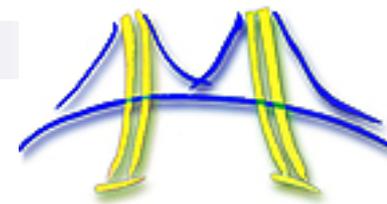


Autotuning Collective Operations in a Multicore Environment

Rajesh Nishtala, Paul Hargrove, Yili
Zheng, and Kathy Yelick

Berkeley UPC Group

Collectives Examples



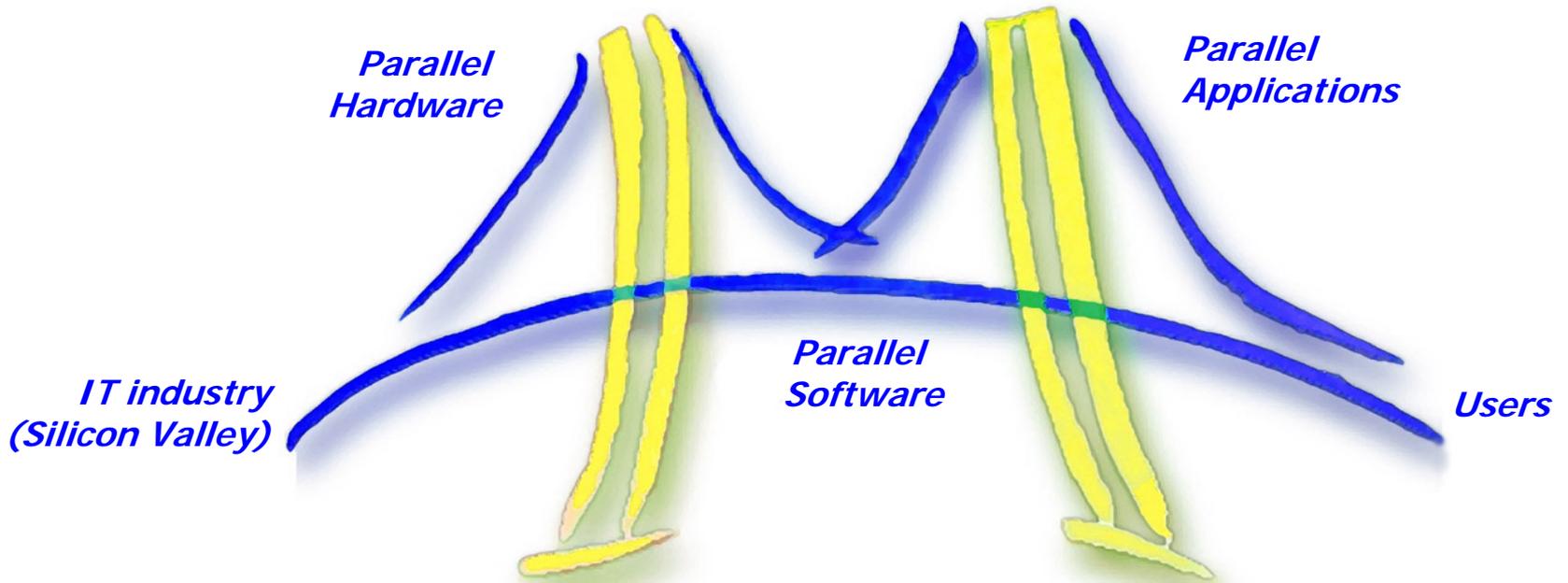
- Operations that perform globally coordinated communication
- Most modern parallel programming libraries and languages have versions of these operations

One-to-Many

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

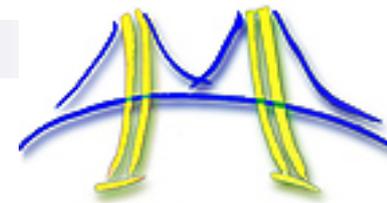
Many-to-Many

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

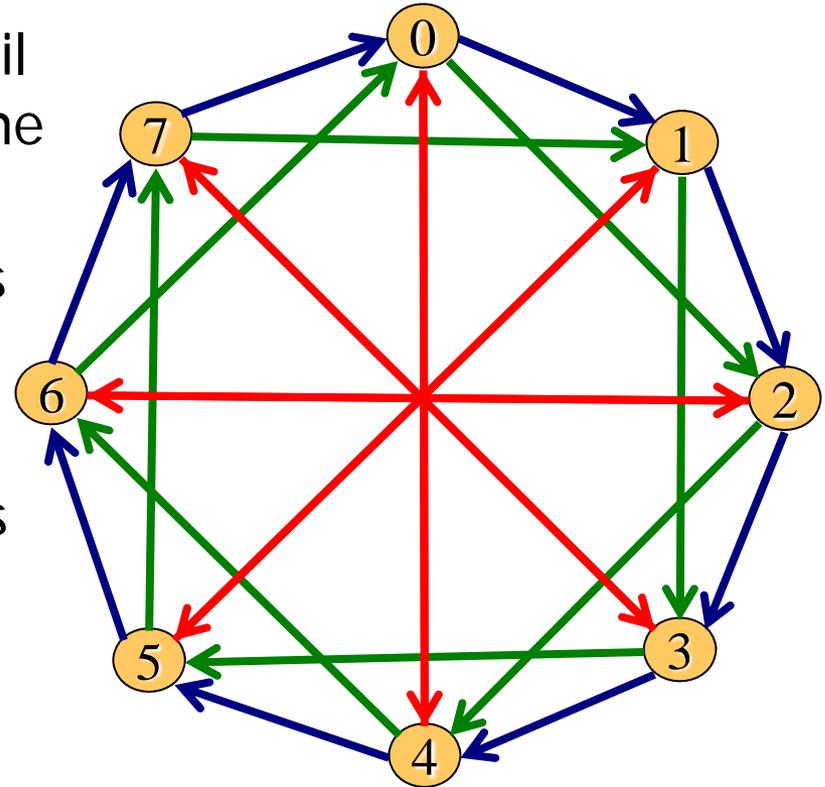


Shared Memory Collectives

Barrier (dissemination algorithm)

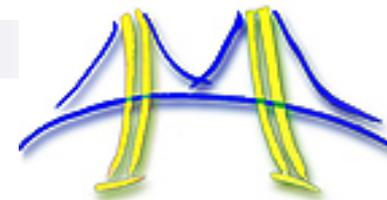


- Synchronization Construct
 - Can't return from a Barrier until all other threads have called the Barrier
- Complete Barrier in $\log_2(T)$ stages
- Each stage we learn about twice the number of processors
- Dissemination required all threads to be active all the time
 - $O(T \log T)$ "messages"
 - Time: $L * (\log T)$ ($L = \text{latency}$)

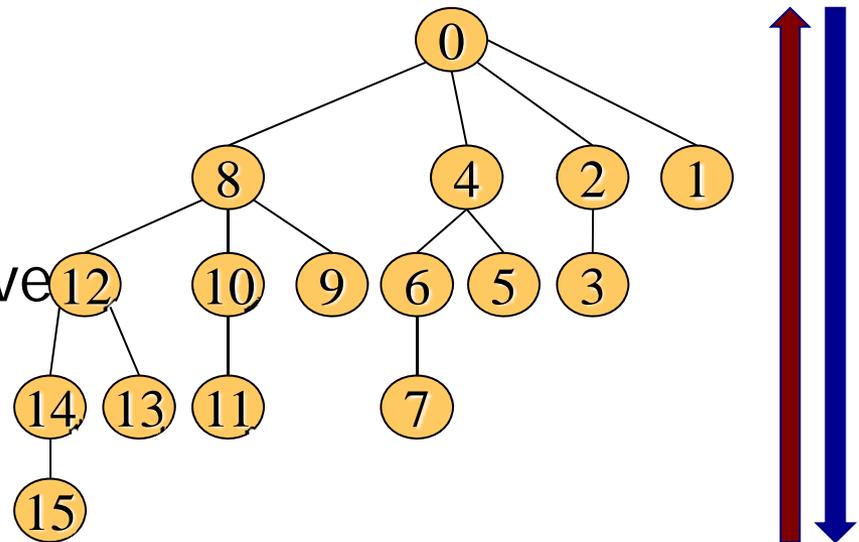


View from Thread 0	T0	T1	T2	T3	T4	T5	T6	T7
Who knows about T0	✓	✓	✓	✓	✓	✓	✓	✓
Who T0 knows about	✓	✓	✓	✓	✓	✓	✓	✓

Barrier (tree algorithm)

