Designing Topology-Aware Collective Communication Algorithms for Large Scale InfiniBand Clusters: Case Studies with Scatter and Gather

Krishna Kandalla, Hari Subramoni, Abhinav Vishnu and Dhabaleswar K. (DK) Panda

IPDPS 2010

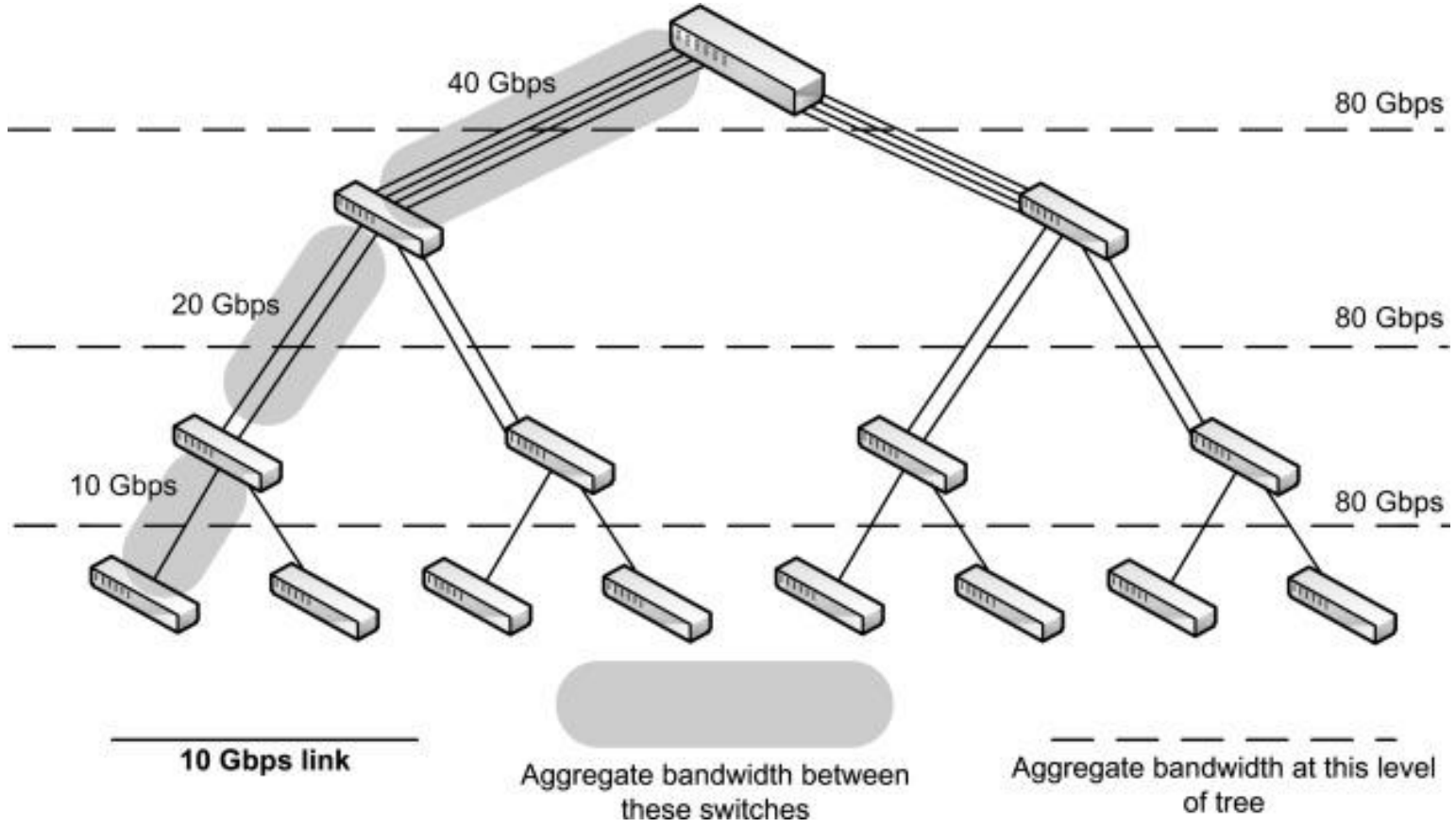
Lecture 20
April 16, 2025


IBM
Blue Gene/P

Research Questions

- How do we efficiently discover the topology of a large scale InfiniBand cluster?
- What are the challenges involved in designing efficient collective algorithms that are aware of the network topology?
- Can we derive communication cost models for collective operations on large-scale systems with several levels of hierarchies?
- What is the effect of the background traffic on the performance of collective operations? Can we leverage the topology information to design algorithms that are resilient to network contention?

