorithm is a tuning question that we will address in the automatic tuner section
- Penalty for picking bad algorithm is high
  - Radix-2 is best at 8 bytes but worst at 16k bytes
  - Flat algorithm becomes the best between 512 and 1k byte exchange
    - order of magnitude worse at 8 bytes
    - 28% (~73 ms) faster at 16 Kbytes

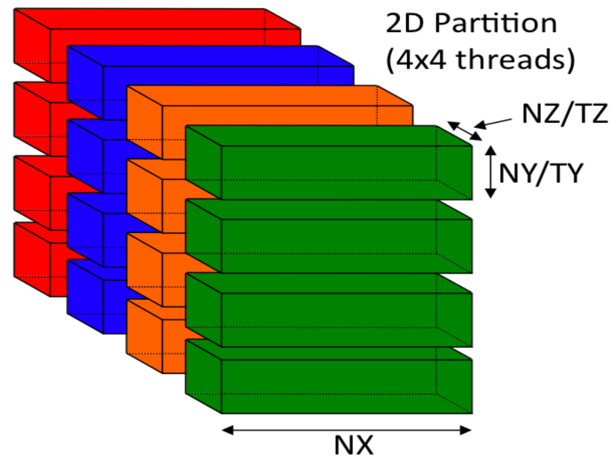


Exchange on Sun Constellation (256 cores)

APPLICATION EXAMPLE

Case Study: NAS FT Benchmark



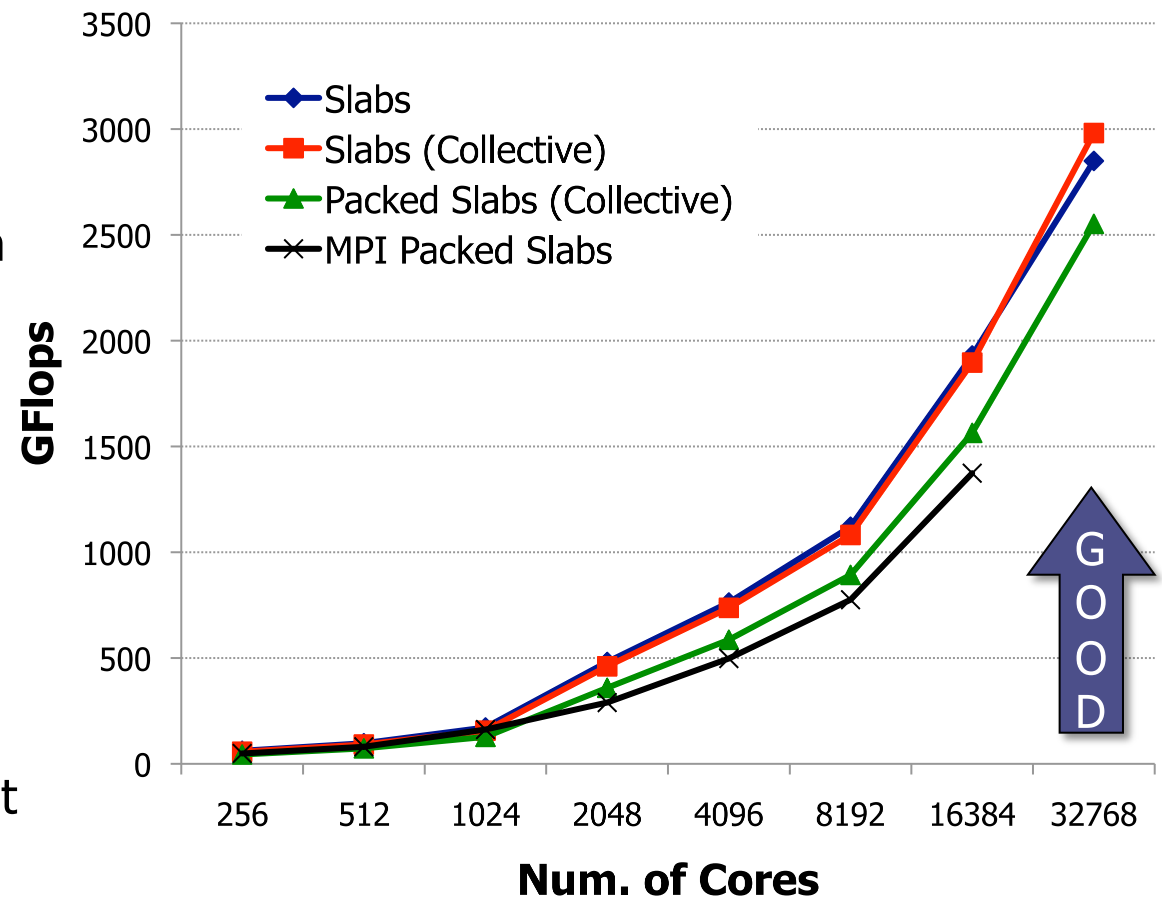


- Perform a large 3D FFT
  - Molecular dynamics, CFD, image processing, signal processing, astrophysics, etc.
  - Representative of a class of communication intensive algorithms
    - Requires parallel many-to-many communication
    - Stresses communication subsystem
    - Limited by bandwidth (namely bisection bandwidth) of the network
- Building on our previous work, we perform a 2D partition of the domain
  - Requires two rounds of communication rather than one
  - Each processor communicates in two rounds with $O(\sqrt{T})$ threads in each
- Leverage nonblocking communication to maximize communication/computation overlap

FFT Performance on BlueGene/P

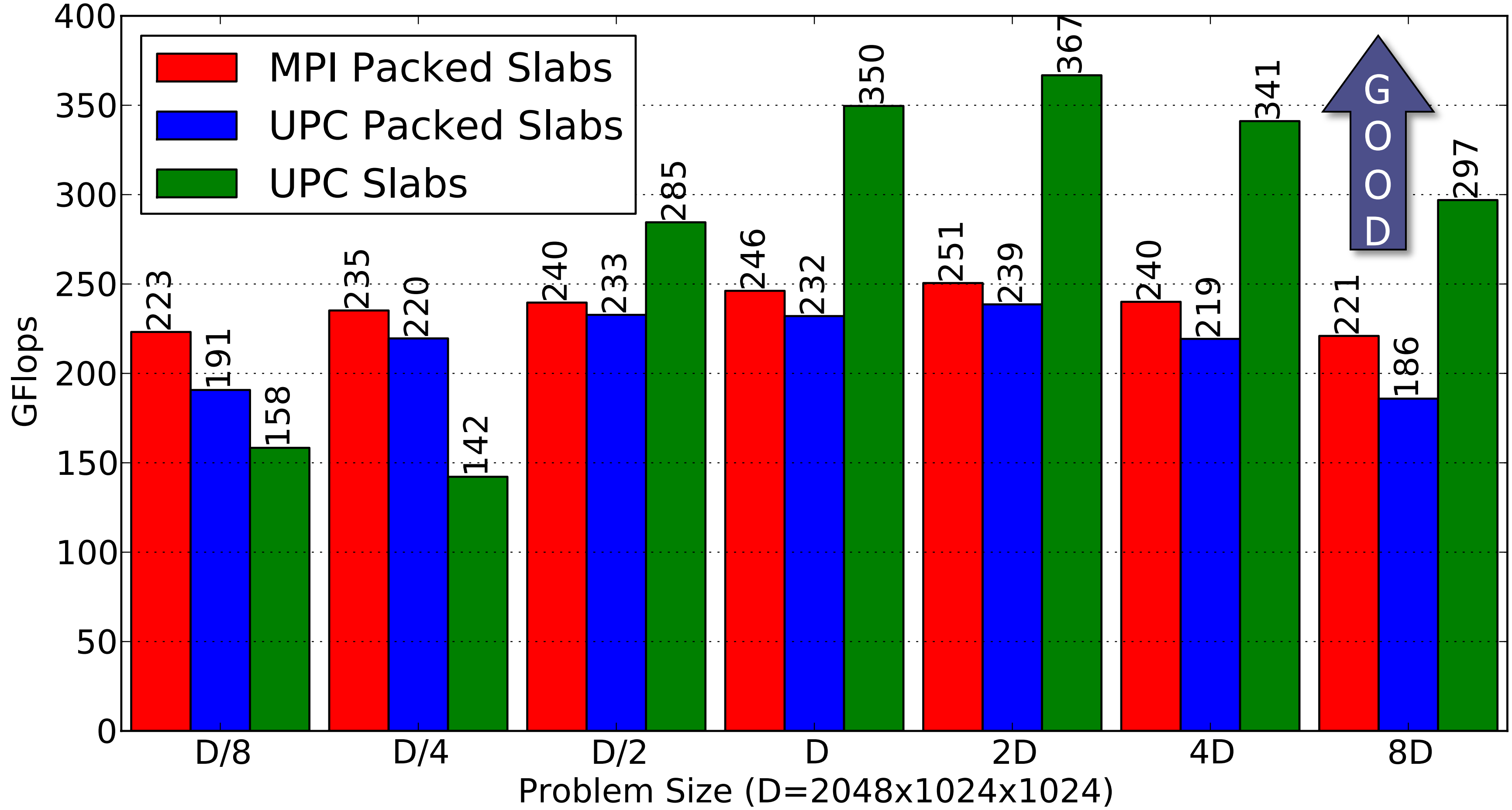
- PGAS implementations consistently outperform MPI
- Leveraging communication/ computation overlap yields best performance
  - More collectives in flight and more communication leads to better performance
  - At 32k cores, overlap algorithms yield 17% improvement in overall application time
- Numbers are getting close to HPC record
  - Future work to try to beat the record

HPC Challenge Peak as of July 09 is ~4.5 TFlops on 128k Cores



FFT Performance on Cray XT4

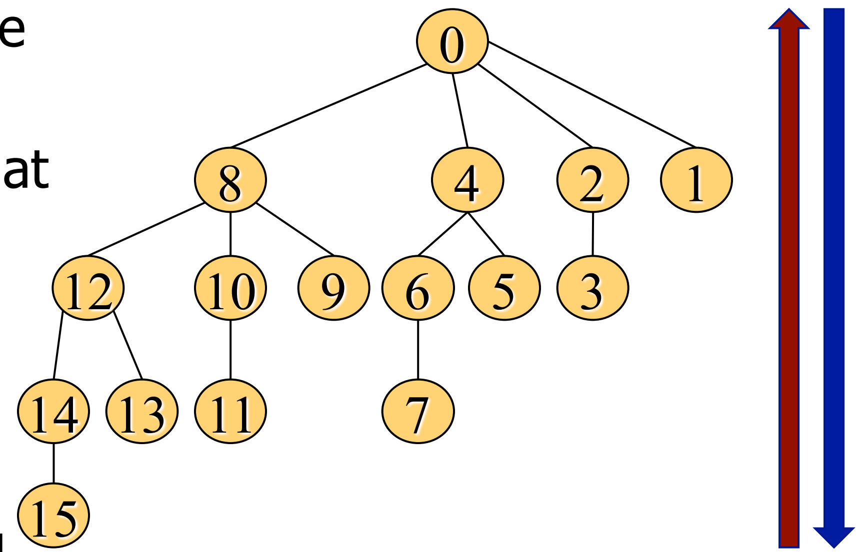
- 1024 Cores of the Cray XT4
  - Uses FFTW for local FFTs
  - Larger the problem size the more effective the overlap