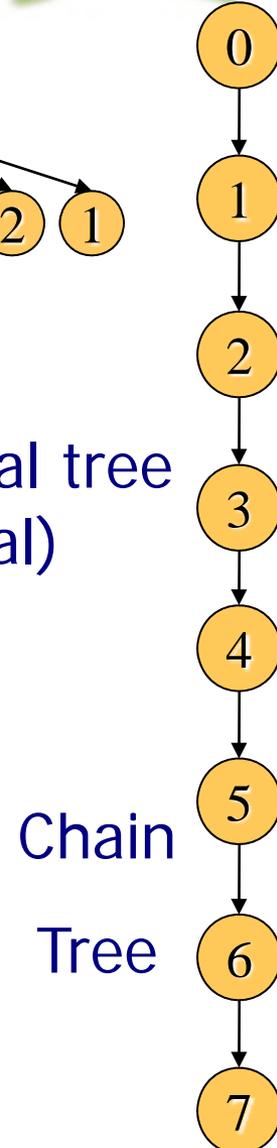
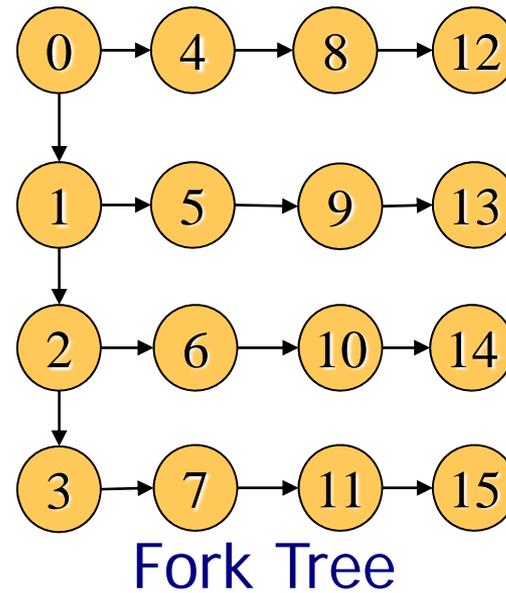
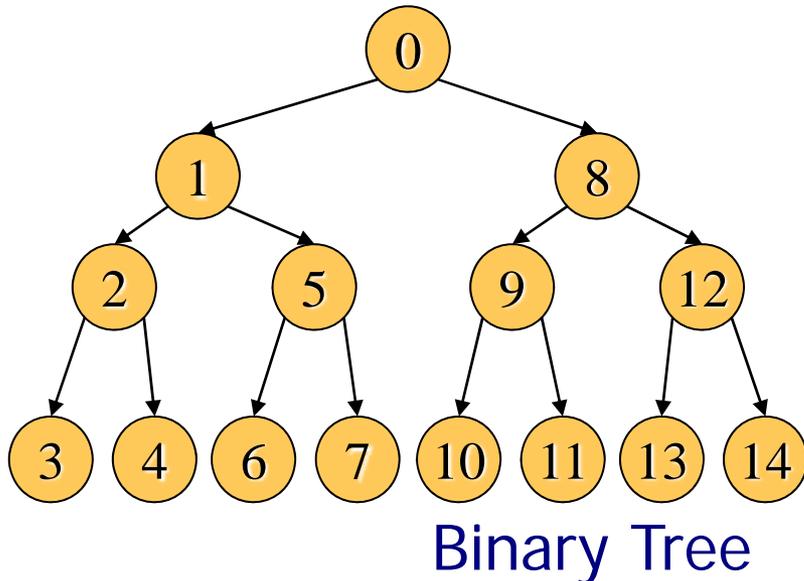
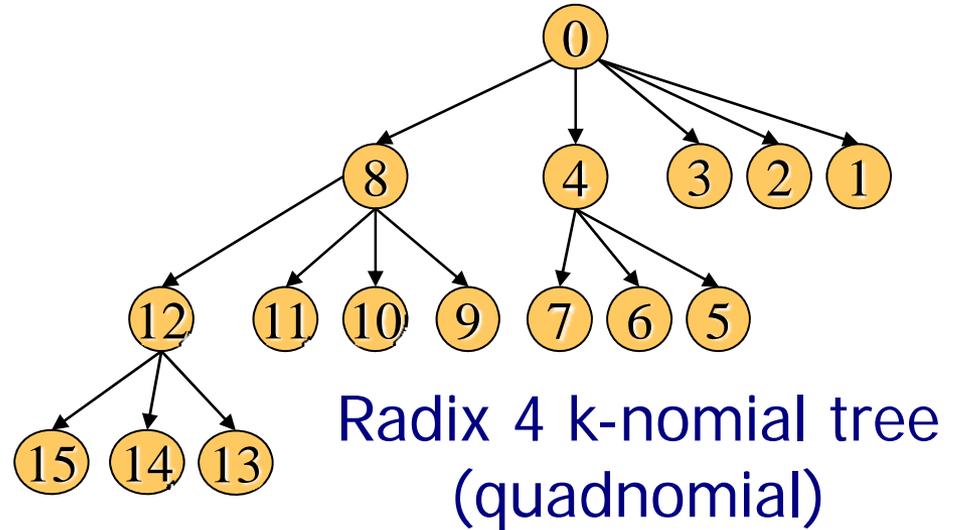
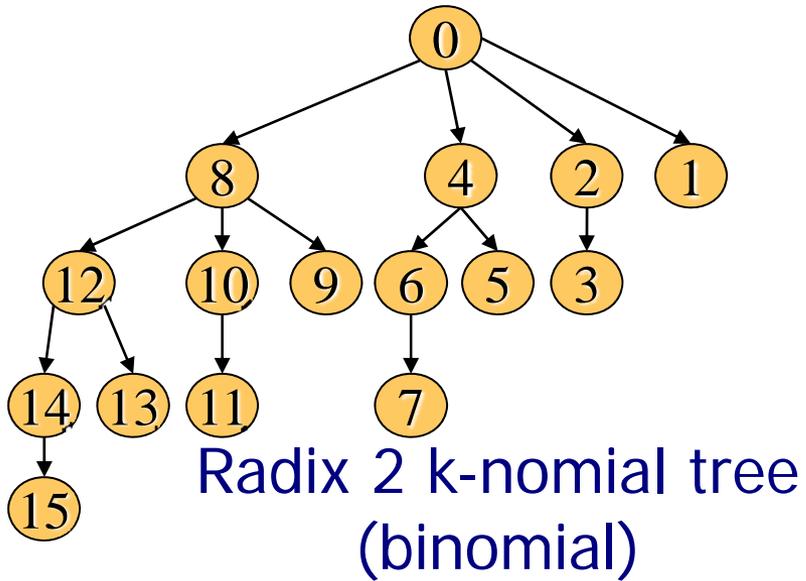
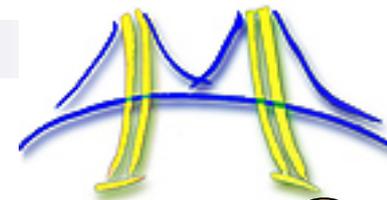


- Requires two passes of a tree
 - First (**UP**) pass tells parent subtree has arrived.
 - Second (**DOWN**) pass indicates that all threads have arrived
 - $O(T)$ "messages"
 - Time: $2L * (\log T)$
- Two ways to signal others:
 - Push: write a remote variable and spin wait on a local variable
 - Pull: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

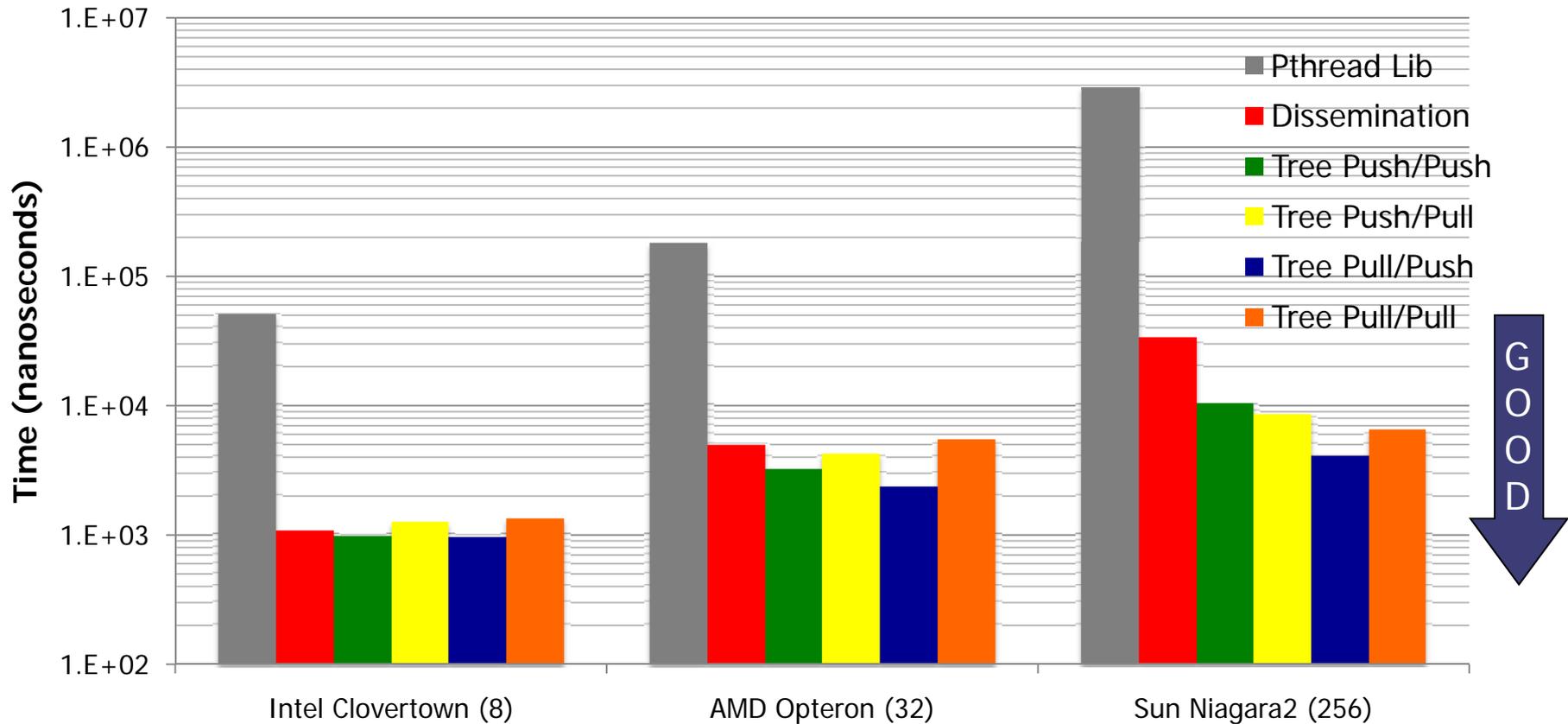
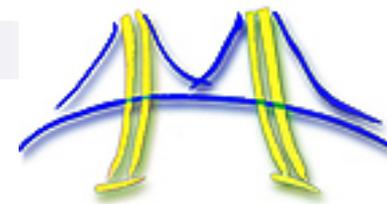


- Leads to 4 unique tree algorithms
- Performance of each is dependent on how systems handle coherency and atomic ops