Network Topology



Fat-tree topology

[www.sciencedirect.com]

A typical topology of large-scale systems (TACC Ranger system)

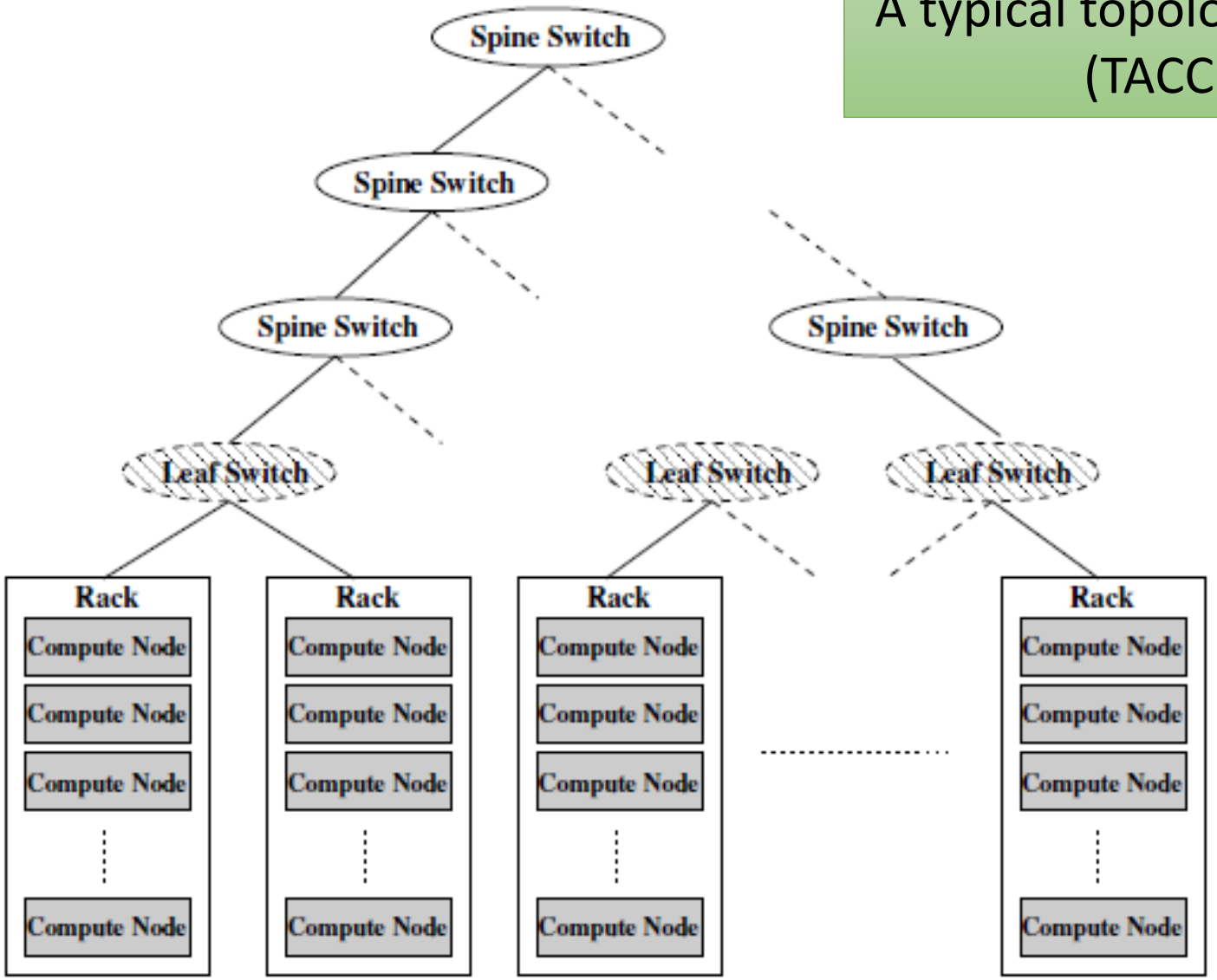
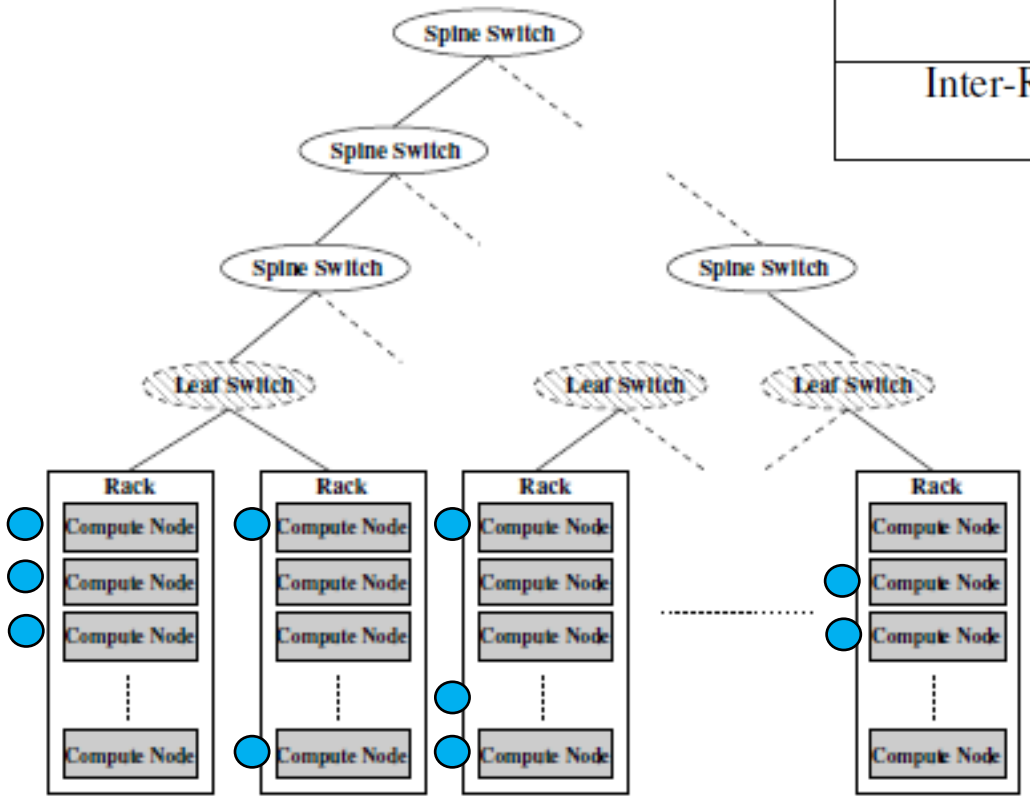