TUNING COLLECTIVE COMMUNICATION FOR SHARED MEMORY

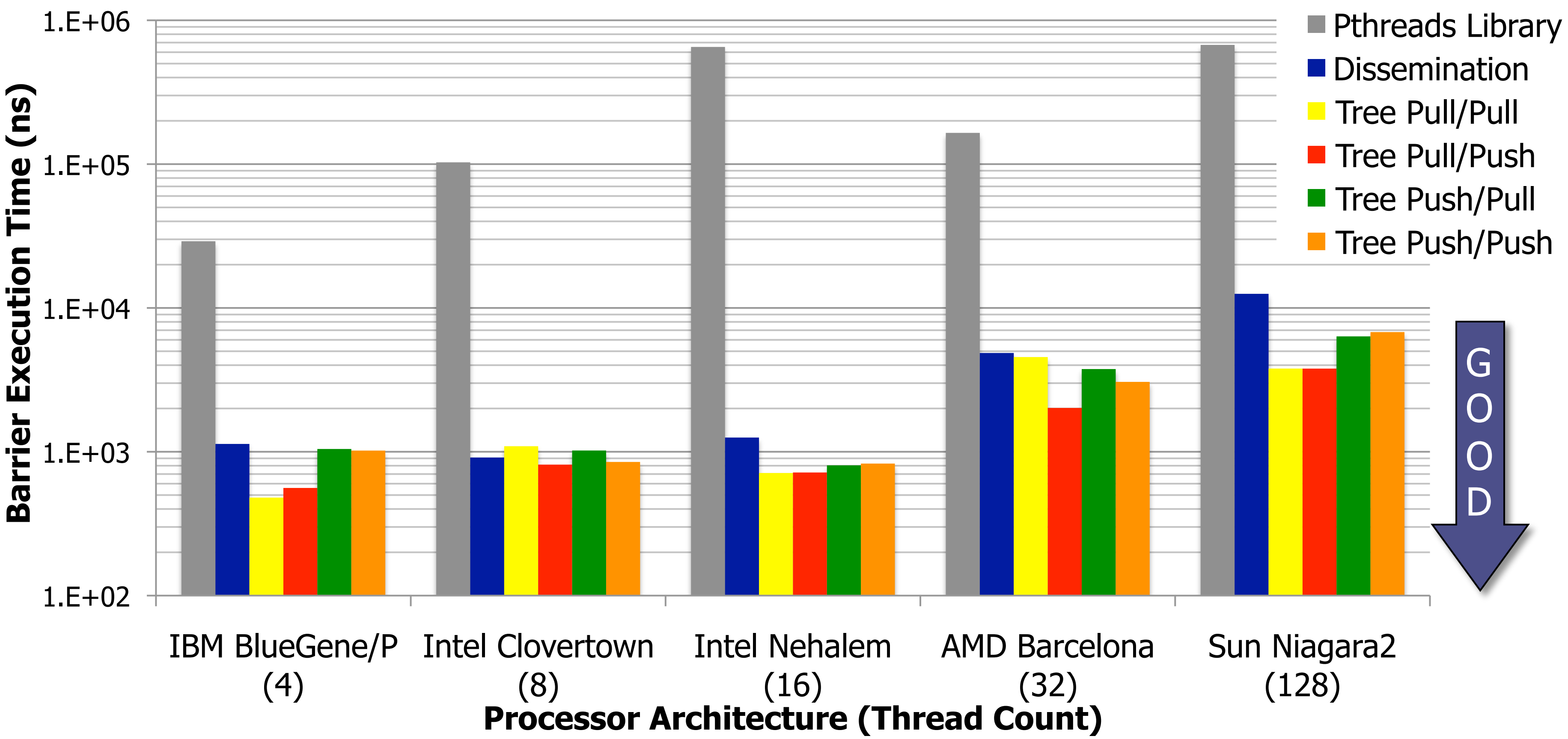
Barrier (tree algorithm)

- Requires two passes of a tree
  - First (UP) pass tells parent subtree has arrived.
  - Second (DOWN) pass indicates that all threads have arrived
  - O(T) “messages”
  - Time: 2L*(log T)
- Two ways to signal others:
  - Push: write a remote variable and spin wait on a local variable
  - Pull: write a local variable and spin on a remote variable



- Leads to 4 unique tree algorithms
- Performance of each is dependent on how systems handle coherency and atomic ops

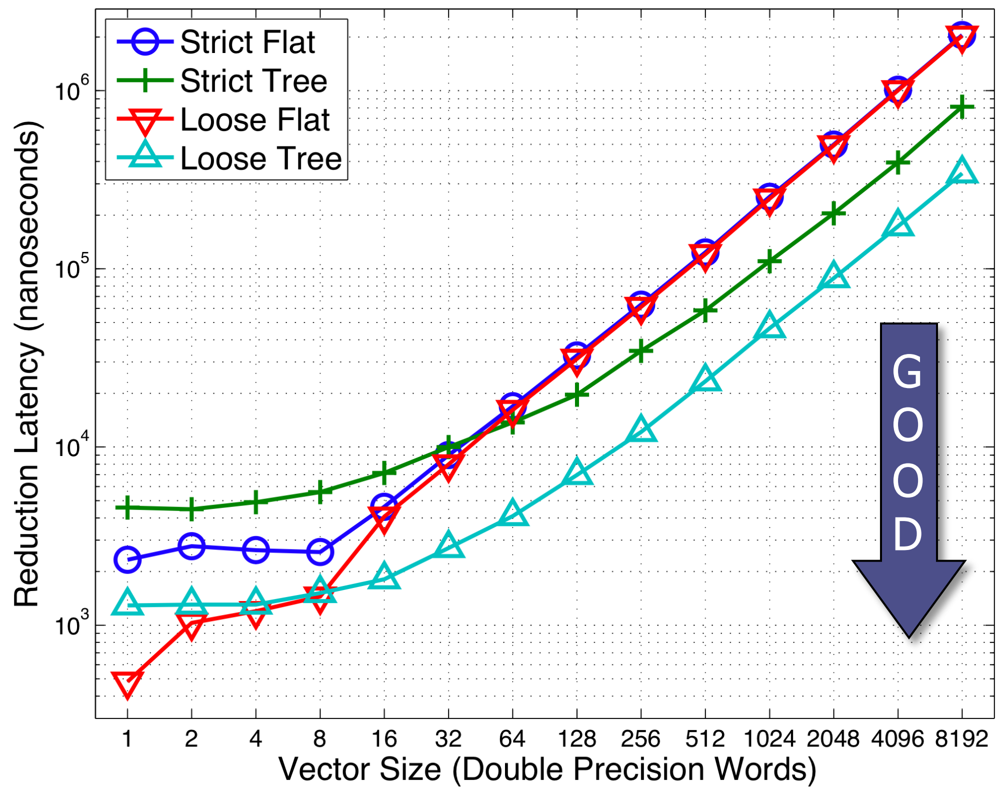
Barrier Performance Results



- "Traditional pthread barriers" yield poor performance
- Performance penalty for picking bad algorithm can be quite substantial
- Same code base across all platforms

Autotuning and Synchronization

- Strict synchronization enforces barriers between collectives to protect shared memory
  - Loose allows user to handle own synchronization
- Tradeoff between Flat and Tree based topology exposes cost of synchronization vs. benefit of extra parallelism
  - Flat trees have little parallelism in the computation but require less synchronization
- Optimal algorithm is affected by the synchronization flags
- Looser Synch. enables trees to realize better performance at lower message sizes

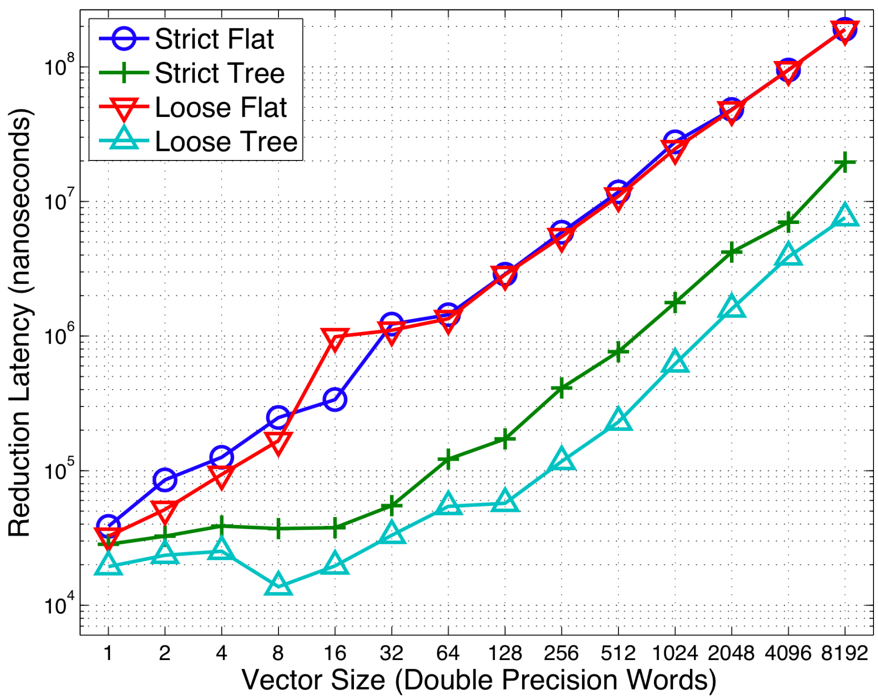


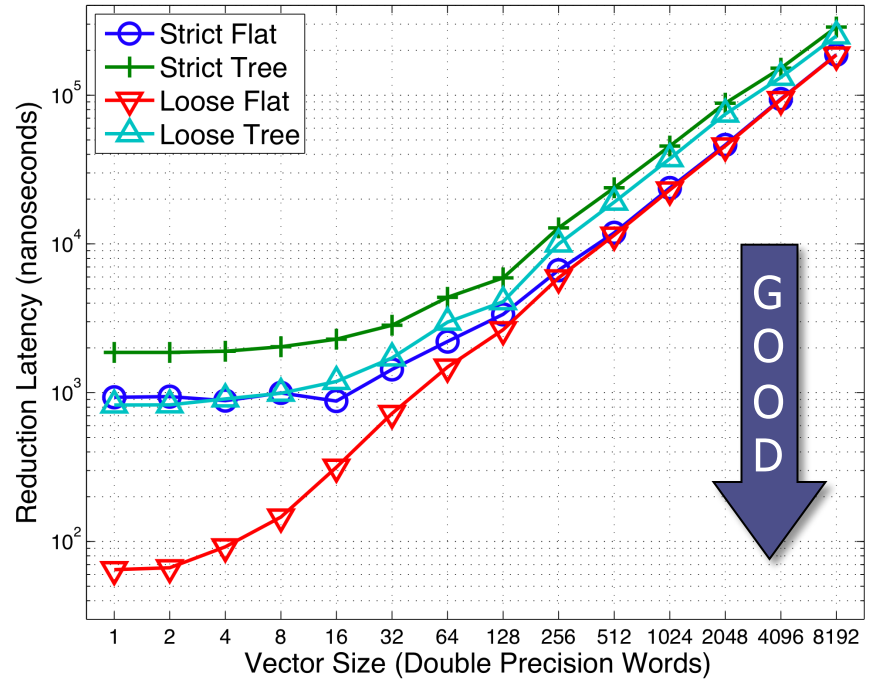
AMD Opteron (32 threads)
Reduction Performance

Autotuning and Synchronization (cont.)