Example Tree Topologies



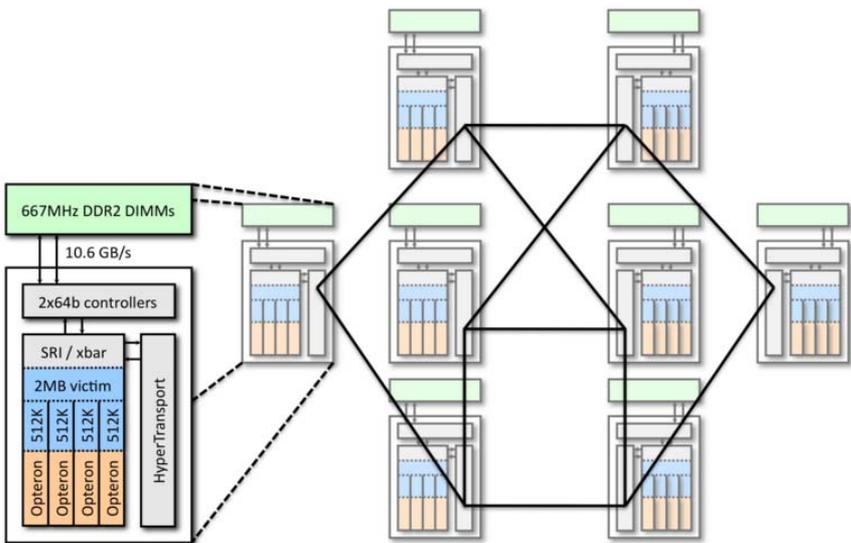
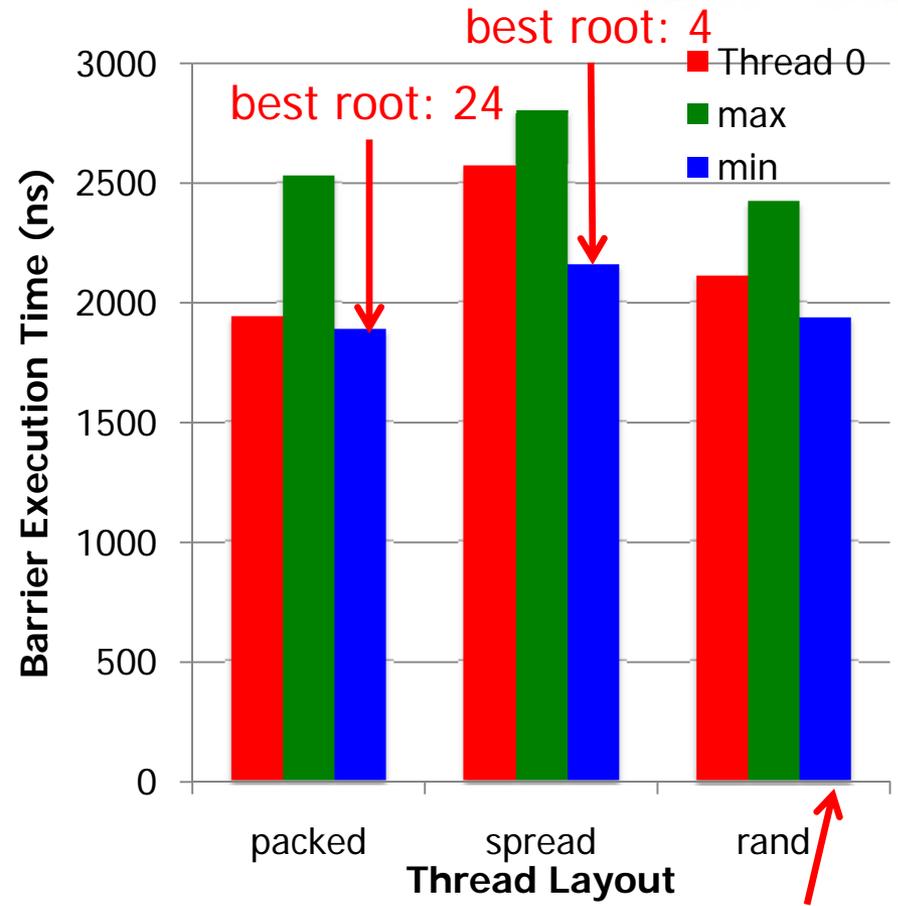
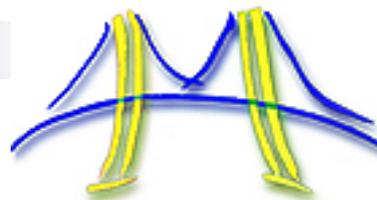
Barrier Performance Results



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

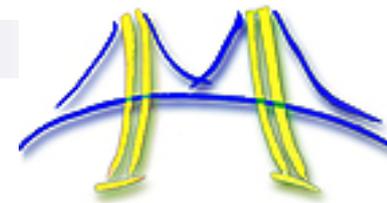
Barrier Tuning Parameters

- Algorithm
- Signaling Mechanisms
 - (Previous Slide)
- Tree Geometry
 - Tree Root
 - Tree Shape

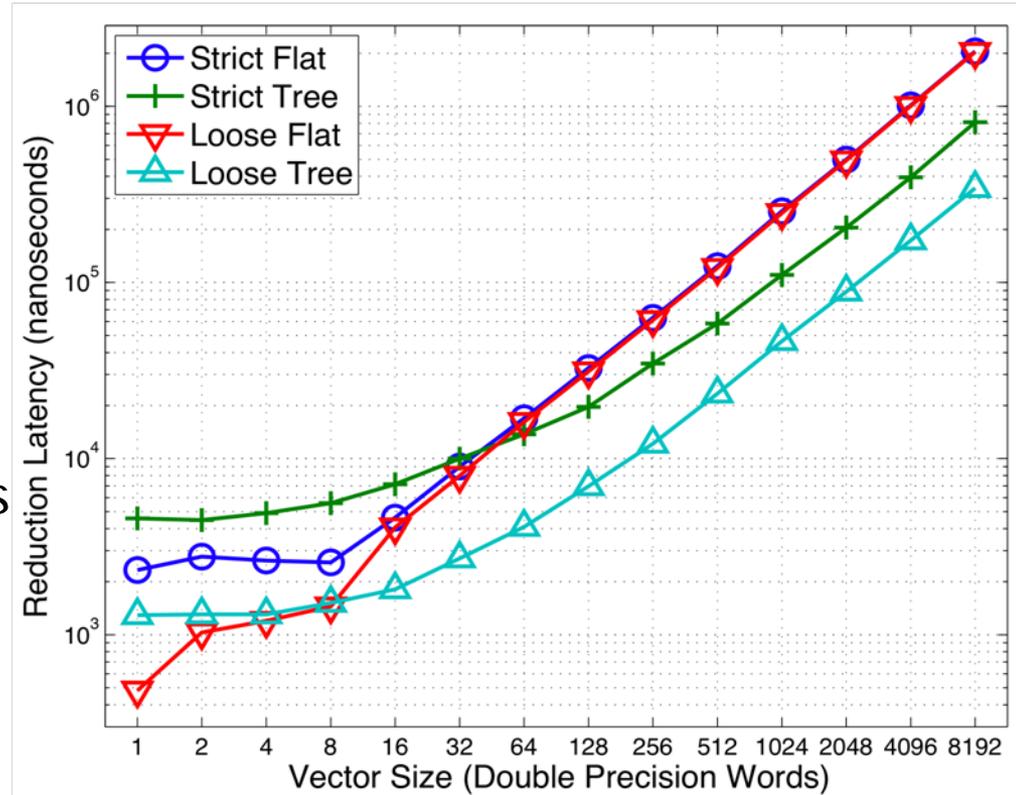


best root: 18
 AMD Opteron (32 threads)
 Barrier Performance (varying root)

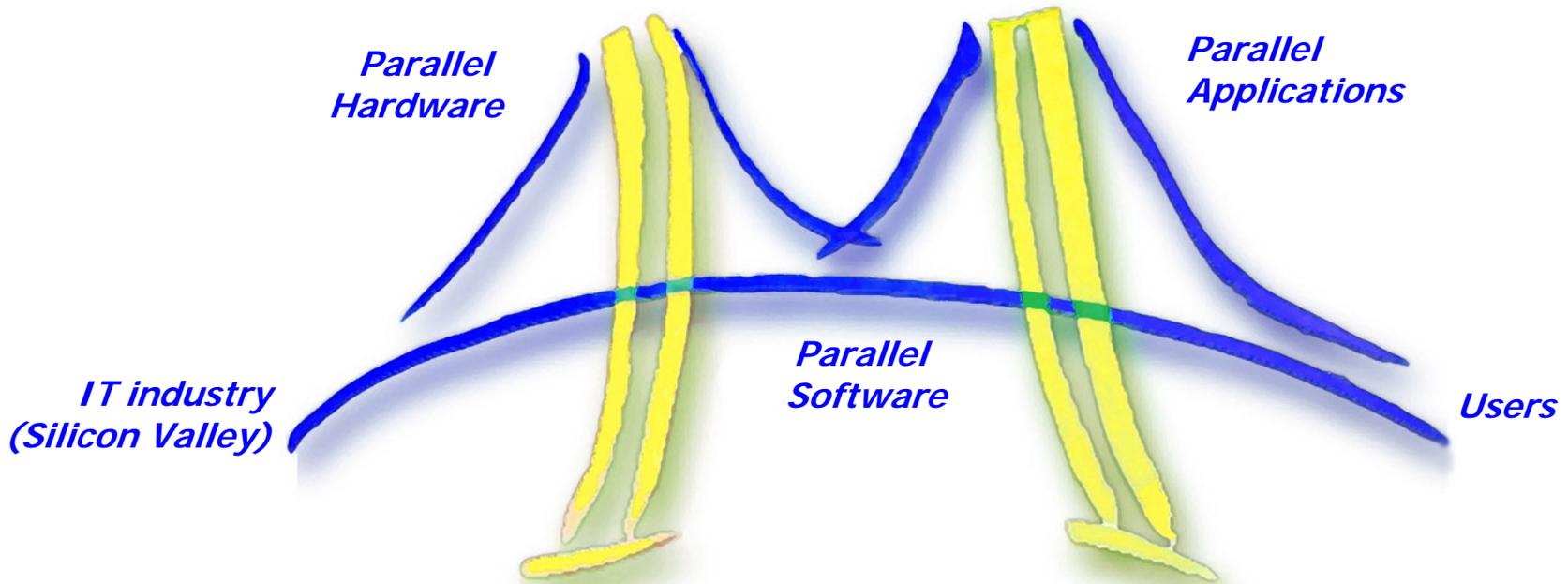
Autotuning and Synchronization



- Tradeoff between Flat and Tree based topology exposes cost of synchronization vs. benefit of extra parallelism
- Optimal algorithm choice is affected by the synchronization flags
- Looser Synchronization enables trees to realize better performance at lower message sizes.

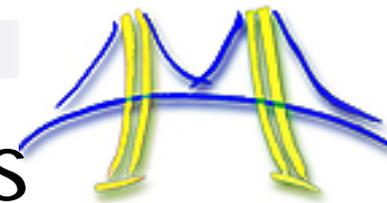


AMD Opteron (32 threads)
Reduction Performance



Collectives for Clusters of Multicore Processors

Design Goals for GASNet Collectives



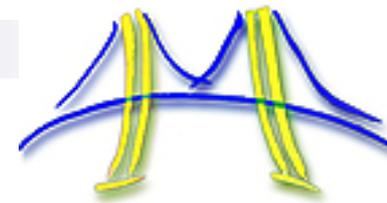
□ Interface

- 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)

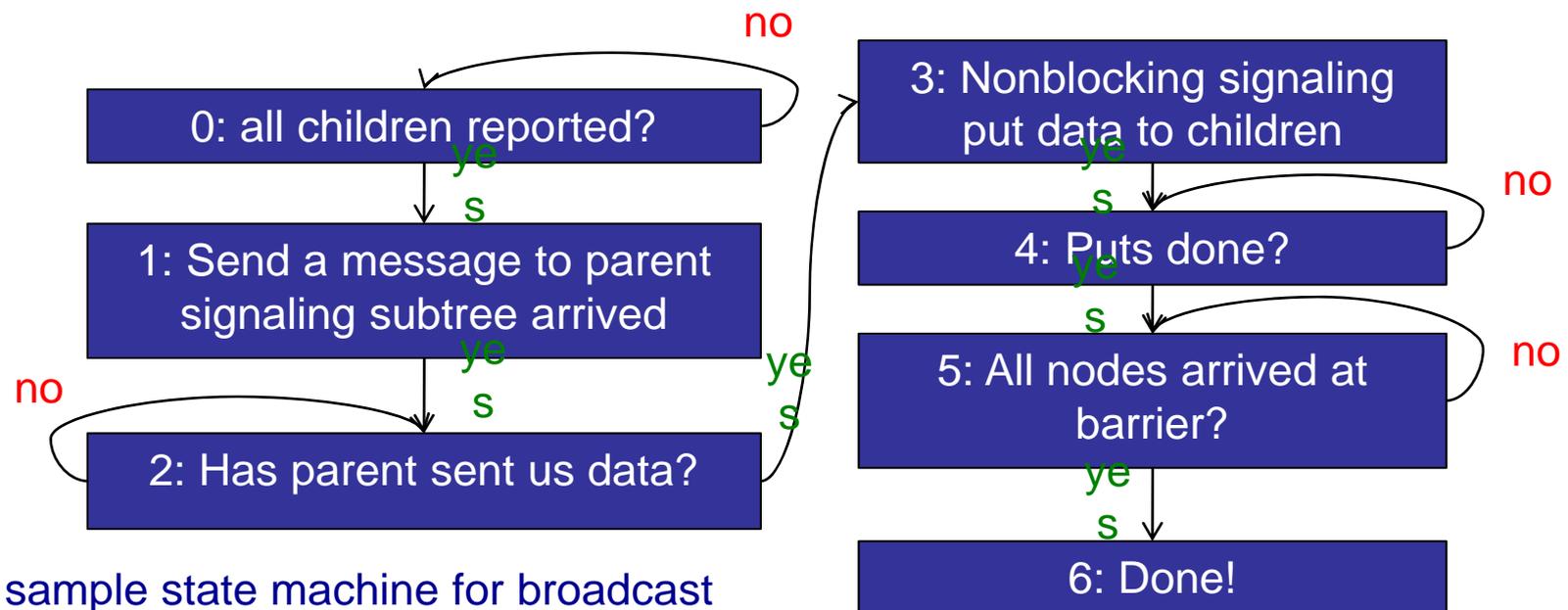
□ Implementation

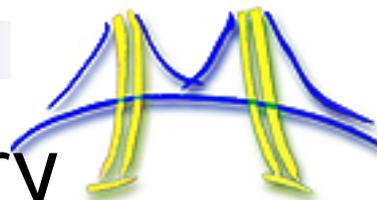
- Leverage shared memory whenever it's available
- Collectives are automatically tuned
 - Infrastructure should be able to include hardware collectives on platforms where applicable

Nonblocking Collectives



- Uses state machines to control when certain actions can be taken
- Collective initiation creates a state machine and puts it on a “runnable” queue
- Instead of spin waiting at a state function returns control to poller
 - `gasnet_coll_trysync()` polls all active collectives once

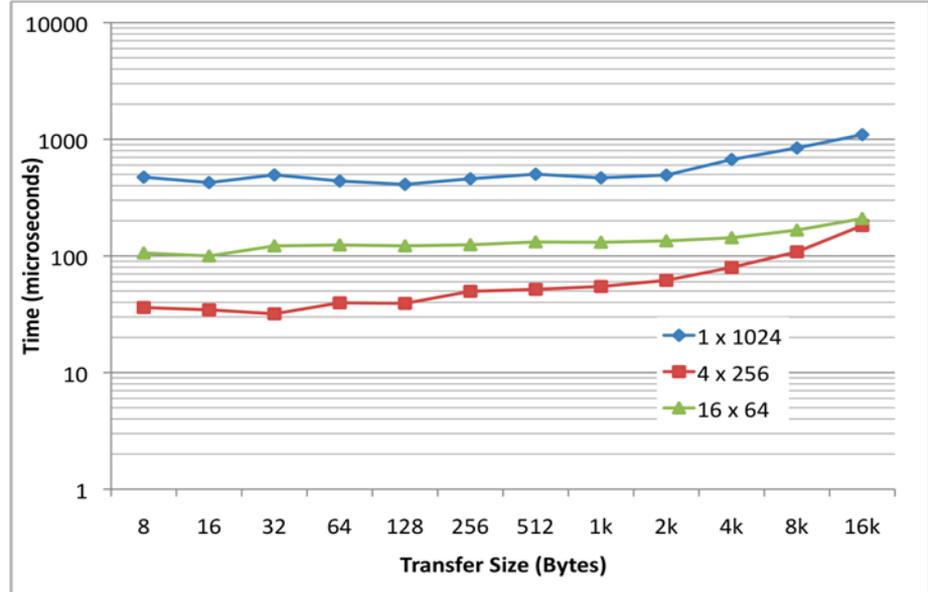




Taking Advantage of Shared Memory

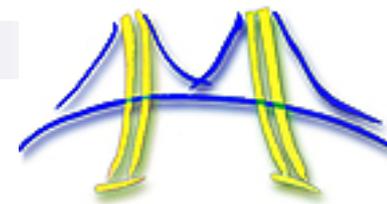
- Use one representative thread per node to handle communication
 - Minimize contention of communication resources
- Perform memcpy to pack and unpack data for local collective
 - Increases size of messages on the wire
 - Incurs overhead for packing and unpacking data

Broadcast on Ranger
(varying threads per node)



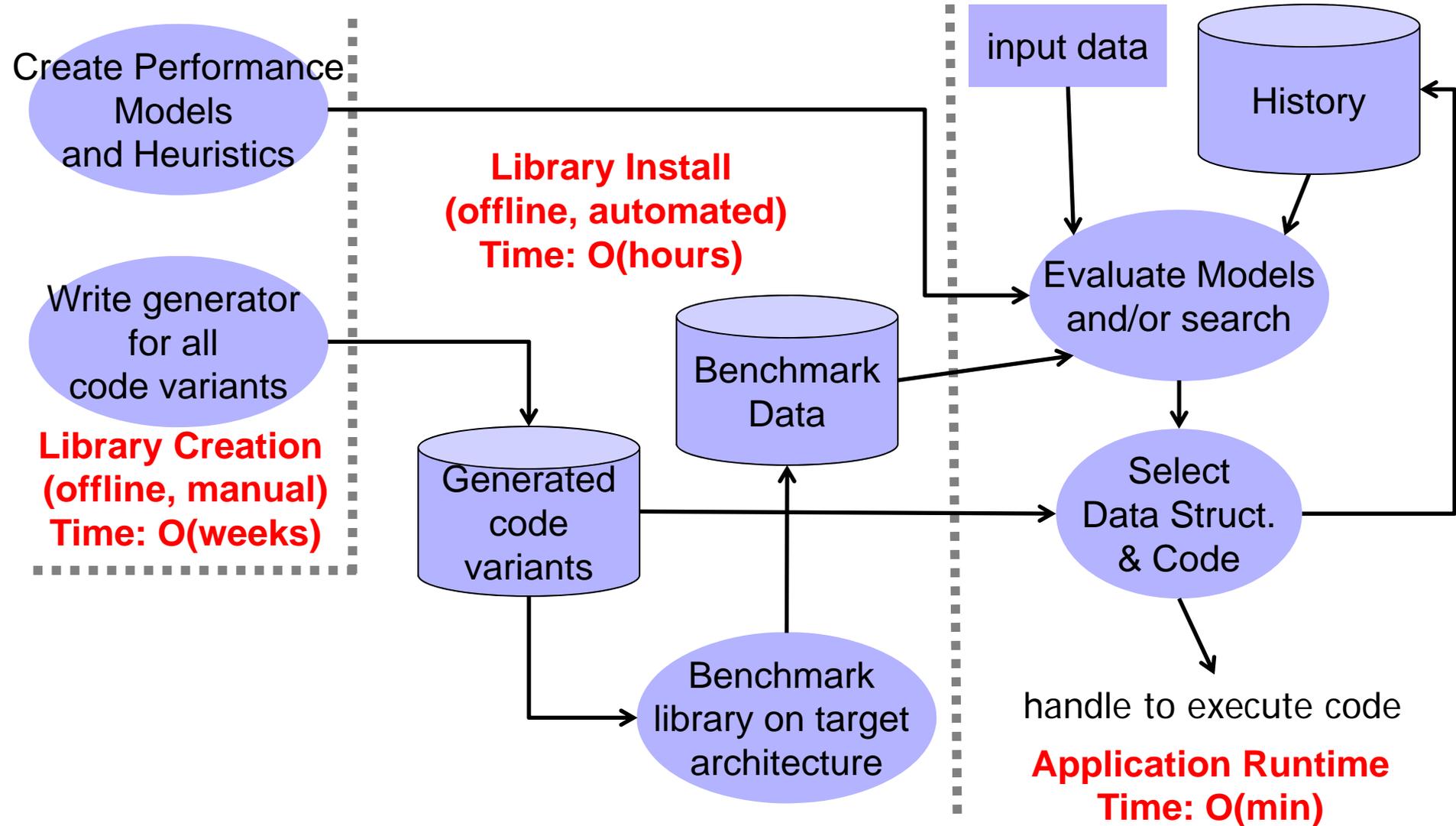
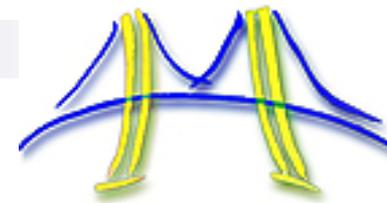
- 4 threads per process consistently yields best performance
 - One process per socket
 - Allows both network cards to be used effectively