- Different platforms have different crossover points between the algorithms
- On Intel Clovertown, flat algorithms always beat out the trees

Sun Niagara 2 (256 threads)
Reduction Performance





Intel Clovertown (8 threads)
Reduction Performance

- However on Sun Niagara2 the trees always win
  - High thread count implies that scalable collectives must be implemented for all sizes

SOFTWARE ARCHITECTURE OF THE AUTOMATIC TUNER

Automatic Tuning Overview

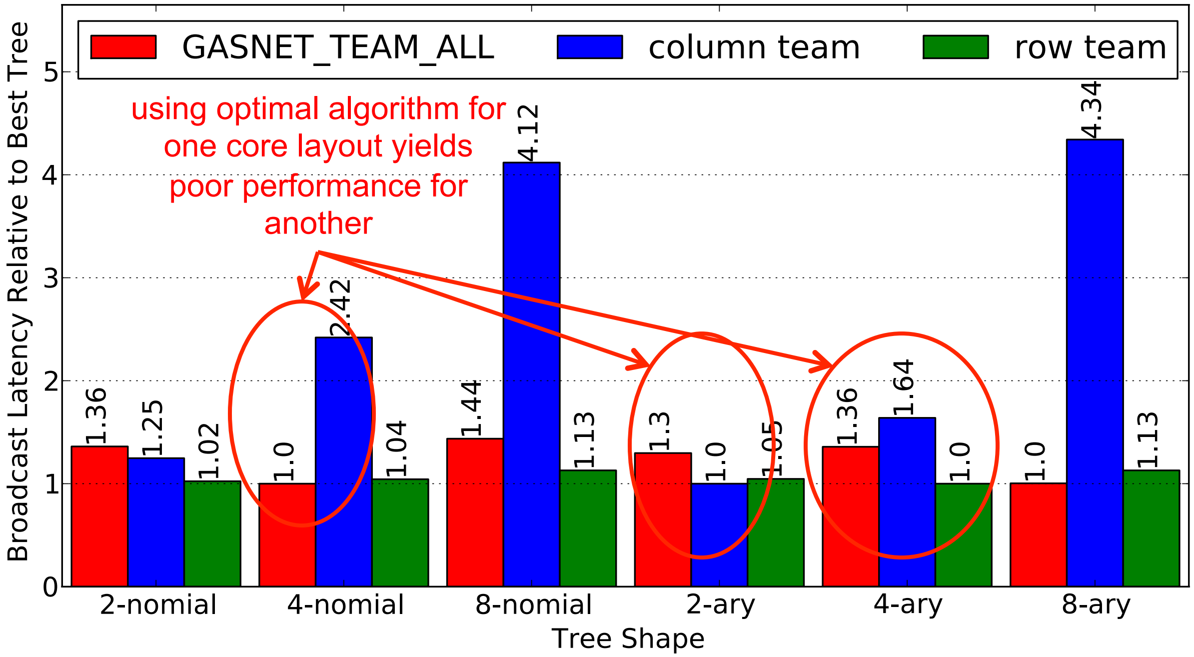
- Each collective have many implementations in GASNet
  - Variants such as eager, rendezvous, direct put, direct get
  - Orthogonally, there are many possible trees that we can use
- GASNet collective infrastructure indexes all the algorithms
  - Hardware collectives for certain conduits go into this index
    - Allows for easy extensibility for new algorithms and platforms
  - Each collective algorithm advertises capabilities and requirements
    - Not all algorithms have to work for in call cases
  - Tuning can be done either online or offline depending on how much time the user is willing to devote for search
  - Like FFTW and other automatic tuning projects, the automatic tuning data is saved across runs
- Performance models will be used to prune search space
    - Need the constants for the models!
    - More accurate the models the less time devoted to search
    - Models can't capture important features like network load so some search will still be needed

Automatic Tuning Overview (cont.)

- Portable Performance
  - Many factors that influence the optimal algorithm
  - Importance of different factors depend on the target platform
- Some factors are very difficult to capture through analytic models and necessitate search

INSTALL-TIME	RUN-TIME
• Processor type/speed • Memory system • Number of cores per socket • Number of network cards • Interconnect Latency • Interconnect Bandwidth • Interconnect Topology	• Number of processors • Sizes of the messages • Synchronization mode • Processor connectivity • Network load • Mix of collectives and computation

Layout Matters

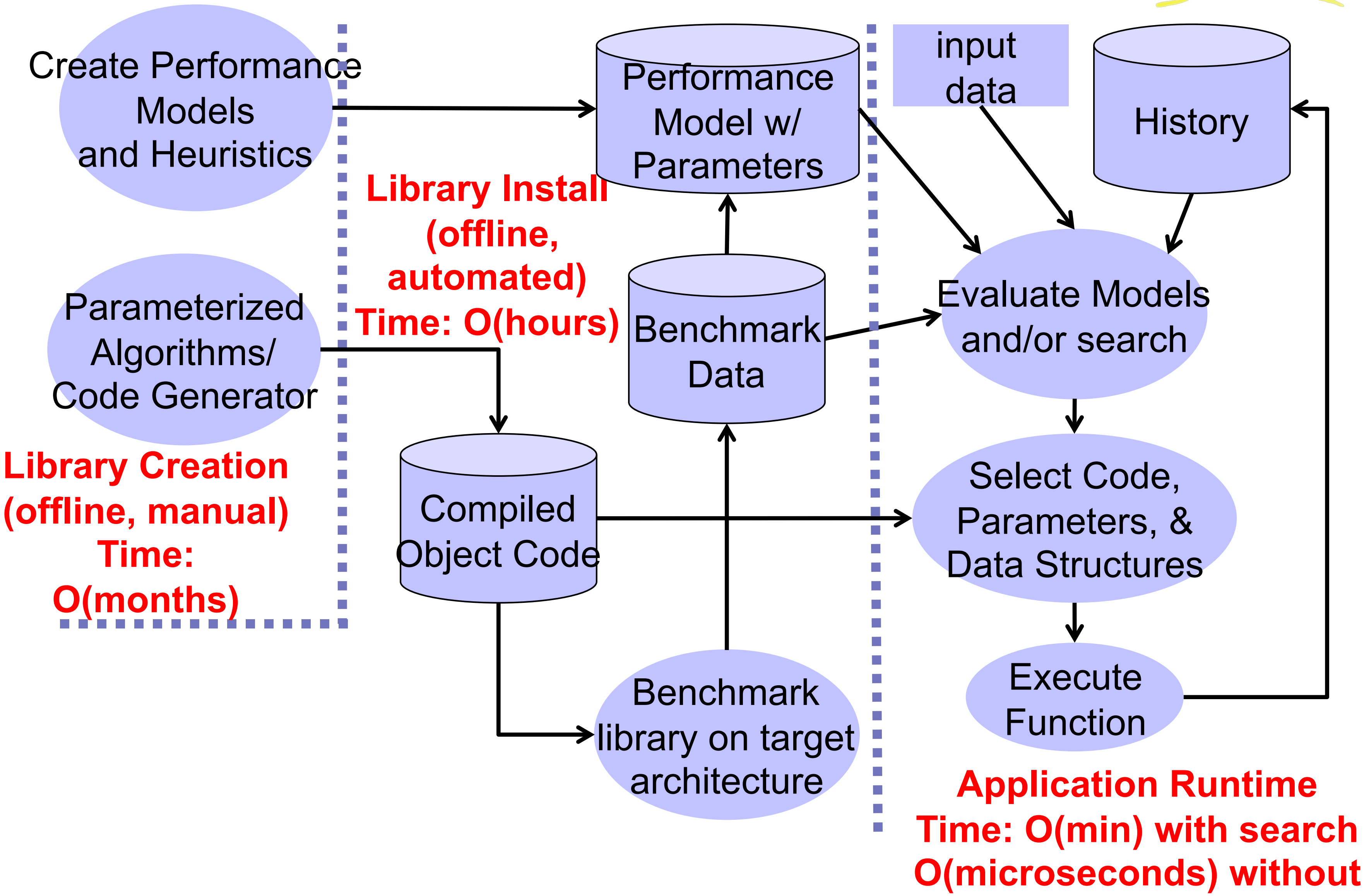


- 256 cores Sun Constellation
  - 16 nodes with 16 cores per node
    - 16 x 16 processor grid
  - make row teams
    - All cores in one node are part of the same team
  - make column teams
    - Core i from each node is part of team i
- Team members and layout known only at runtime

Previous Successful Efforts

- ATLAS: Dense Linear Algebra
  - Tuning can be done offline so tuning is done at install time
- Spiral and FFTW: Spectral Methods
  - Tuning can be done offline or via code generator
  - Introduce idea of tradeoff between the quality of the solution and time to solution
- Sparsity and OSKI: Sparse Linear Algebra
  - Input matrix matters so tuning has to be done online
  - Use offline heuristics and models to make educated guesses
  - Also introduces idea of specifying quality of algorithm to search time
- Parallel SpMV, Parallel LBMHD and Parallel Stencil Computations
  - Outlined issues that arise with automatic tuning for parallel programming models
  - Roofline models outlined the important aspects of performance tuning for parallel systems
- MPI Collective automatic tuning
  - Closely related work but the MPI collectives have some different tuning goals than UPC/GASNet

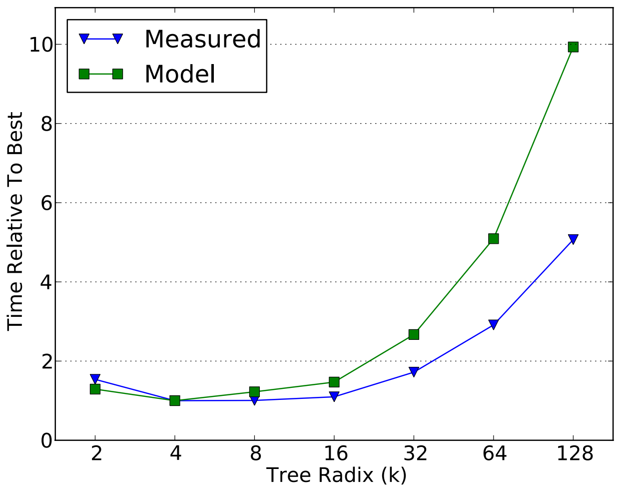
Automatic Tuner Flowchart



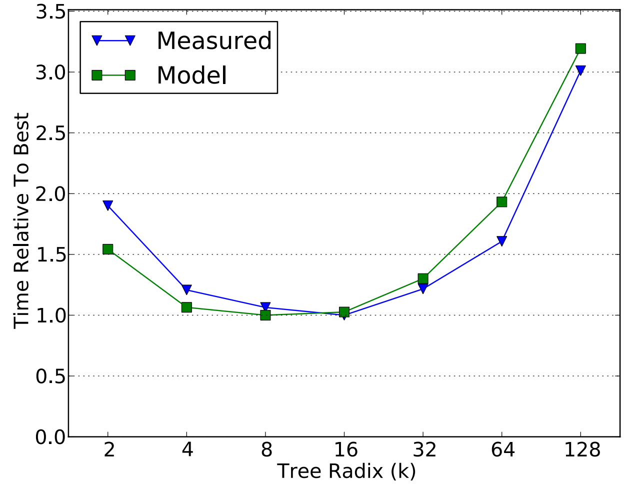
Performance Models

- The optimal collective algorithm depends on many factors
  - Network Performance, processor performance, message size, synchronization mode, etc
- Searching over all possible candidate algorithms at large scale is too expensive
  - Takes too long for exhaustive search
  - Time is money (literally at most cloud/computing centers)
- Minimizing time for search allows search to happen online
- Model constructed using LogGP [Alexandrov et al., '97]
  - Extension of LogP [Culler et al. '93]
  - L (Latency): time taken for message to travel across the network
  - o (overhead): CPU time needed to inject or receive a message from the network
  - g (gap): time between successive message injections or receives
  - G (inverse bandwidth): cost to put a byte into the network for large messages
  - P (number of processors)
- Use performance models to guide the search