Other Infrastructure Features



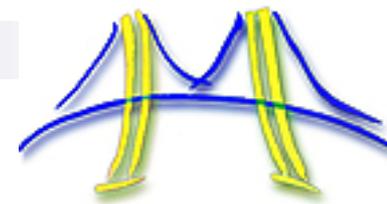
- Distributed Scratch Space Management
 - Some optimal implementations of collectives require extra storage not visible to user
 - Need to manage the auxiliary space in a scalable and distributed way
- Tree Construction
 - Optimal communication schedule is determined by network features
 - Native support for k-nary, k-nomial, and fork of various fanouts
 - Trees can be composed in arbitrary ways to produce more interesting communication topologies that best suit the network

Autotuner Flowchart



[Based on a slide from Vuduc, SciDAC'05 meeting]

Automatic Tuning Overview (1/2)

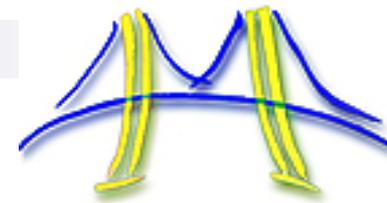


□ Portable Performance

- Many factors that influence the optimal algorithm
- Importance of different factors depend on the target platform

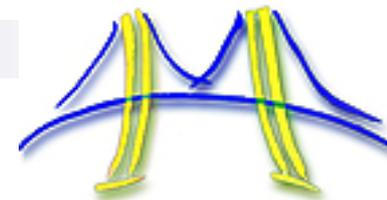
INSTALL-TIME	RUN-TIME
<ul style="list-style-type: none">• Processor type/speed• Memory system• Number of cores per socket• Number of network cards• Interconnect Latency• Interconnect Bandwidth• Interconnect Topology	<ul style="list-style-type: none">• Processor connectivity• Number of processors• Sizes of the messages• Synchronization mode• Network load• Mix of collectives and computation

Automatic Tuning Overview (2/2)



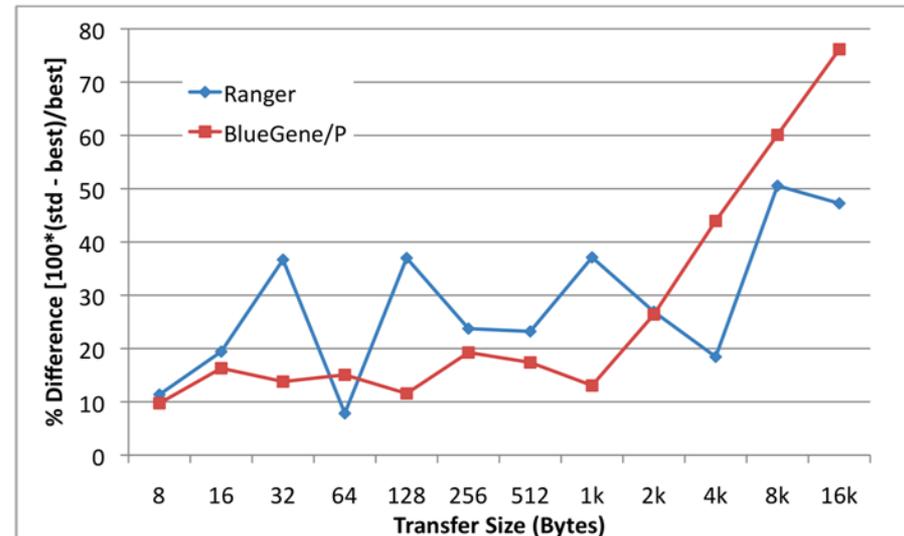
- Each collective have many implementations in GASNet
 - Variants such as eager, rendezvous, direct put, direct get
 - Orthogonally, there are many possible trees that we can use
- GASNet collective infrastructure indexes all the algorithms
 - Hardware collectives for certain conduits go into this index
 - Special tester runs through all possible input combinations and a subset of the (many) possible algorithms to pick the best algorithm.
 - Like FFTW and other automatic tuning projects, the automatic tuning data is saved across runs

Importance of Tuning

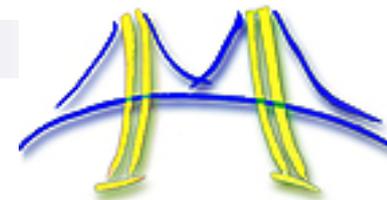


- Compare best algorithm v. a “standard” algorithm used to implement the collectives
 - Importance of tuning varies based on platform and message size
- Same code base runs on both BlueGene/P and Ranger
- Data also show best algorithm changes at 16k bytes on Ranger
 - From Direct Put to Rendez-Vous Get
- Much wider range of optimal trees
 - Varies from Nary to Recursive trees
 - Binary Tree best on BG/P

Broadcast Speedup



Conclusions



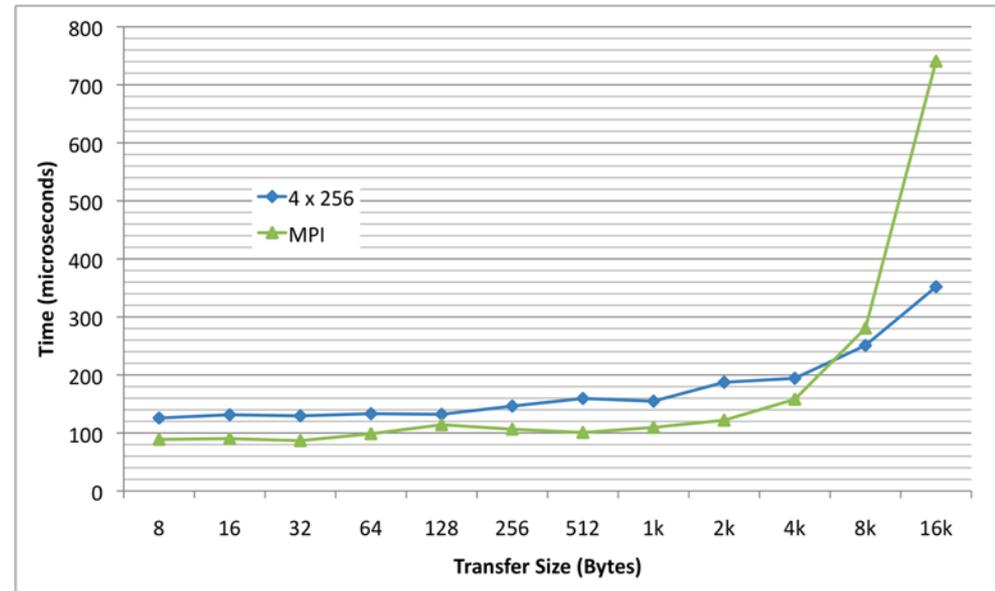
□ Significant progress on GASNet Collectives

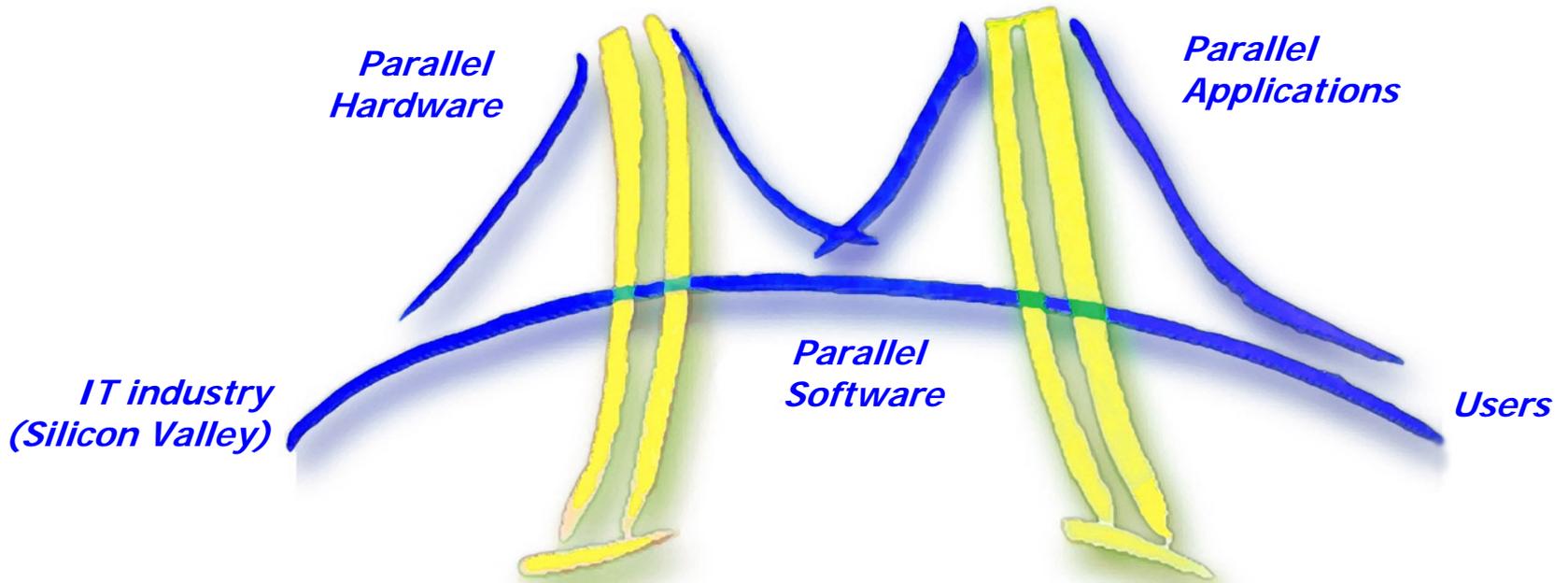
- Added Teams
- Added automatic tuning infrastructure and tests
- Many performance bug fixes
- Test collectives and infrastructure at large scale

□ Future Work

- Continue to build out automatic tuner
- Explore more hardware collectives
- Add more algorithms to search space

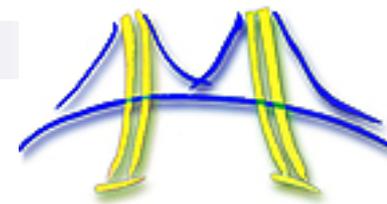
Comparison of MPI and GASNet Broadcast





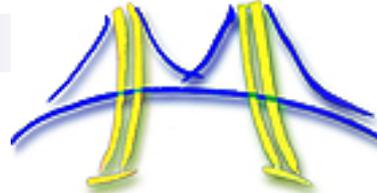
Backup Slides

Case Study: Barrier

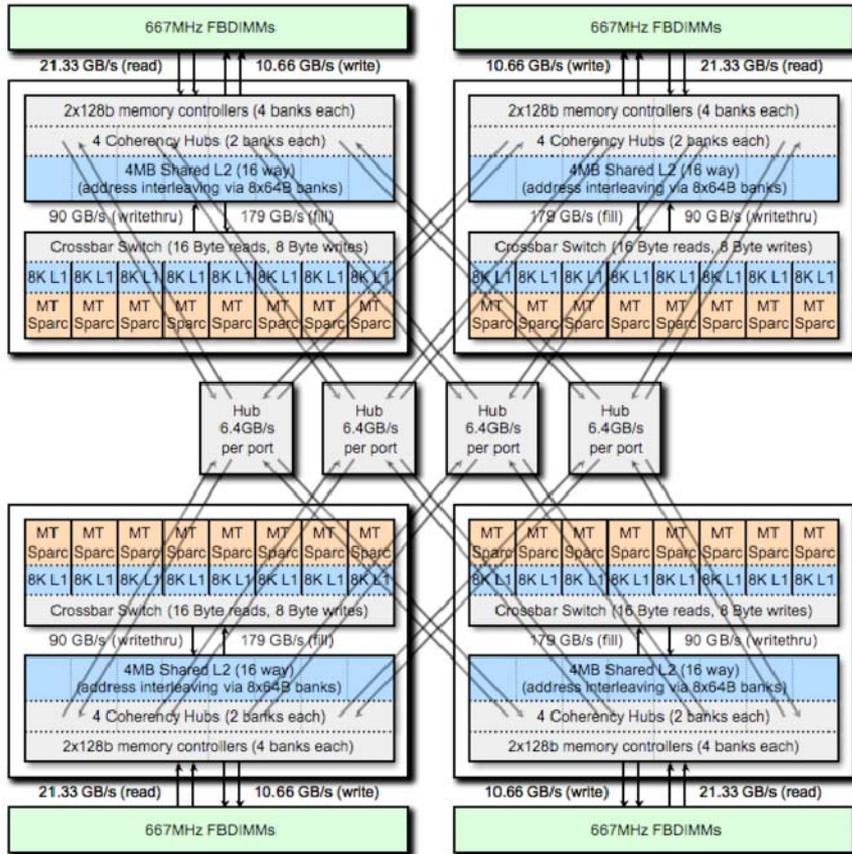


- This talk outlines Barrier as a case study
 - Our software automatically tunes all the aforementioned collectives
- Synchronization Construct
 - Can't return from a Barrier until all other threads have called the Barrier
- Why does a fast barrier help us?
 - Synchronous programs are a lot easier to understand and debug than their asynchronous counterparts
 - If the barriers separating different phases are slow, Amdahl's law limits benefits from parallelism
 - Faster barriers enable finer-grained parallelism without resorting to asynchronous, and error prone, code

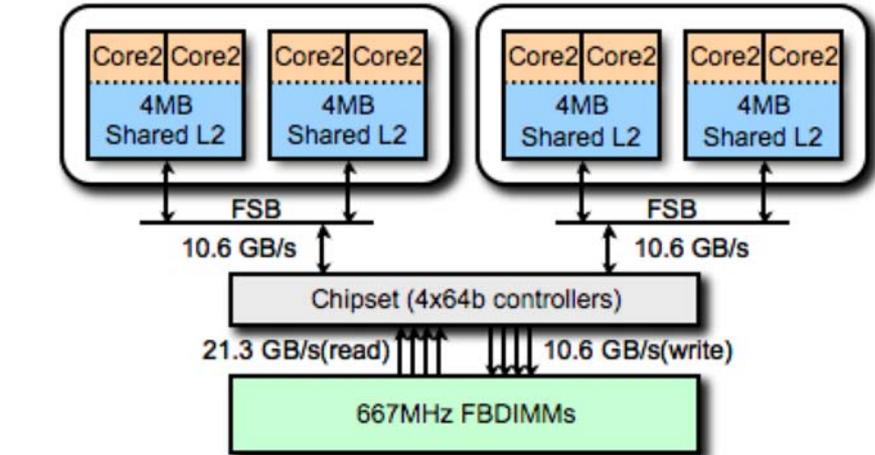
Experimental Platforms



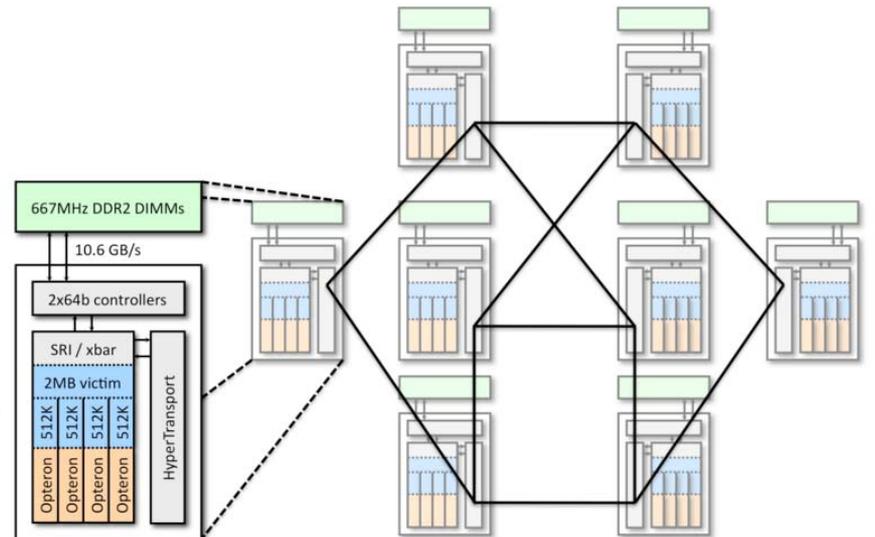
Sun Niagara2 (256 threads)



AMD Opteron (32 threads)

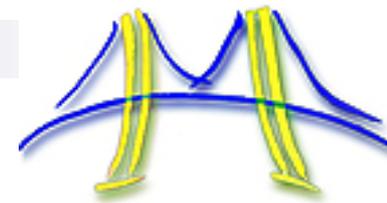


Intel Clovertown (8 threads)



[Diagrams Courtesy of Sam W. Williams]

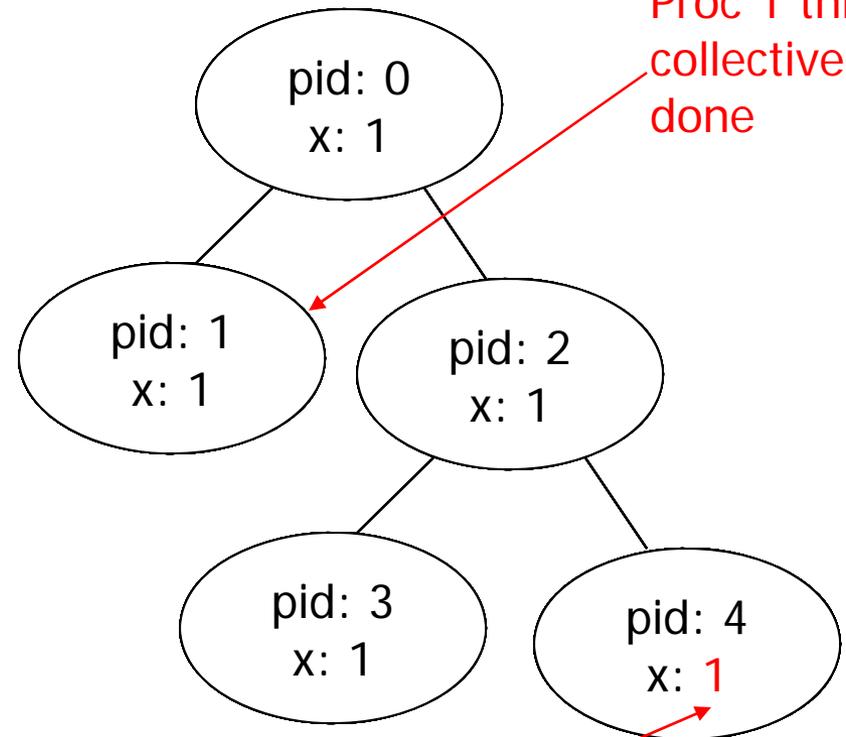
Potential Synchronization Problem



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

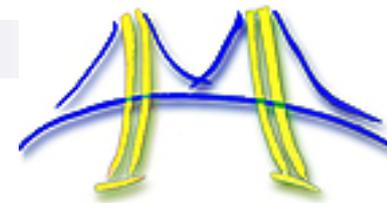
broadcast $x=1$ from proc 0

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

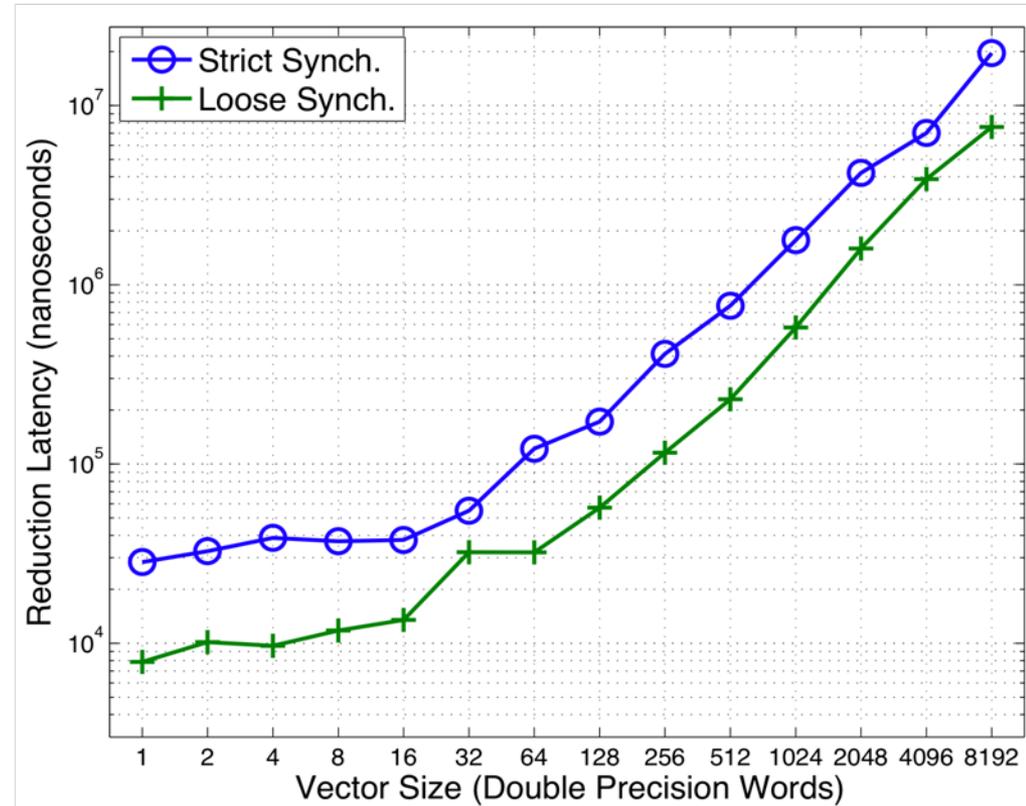


Put of $x=5$ by proc 1 has been lost
Proc 1 observes locally complete but globally incomplete collective

Strict v. Loose Synchronization

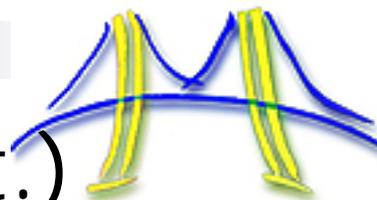


- A fix to the problem
 - Use synchronization before/after the collective
 - Enforce global ordering of the operations
- Is there a problem?
 - We want to decouple synchronization from data movement
 - Let user specify the synchronization requirements
 - Potential to aggregate synchronization
 - Done by the user or a smart compiler
- How can we realize these gains in applications?
- What's the best way to expose all this?