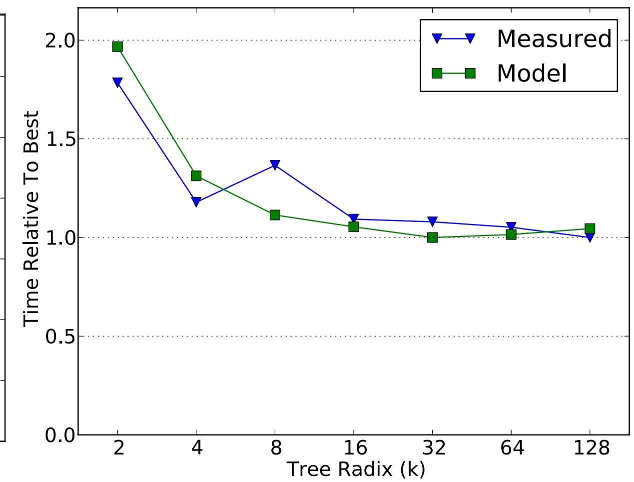
Performance Model: Scatter



8 byte Scatter



128 byte Scatter

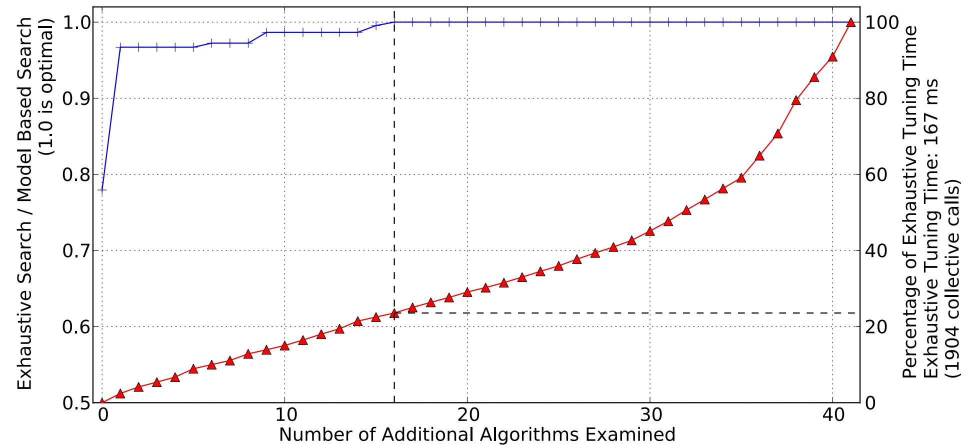


8k byte Scatter

- Scatter Performance Model Verification on 1024 Cores of Sun Constellation
- Goal of Model: Accurately sort the search space and pick the best tree
  - Accurate performance prediction is a nice-to-have but not a need-to-have
- Smaller radices maximize parallelism but also increases bandwidth
  - Data is duplicated in the network many more times
  - As messages increase bandwidth becomes more important
- Models accurately capture trends

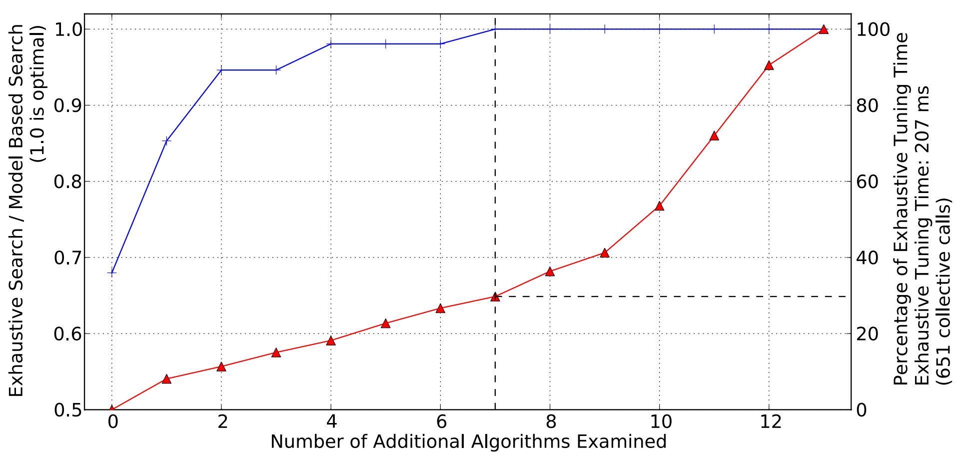
Guided Search

- Sort the algorithm/parameter space based on the performance model
  - Slow algorithms placed at the end
  - Searching just a handful yields an a good algorithm
    - Have to search 17 algorithms to find best
      - 40% of the total space
    - Takes 25% of the search time



8 byte Broadcast on Sun Constellation (1024 cores)

- Fewer algorithms in the search space
- Search takes 8 algorithms to find the best
  - However can get to within 90% of the best after just searching 3
- Tradeoff time to search for the accuracy of the result
  - Similar to what FFTW and OSKI currently offer



128 byte Scatter on Cray XT5 (1536 cores)

SUMMARY AND FUTURE WORK

Future Work

- Add in more collective algorithms as they are discovered
  - Automatic tuning system was designed to be extensible
- More accurate performance models
  - The more accurate the model the less time to do the search
- Statistical Learning
  - Use statistical learning methods to further guide the search and be able to explore even more algorithms
- More Apps in PGAS languages
  - Microbenchmarks can only shed so much light on the story
- More novel collective interfaces
  - MPI-like SPMD collectives are very rigid
    - PGAS languages break this model in some novel ways that introduces more interesting tuning
  - How would collectives look like in new languages
  - How easily can these techniques be applied in MapReduce and Hadoop?

Summary

- Future performance gains are primarily going to be from parallelism
  - Optimally communicating data between the cores is key
  - Need to abstract common communication patterns so that they can be hidden behind a library and be well tuned and reused
  - Allow collectives to be overlapped with computation to ensure best usage of available resources
- Optimal collective performance varies based on many things
  - Need to choose the best algorithm at runtime
  - Many ways to implement the same collective
- System architectures for both distributed and shared memory platforms are getting more diverse
  - New interconnect topologies and increased sharing of parallel systems
  - Need a system that can automatically tune the operations
    - Don’t want to retune the collective for every new platform or topology
  - Implement a family of algorithms that perform the same collective
    - Each is well suited for certain cases
- Use performance model to decrease the time needed for search

Don’t take my word for it!

- Automatically tuned collectives have been incorporated into latest release of Berkeley UPC and GASNet

- Download all the source code from http://upc.lbl.gov
  - Current usage:
    - upcc program.upc
    - env GASNET_COLL_ENABLE_SEARCH=1 upcrun –n 4 ./a.out
  - Full documentation available online

Acknowledgements

- Kathy and Jim for all their support and invaluable advice over the years
- Quals Committee: Dave Patterson and Panos Papadopoulos
- BeBOP Group Past and Current Members
  - Kaushik Datta, Shoaib Kamil, Sam Williams, Mark Hoemmen, Rich Vuduc, Ankit Jain, etc
- Berkeley UPC Group Past and Current Members
  - Paul Hargrove, Dan Bonachea, Yili Zheng, Christian Bell, Costin Iancu, Filip Blagojevic, Wei Tu, Seung-Jai Min, Jason Duell, etc
- Rest of the Parlab
  - Krste Asanović, John Kubiatowicz, Jimmy Su, Amir Kamil, Heidi Pan, Chris Batten , etc …

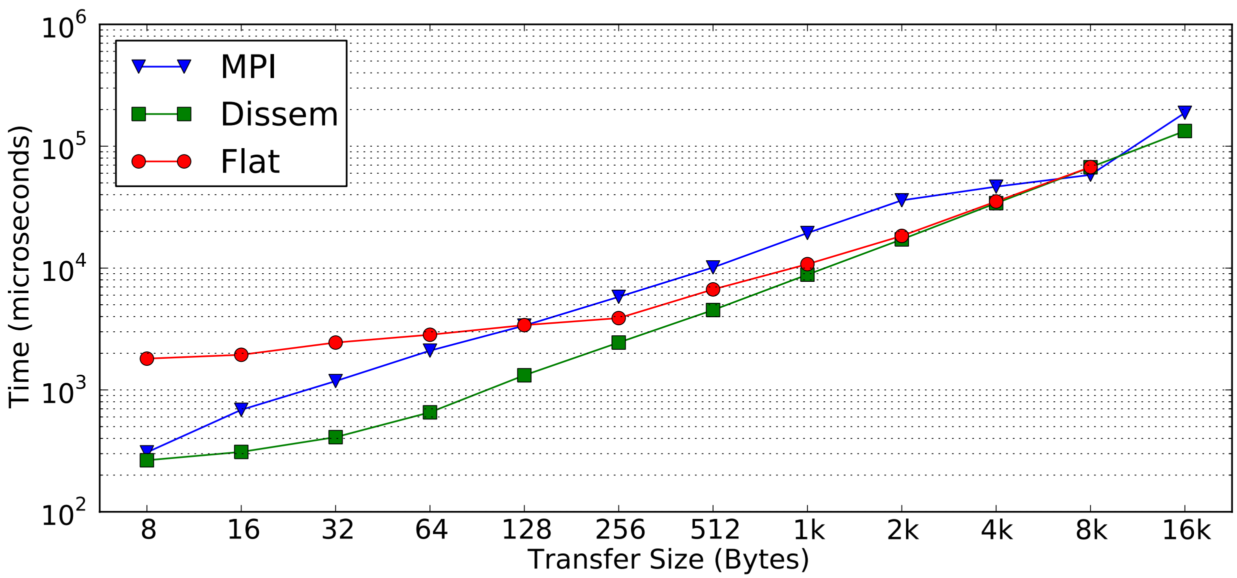
- Keep in Touch! Write on my Wall ;-)

GO BEARS!

THANKS! ANY QUESTIONS?

BACKUP SLIDES

Gather-To-All

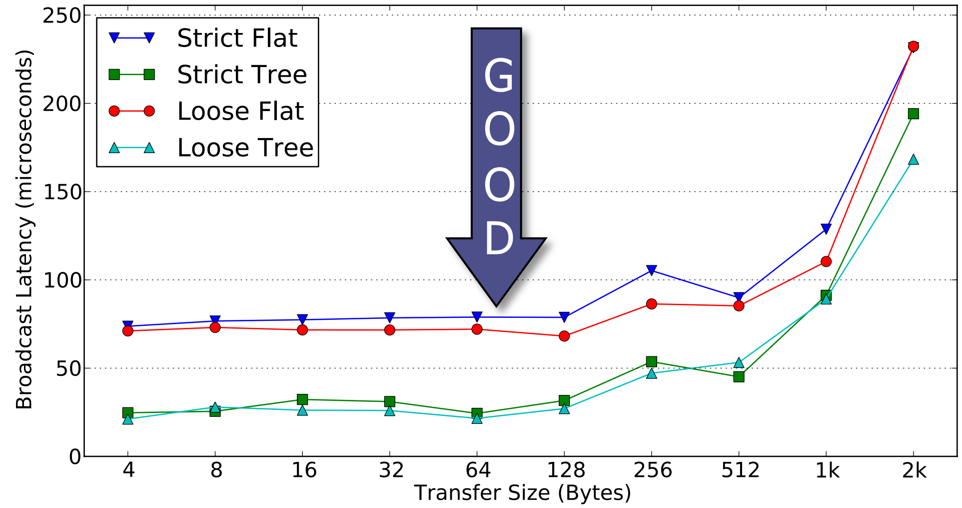


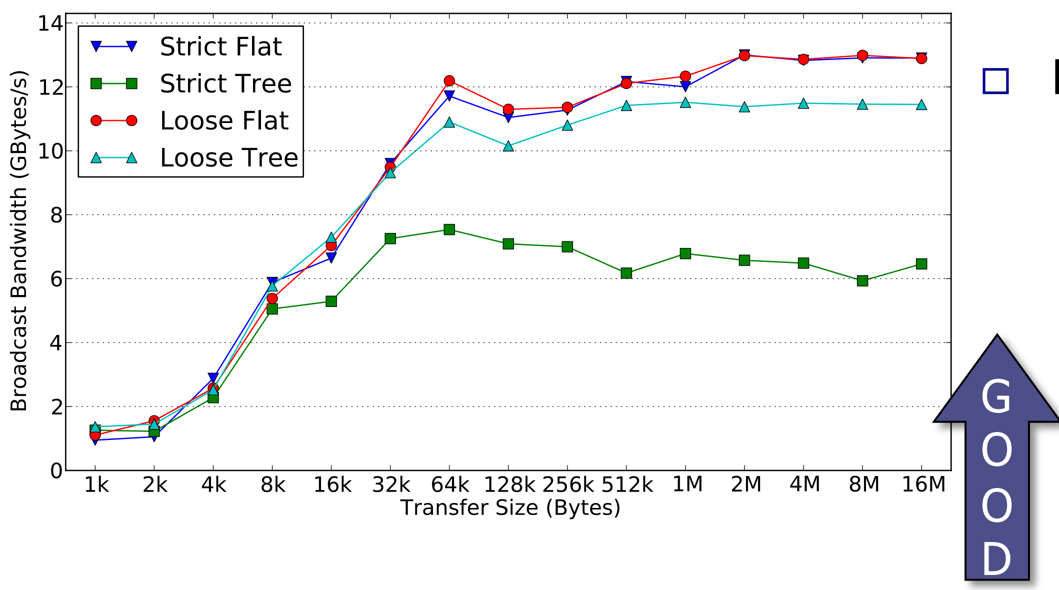
Gather-to-All on Cray XT5 (1536 Cores)

- Unlike Exchange Gather-to-All sends same message to everyone
  - W/ Dissemination algorithm, message sizes double at every round
    - Dissemination algorithm does not use extra bandwidth
  - Same operation can be done in fewer O(n log n) messages rather than $O(n^2)$ and thus Dissemination always wins
- GASNet consistently outperforms MPI

Sun Niagara2 Broadcast

- Broadcast latency on 128 threads
  - Loosening the synchronization doesn't help
  - Memory system resources are shared
    - Harder to get collectives pipelined behind each other
  - Trees yield important improvements





- Broadcast bandwidth on 128 threads
  - Flat trees yield the best bandwidth
  - Most efficient to use flat trees
    - Data becomes too large to fit in caches
    - Using one thread yields the best person

3D FFT: Packed Slabs

Message Size Round 1	(NZ/TZ) × (NY/TY) × (NX/TY) elements
# Messages in Round 1	TY
Message Size Round 2	(NZ/TZ) × (NX/TY) × (NY/TZ) elements
# Messages in Round 2	TZ

- Perform communication and computation in two distinct phases
  - First perform the computation for all the rows in X-dimension
    - Communication system is idle during this time
  - Perform a Transpose to relocalize the Y-dimension
    - Requires Packing and Unpacking
    - Performed across all the processors with the same color
  - Perform the FFT for all the columns
  - Perform a transpose to relocalize the Z-dimension
  - Perform the final set of FFTs
- As per conventional wisdom, data is packed to increase message size
  - Only exploits communication/communication overlap during the transpose
  - MPI implements transpose as in memory data movement plus one call to MPI_Alltoall() for each round
    - Minimum number of calls to MPI

3D FFT: Slabs

- Observation:
  - After one of the NZ/TZ planes of row FFTs is done we can start transferring the data
  - Allows communication/ communication overlap and communication/ computation overlap

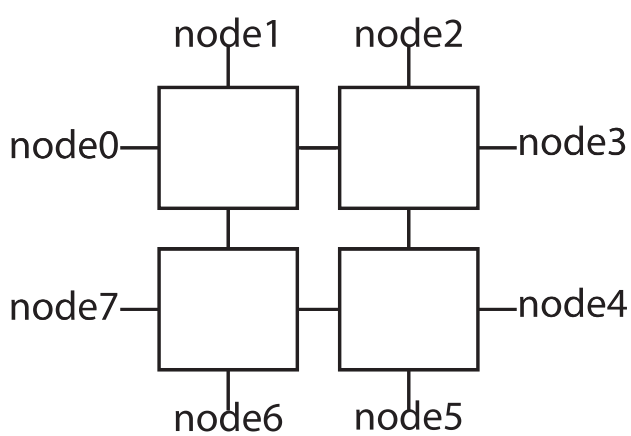
Message Size Round 1	(NY/TY) × (NX/TY) elements
# Messages in Round 1	(NZ/TZ) × TY
Message Size Round 2	(NX/TY) × (NY/TZ) elements
# Messages in Round 2	(NZ/TZ) × TZ

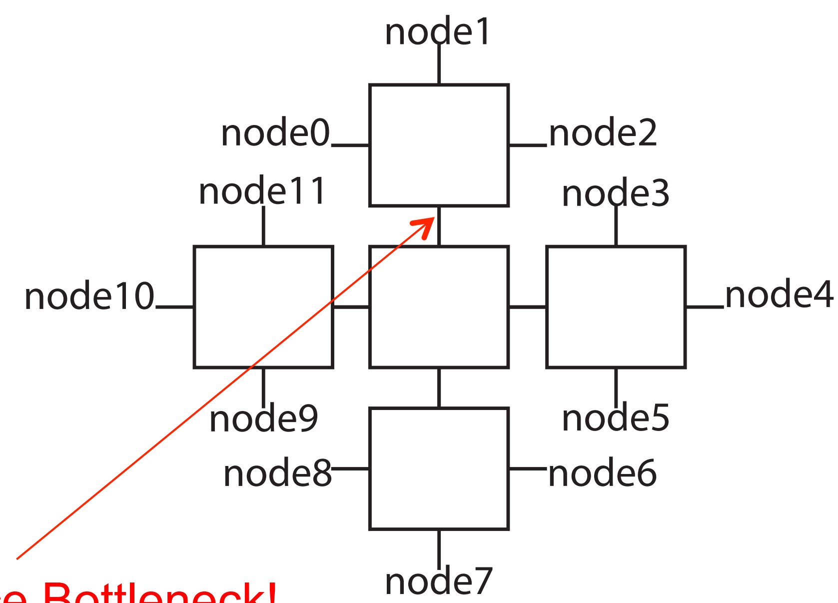
- Algorithm sketch:

1. for each of the NZ/TZ planes
   1. perform all NY/TY row FFTs (len NX)
   2. pack data for this plane
   3. initiate nonblocking all-to-all
2. wait for all all-to-alls to finish
3. unpack data
4. for each of the NZ/TZ planes
   1. perform all NX/TY row FFTs (len NY)
   2. pack data for this plane
   3. Initiate nonblocking all-to-all
5. wait for all all-to-alls to finish
6. unpack data
7. perform last round of (NY/TZ) (NX/TY) FFTs (len NZ)

Switched Networks

- Nodes can be connected through intermediary switches
  - A switch is a device that can route a message between any input port to any output port
  - Use multiple levels of switches to connect many pieces of the network together



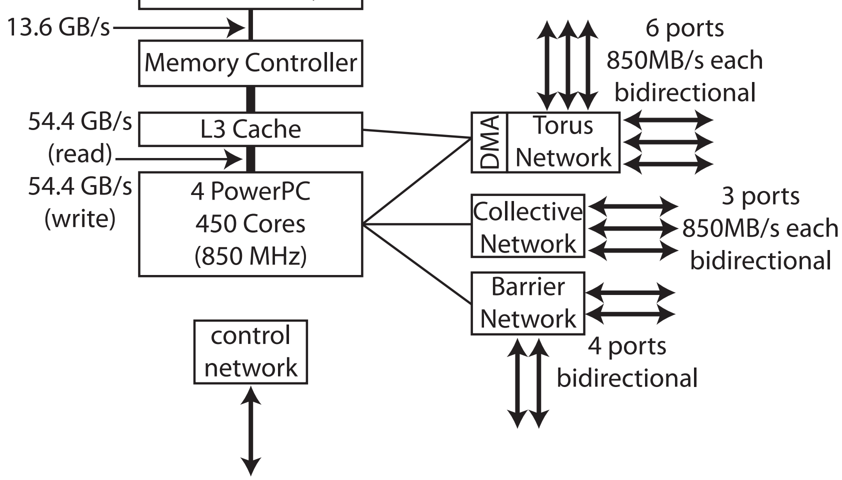


Performance Bottleneck!

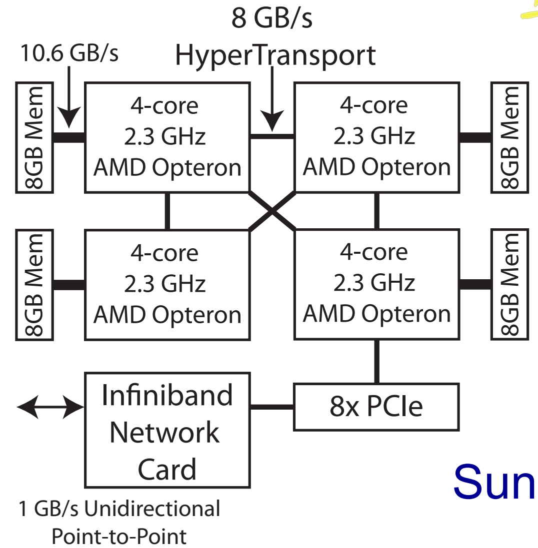
Bandwidth to different parts of the network is 1/3 of local bandwidth