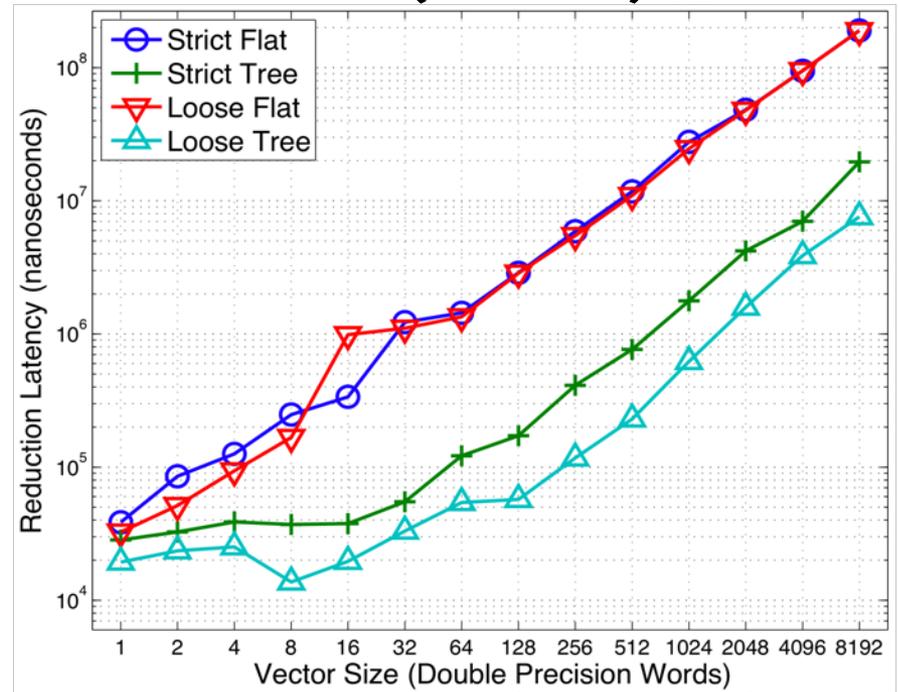
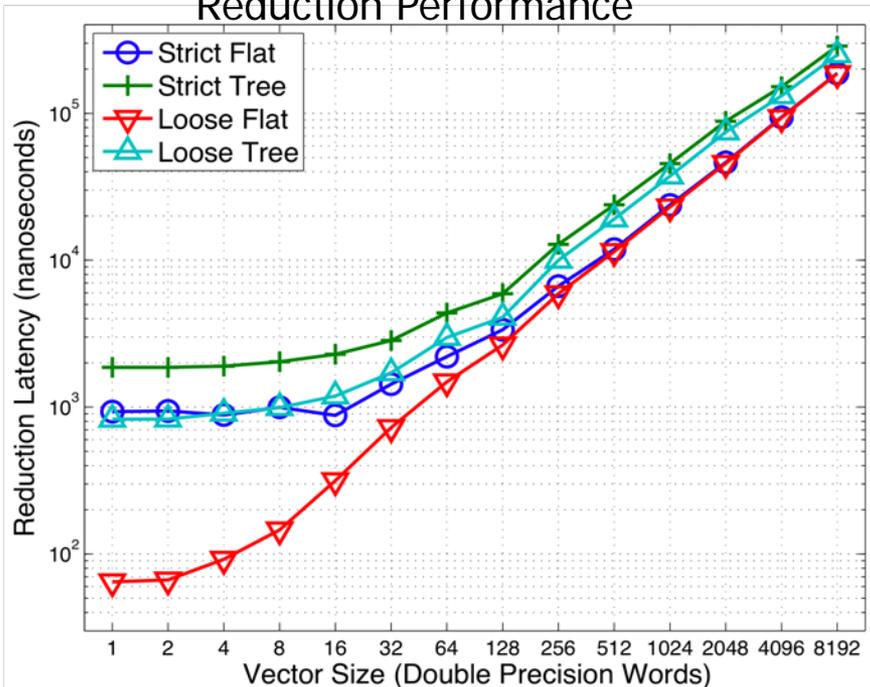
Sun Niagara2 (256 threads)
Reduction Performance

Autotuning and Synchronization (cont.)



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

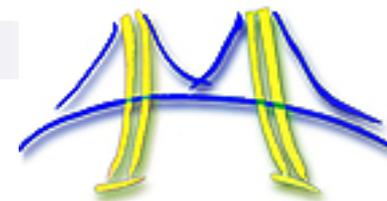
Intel Clovertown (8 threads)
Reduction Performance



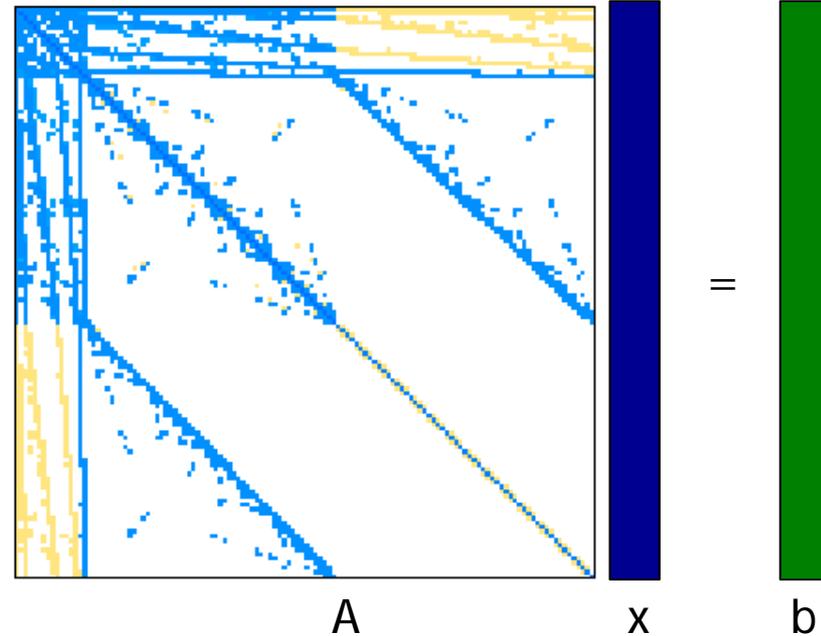
Sun Niagara 2 (256 threads)
Reduction Performance

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

Auto-tuned Conjugate Gradient

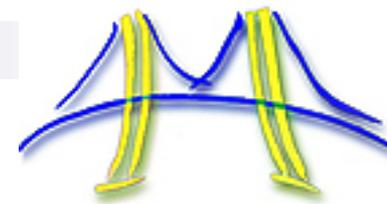


- Incorporate tuned collectives into an important kernel
- Sparse Conjugate Gradient
 - Part of Sparse Motif
 - Iteratively solve $Ax=b$ for x given A and b
 - Relies heavily on optimized SPMV and tuned BLAS1 operations
 - Matrix Partitioned Row-wise for our application
- Automatic tuning for a parallel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