Node Architectures

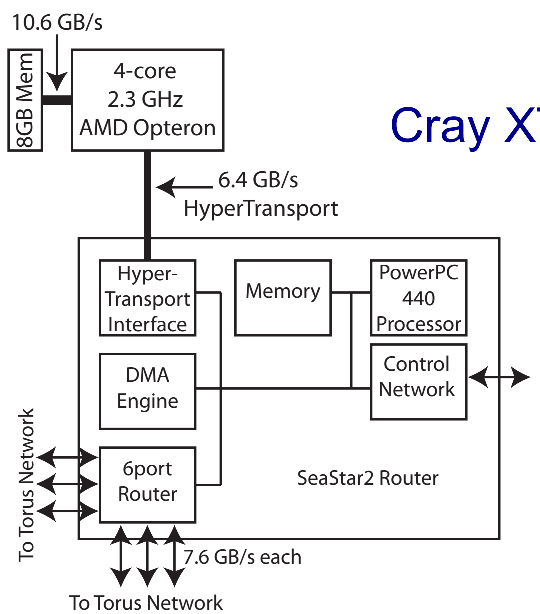
IBM BlueGene/P



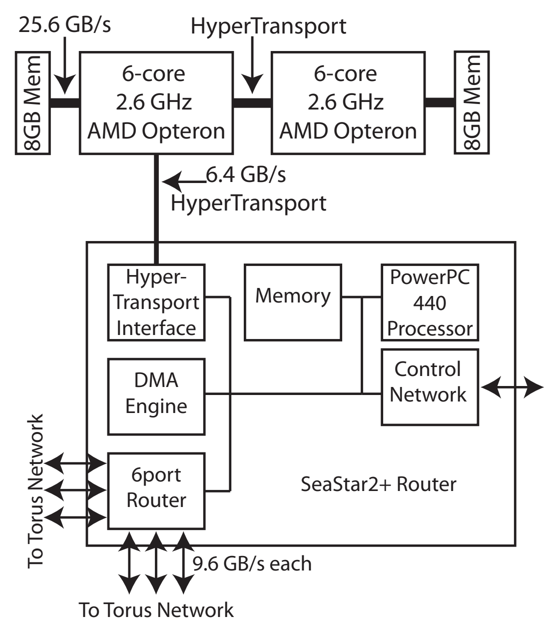
Sun Constellation



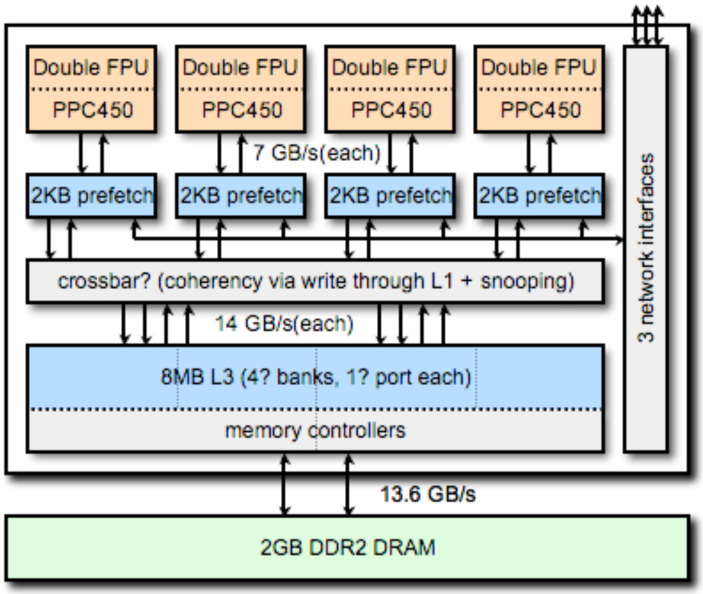
Cray XT4



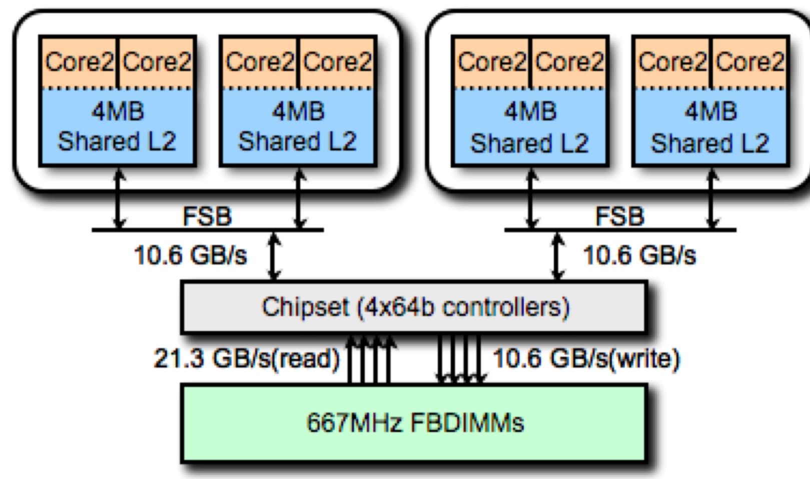
Cray XT5



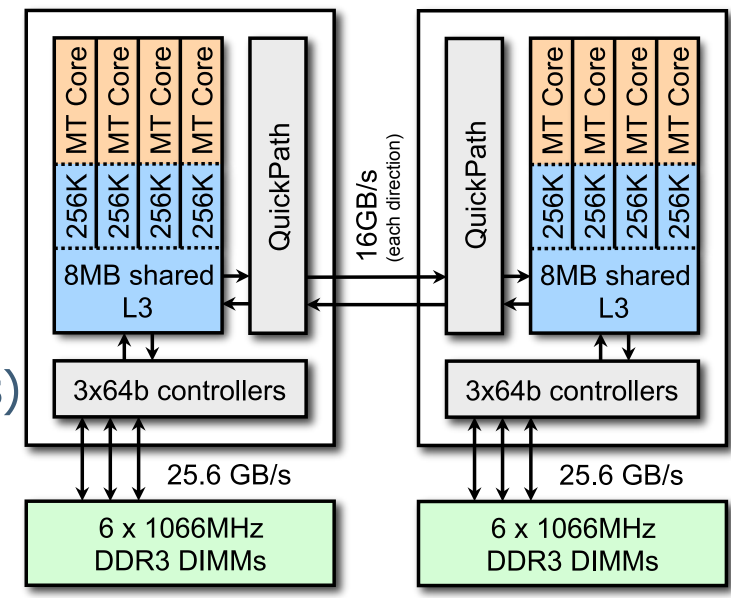
Modern Shared Memory Systems



IBM BlueGene/P (4 threads)



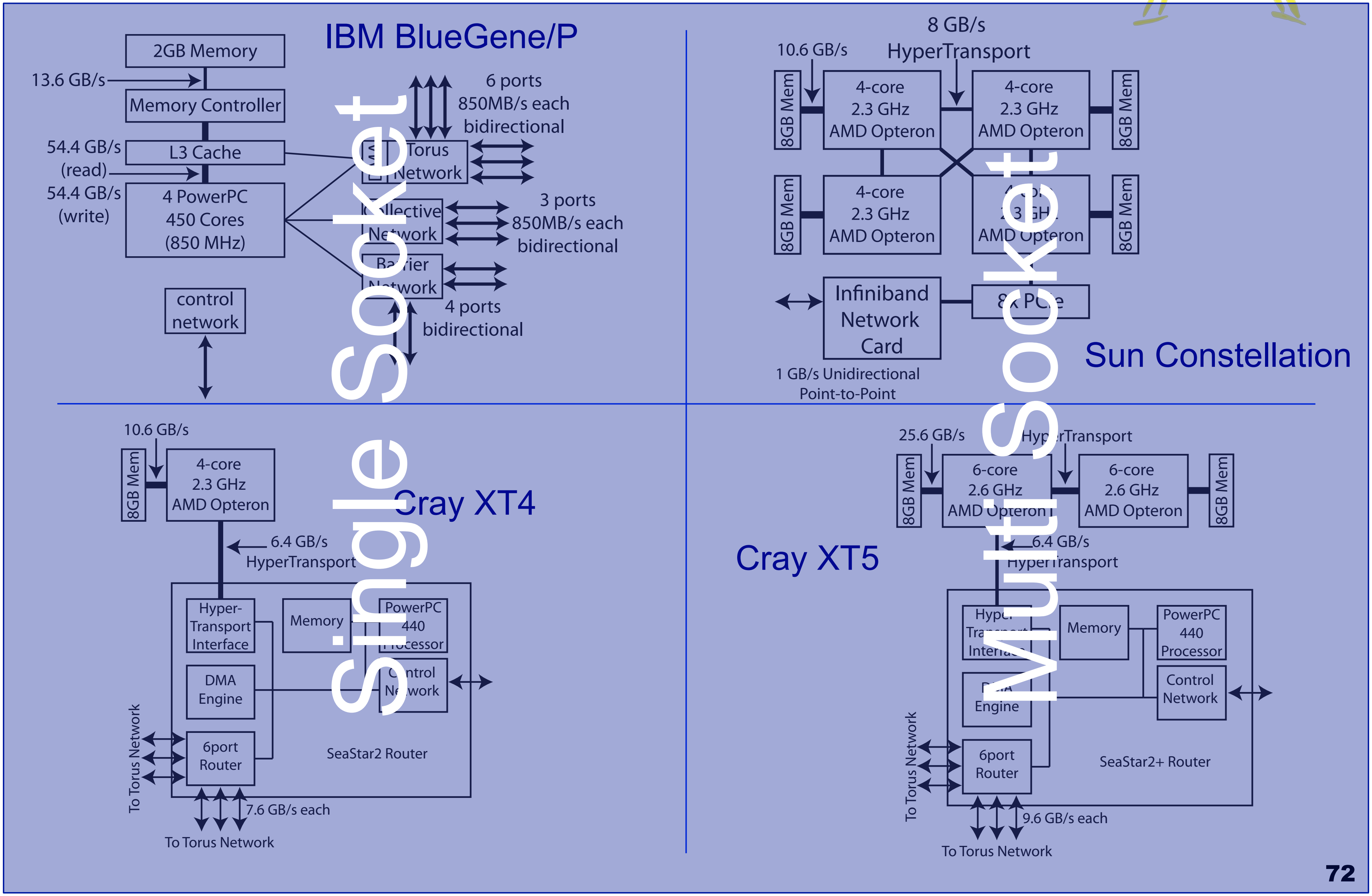
Intel Clovertown (8 threads)



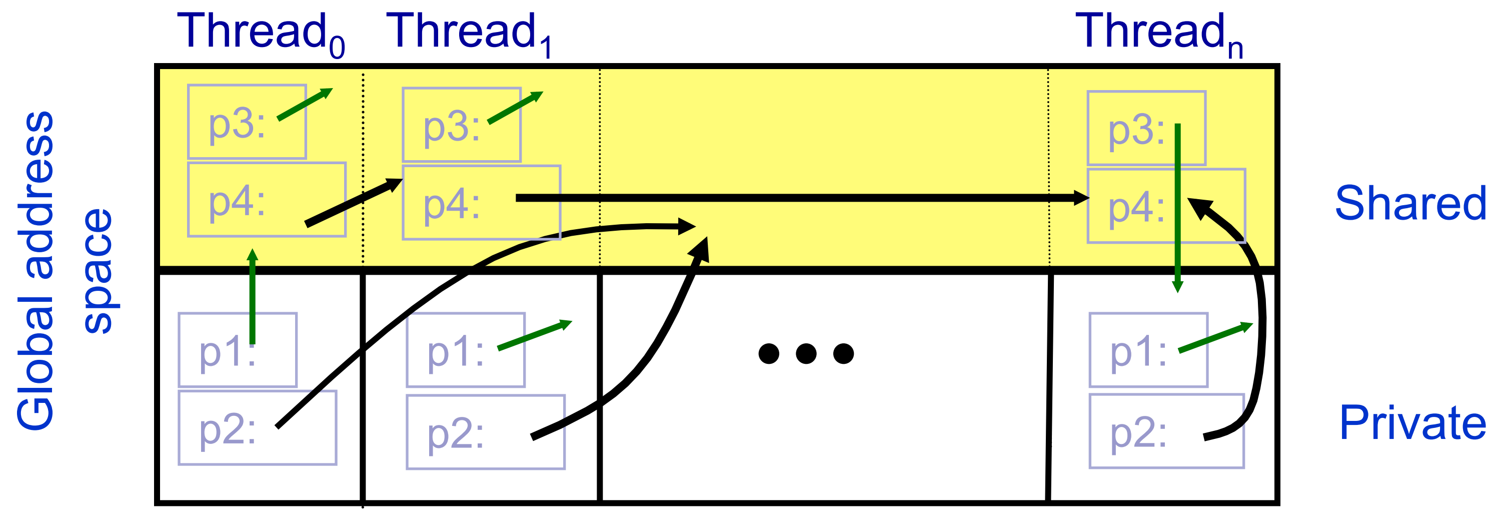
Intel Nehalem (16 threads)

[Diagrams Courtesy of Sam W. Williams]

Node Architectures



UPC Pointers

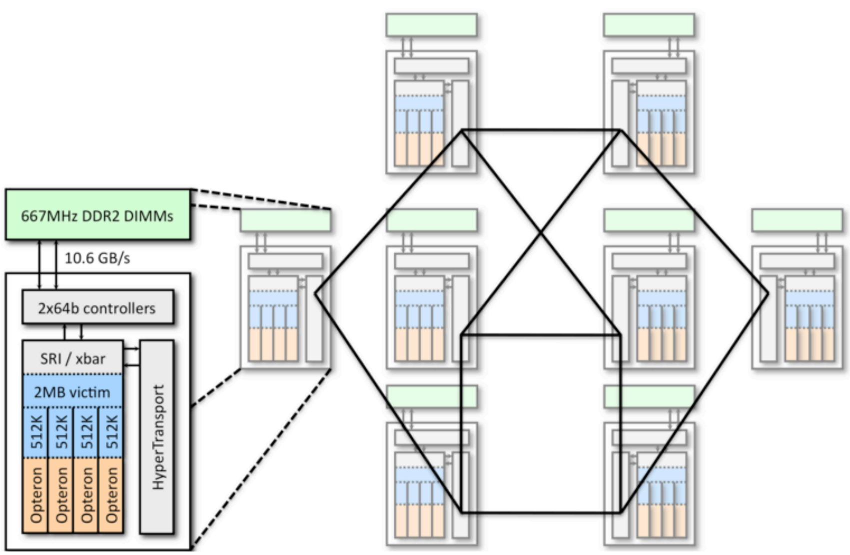


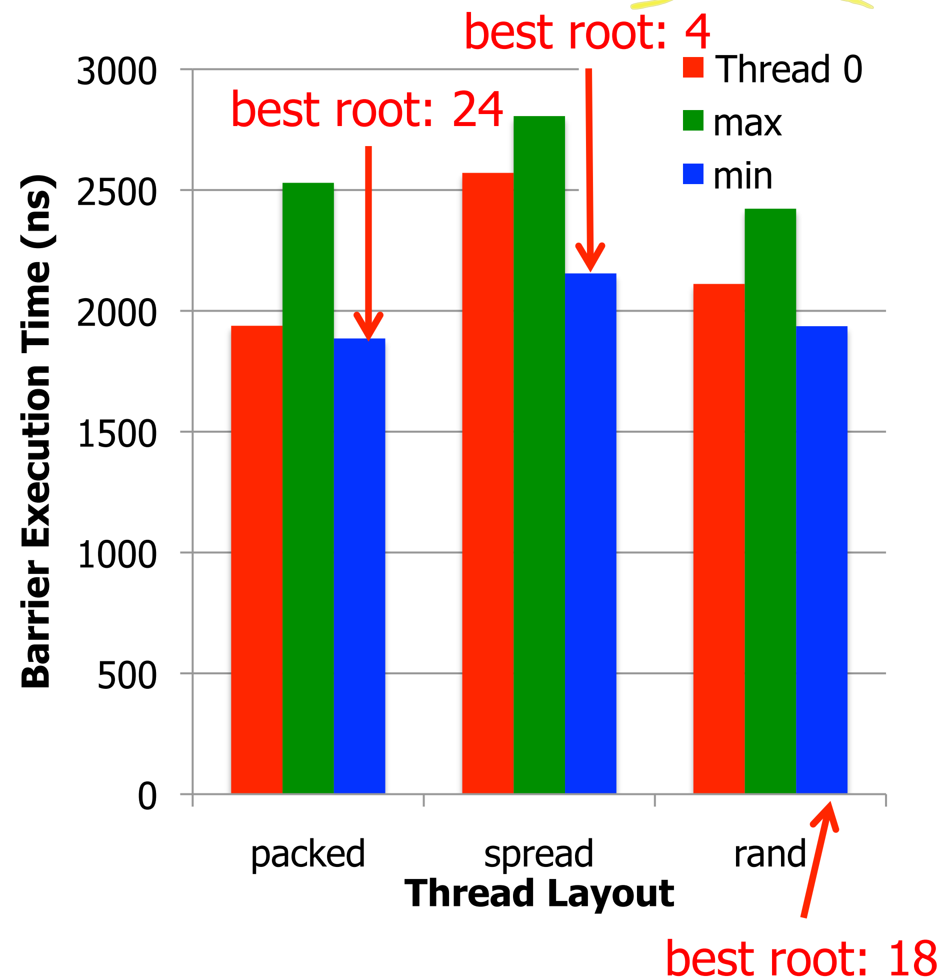

```
int *p1;            /* private pointer to local memory */
shared int *p2; /* private pointer to shared space */
int *shared p3; /* shared pointer to local memory */
shared int  *shared p4; /* shared pointer to
                          shared space */
```

Pointers to shared often require more storage and are more costly to dereference; they may refer to local or remote memory.

Barrier Tuning Parameters

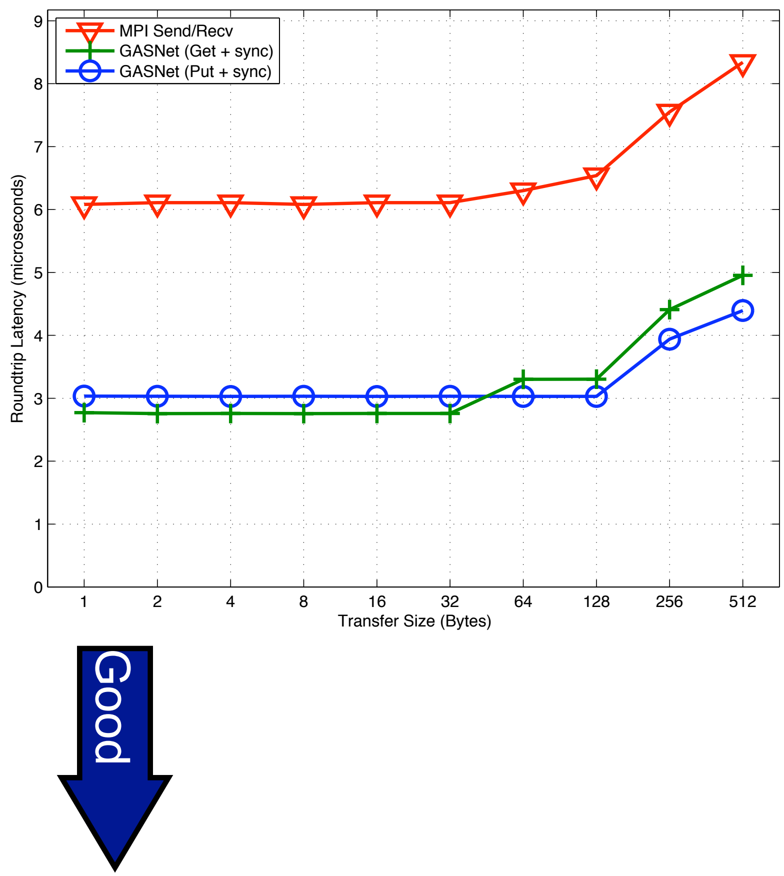
- Algorithm
- Signaling Mechanisms
- Tree Geometry
  - Tree Root
  - Tree Shape





AMD Opteron (32 threads)
Barrier Performance (varying root)

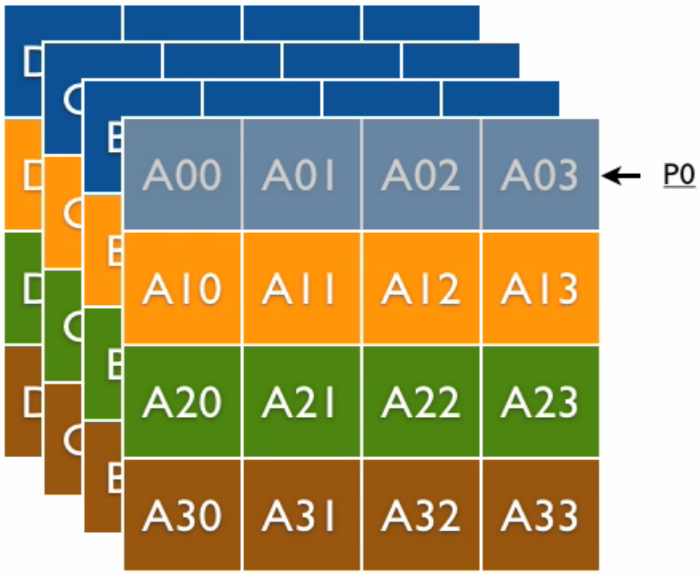
GASNet Latency Performance



- GASNet implemented on top of Deep Computing Messaging Framework (DCMF)
  - Lower level than MPI
  - Provides Puts, Gets, AMSend, and Collectives
- Point-to-point ping-ack latency performance
  - N-byte transfer w/ 0 byte acknowledgement
    - GASNet takes advantage of DCMF remote completion notification
  - Minimum semantics needed to implement the UPC memory model
  - Almost a factor of two difference until 32 bytes
  - Indication of better semantic match to underlying communication system

FFT Transpose

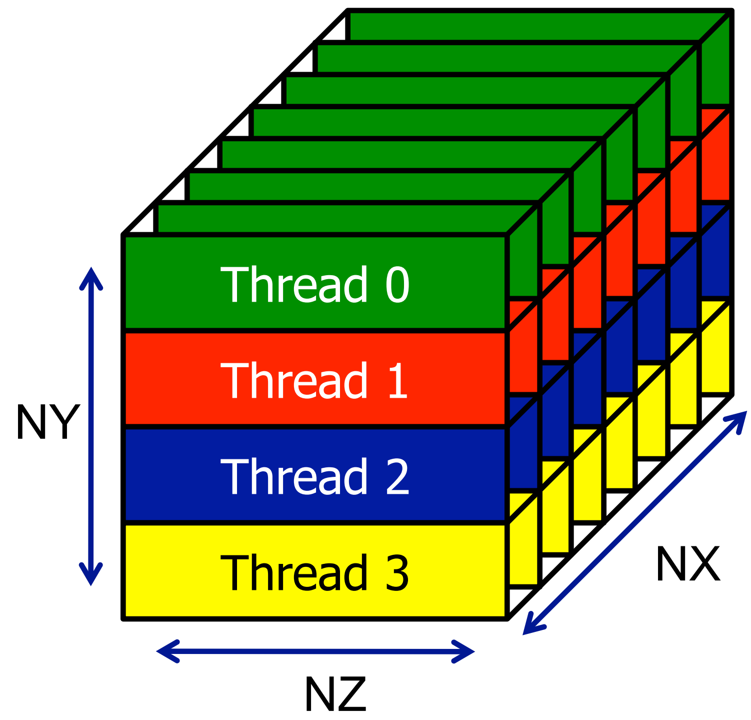
- Two transposes that exchange the entire domain
  - Stresses the bisection bandwidth of the network
  - On many machines communication costs are on par w/ computation costs
- Conventional wisdom is to pack messages to maximize message sizes and achieve peak bandwidth
  - Is that really the best though?