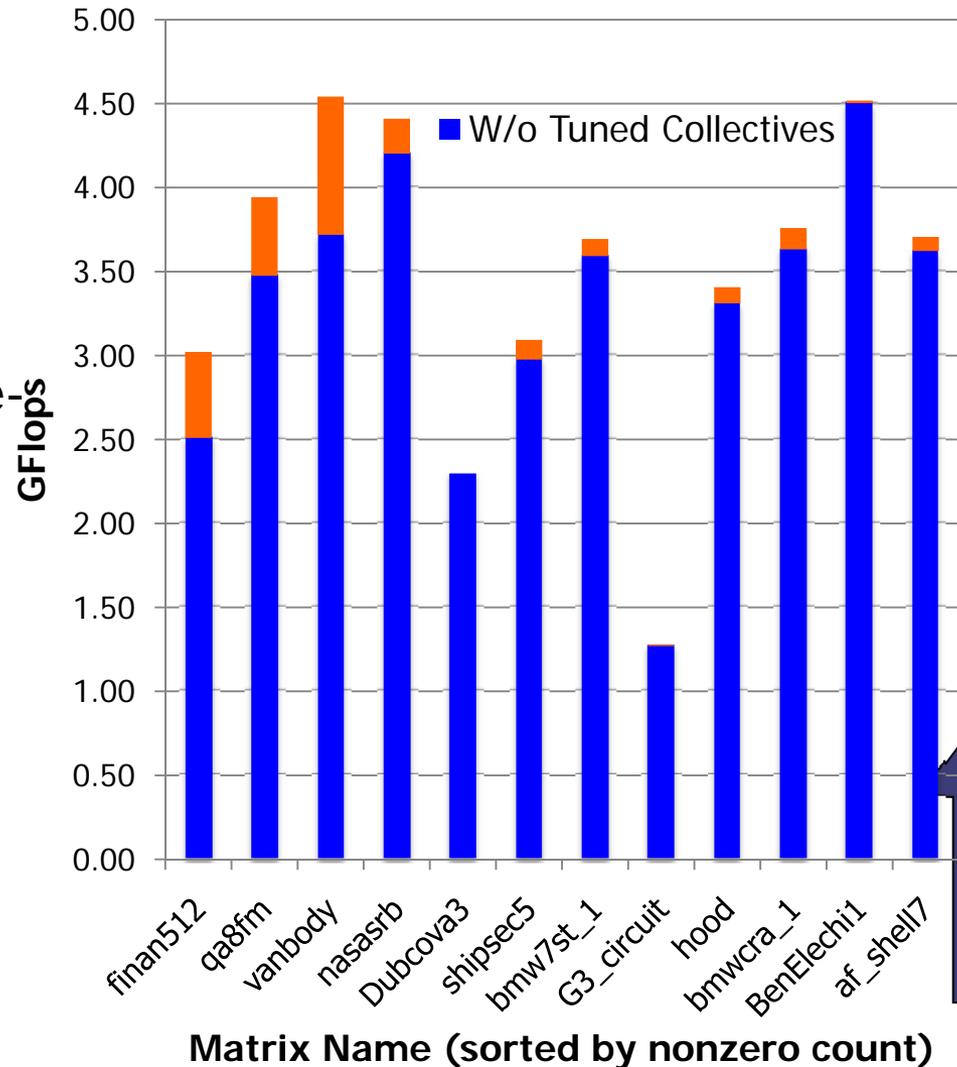


- Collectives Used:
 - Scalar Reduce-To-All for Dot Products
 - Barriers

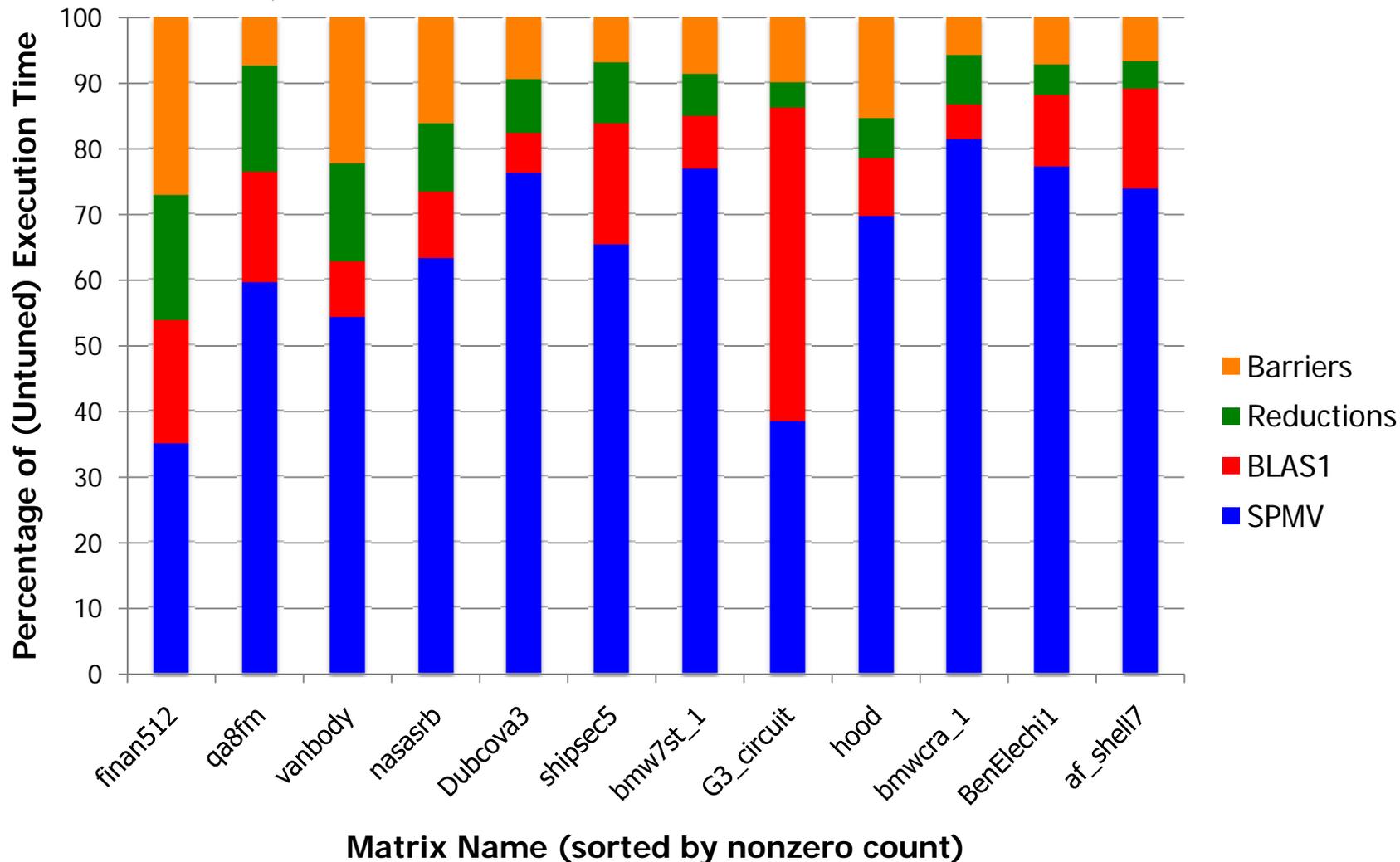
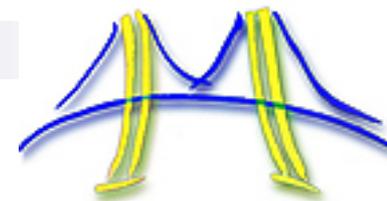
Conjugate Gradient Performance



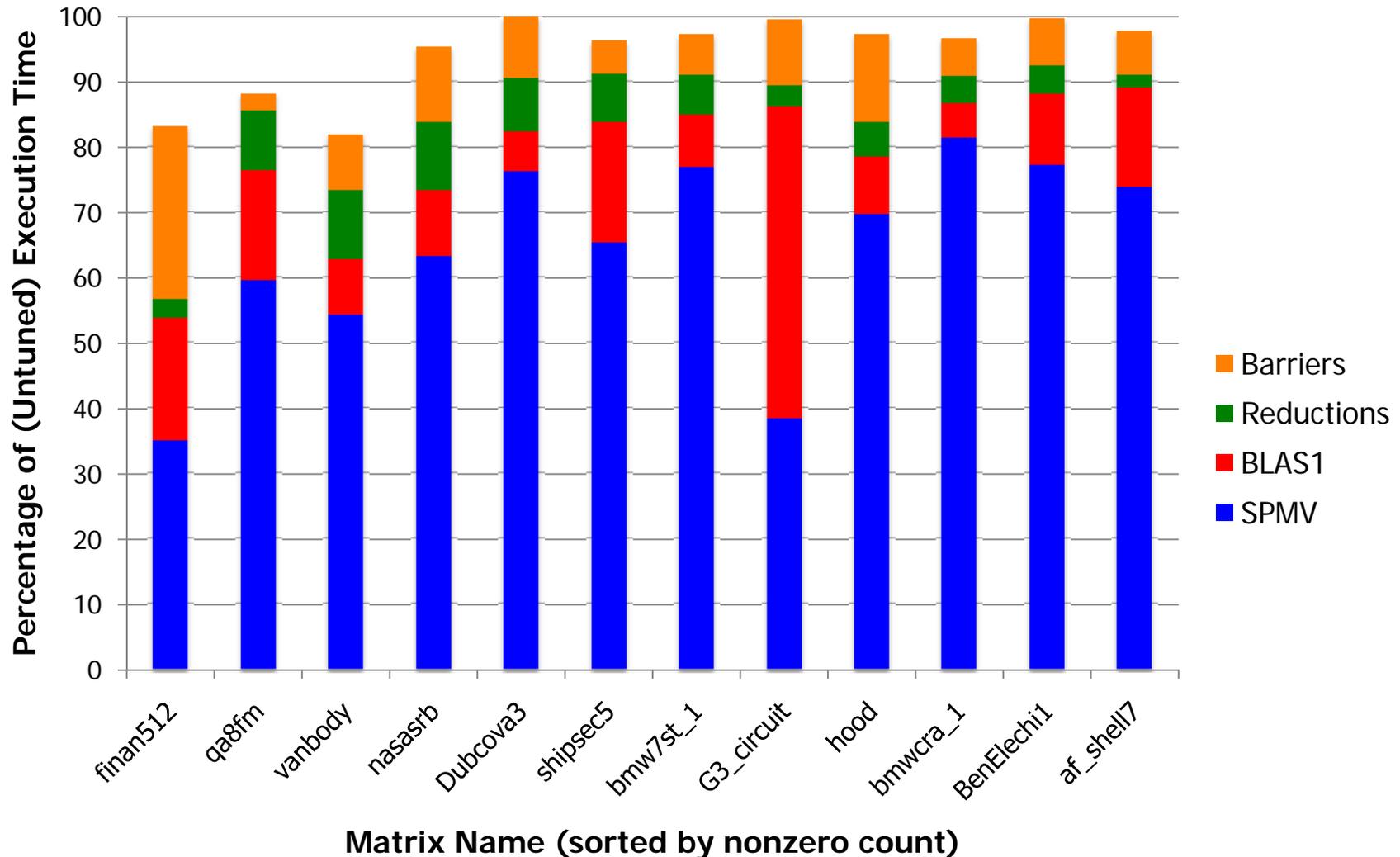
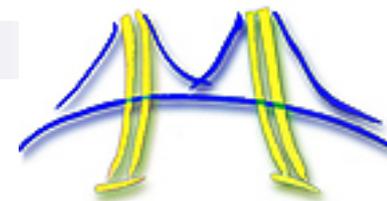
- Auto-tuned SPMV from Sam Williams [Williams et. al, SC'07]
- Sun Performance Library for local BLAS1 operations
- Incorporate aforementioned tuned barrier and tuned Reduce-to-All for inter-thread communication
- Matrix parallelized row-wise
 - reductions are performed across all 128 threads
- Best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



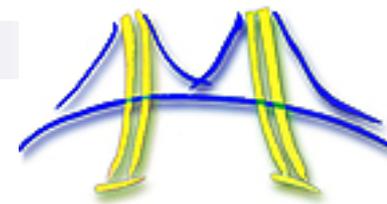
CG Performance Breakdown (untuned)



CG Performance Breakdown (tuned)



Conclusions



- As thread/core counts continue to grow rapidly
 - Collective tuning will become very important
 - Poor collective choice leads to dramatic performance penalties
- Optimal algorithms are dependent on many factors that are hard to model *a priori*
 - Often based on runtime factors such as synchronization requirements of the application
 - Thread layout on the machine affects the optimal algorithm
- We use auto-tuners to pick the best algorithm
- Show up to 21% gains in overall application performance in Conjugate Gradient
- Auto-tuned collectives will soon be incorporated into runtime system for Berkeley UPC compiler