Each processor owns a row of 4 squares (16 processors in example)

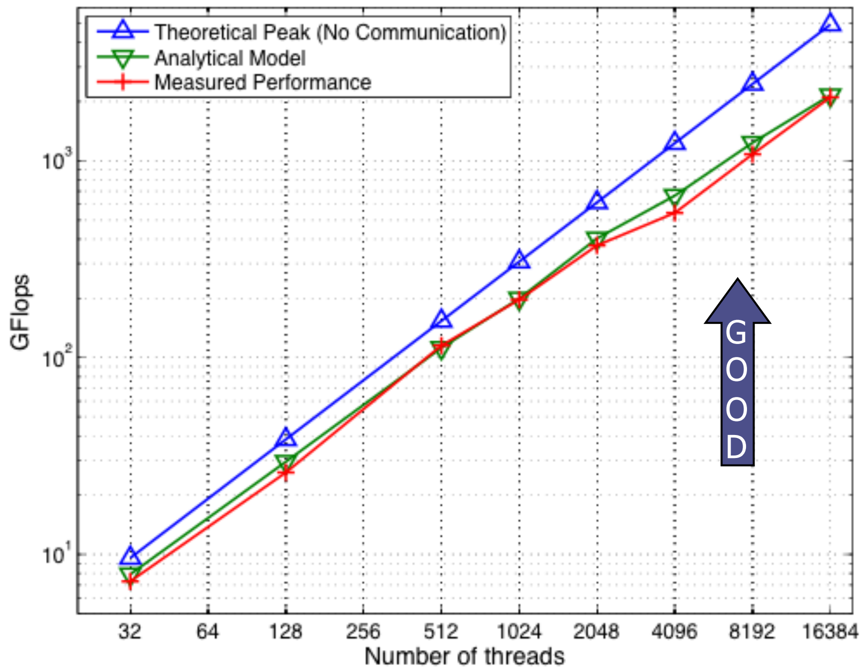
A Motivation for Teams: 3DFFT

- Many applications require collectives to be performed across teams (*i.e.* subsets) of the threads
- Example 3D FFT:
  - Cube is distributed
  - Each processor owns a rectangle (slab)
  - Bandwidth limited problem
- FFTs performed in each dimension
  - 1st FFT is local
  - 2nd FFT requires exchange amongst threads that share a plane
  - 3rd FFT requires exchange amongst row of slabs (same color)



Interface To Collectives

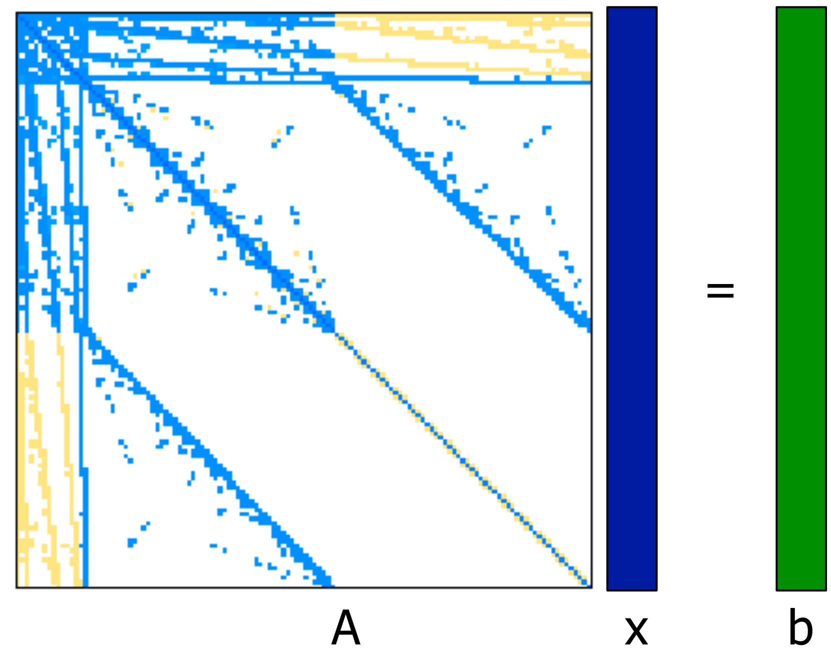
- How do we construct these teams?
  - Thread-Centric: Programmer explicitly specifies the threads that take part in the collective through a language level team construction API
  - Data-Centric: Programmer only specifies the data for the collective. Runtime system then figures out where the data resides and performs the collective
- How do we incorporate these interfaces with the autotuners?



- Wrote 3D FFT w/ Data-centric primitives
  - Ran on BG/L to analyze limits of scalability of interface
- Interface doesn't limit scalability
- 2 Teraflops across 16k threads

Auto-tuned Conjugate Gradient

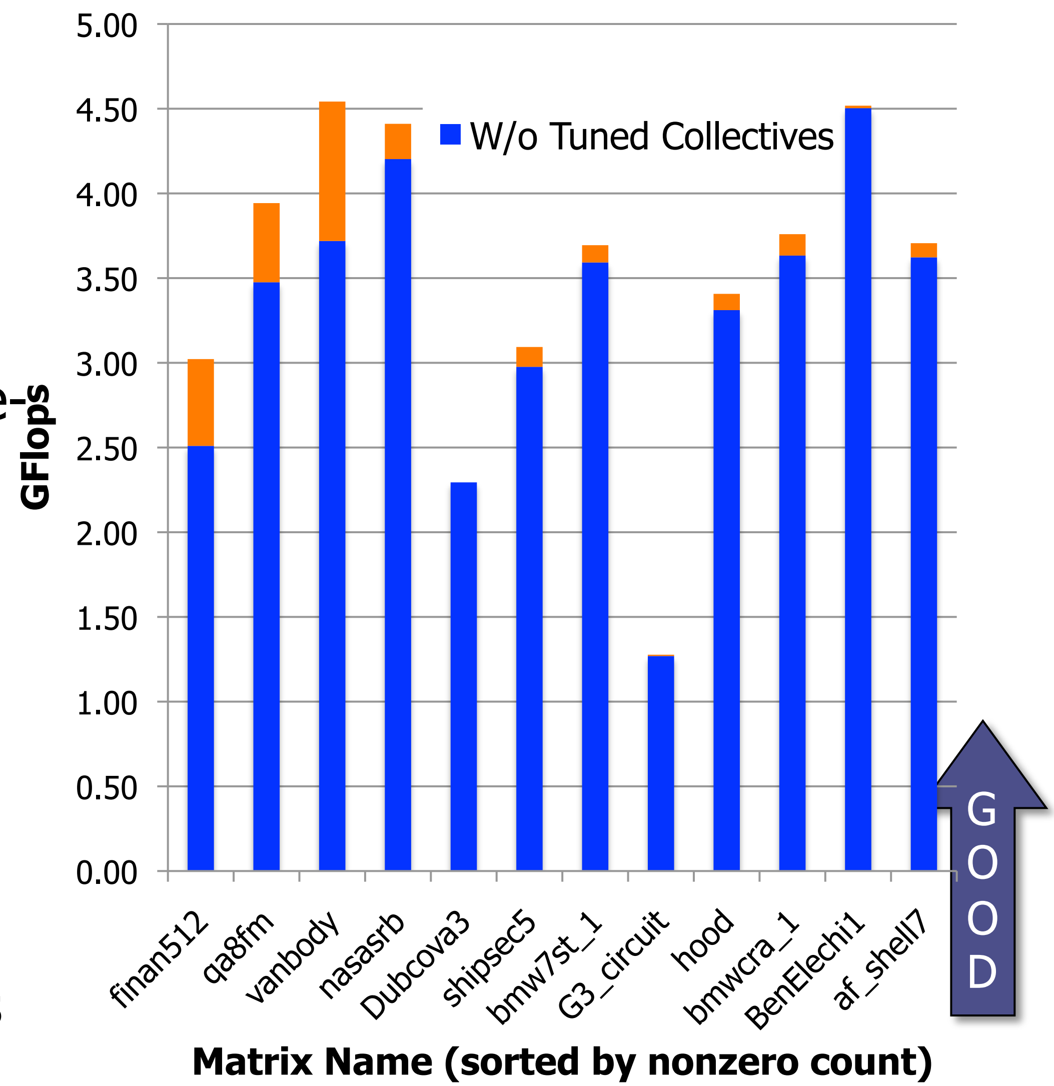
- Incorporate tuned collectives into an important kernel
- Sparse Conjugate Gradient
  - Part of Sparse Motif
  - Iteratively solve Ax=b for x given A and b
  - Relies heavily on optimized SPMV and tuned BLAS1 operations
  - Matrix Partitioned Row-wise for our application
- Automatic tuning for a parallel system
  - Kernels tuned for parallel *and* serial performance
  - Previous related work have focused on serial tuning only



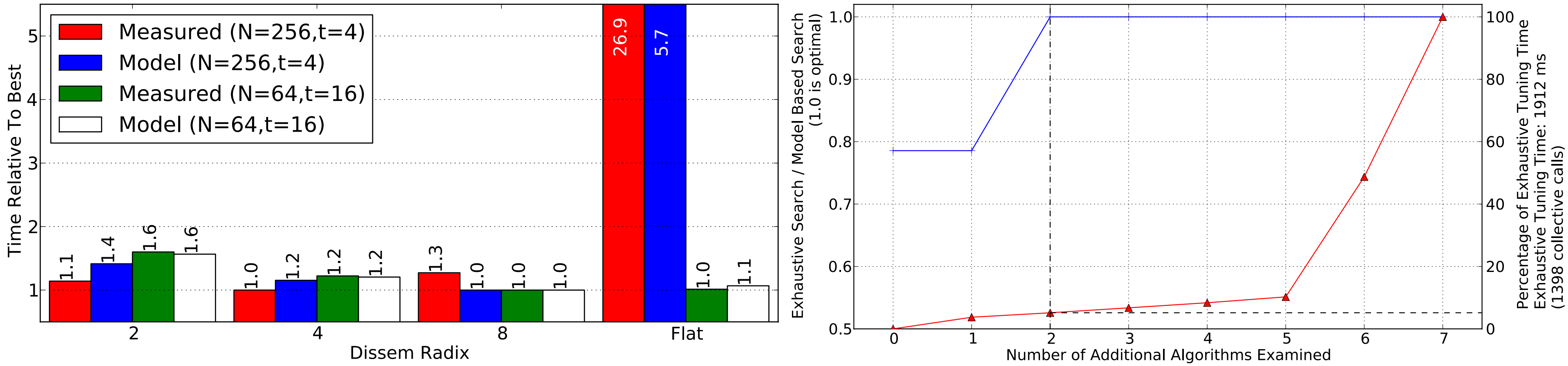
- Collectives Used:
  - Scalar Reduce-To-All for Dot Products
  - Barriers

Conjugate Gradient Performance

- Auto-tuned SPMV from Sam Williams [Williams et. al, SC'07]
- Sun Performance Library for local BLAS1 operations
- Incorporate aforementioned tuned barrier and tuned Reduce-to-All for inter-thread communication
- Matrix parallelized row-wise
  - reductions are performed across all 128 threads
- Best Speedup: 21%
- Median Speedup: 3%
- Auto-tuning took a few seconds to search for best barrier and best



Performance Model: Exchange



8 byte Exchange on Sun Constellation (1024 cores)

- Optimal algorithm also depends on the number of threads per node
  - For 4 threads per node model predicts radix 8 is the best
  - With 16 threads per node this however takes 1.4 times as long as the flat algorithm
  - Using flat algorithm for 4 threads per node also leads to severe penalties
- Model accurately predicts best performer in both cases