Figure 1. A Typical Topology

Effect of Topology on Latency

Process Location		Number of Hops	MPI Latency (*us*)
Intra-Rack	Intra-Chassis	0 Hops in Leaf Switch	
	Inter-Chassis	1 Hop in Leaf Switch	
Inter-Rack		3 Hops Across Spine Switch	
		5 Hops Across Spine Switch	

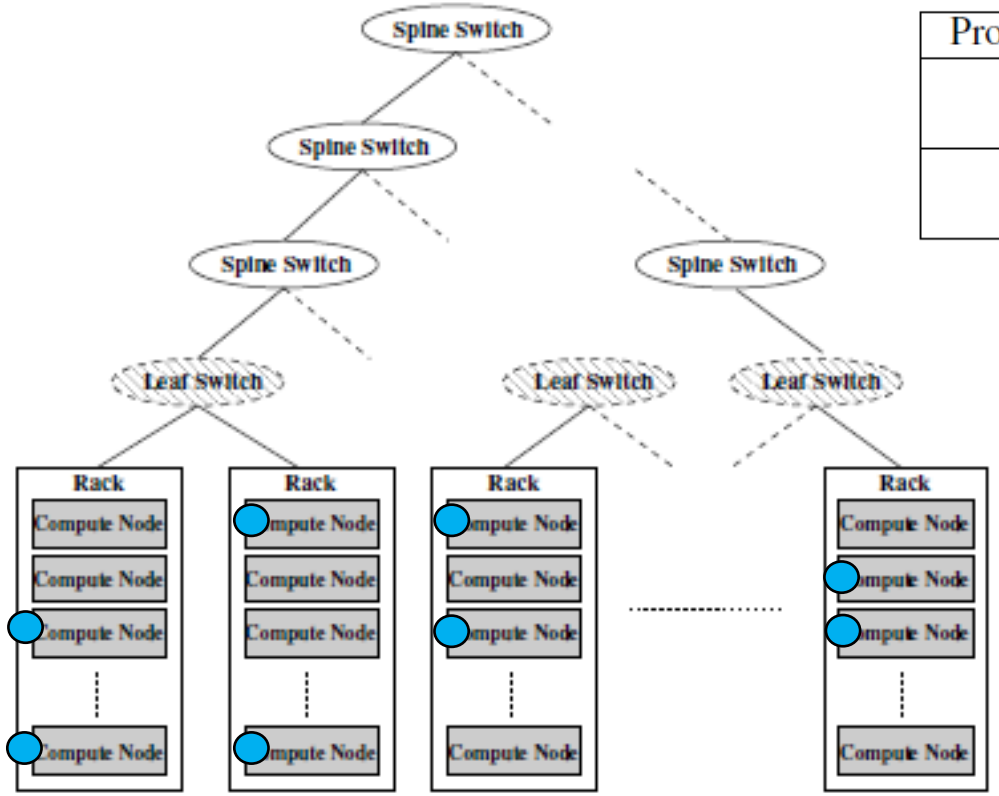


A typical topology of large-scale systems

(TACC Ranger system)

- Allocated nodes are usually scattered in the system
- Different job request sizes and durations
- Contiguous node allocation may increase queue waiting times

MPI_Gather



Process Location	
Intra-Rack	Intra-Chassis
	Inter-Chassis
Inter-Rack	

L_1 L_2 L_3

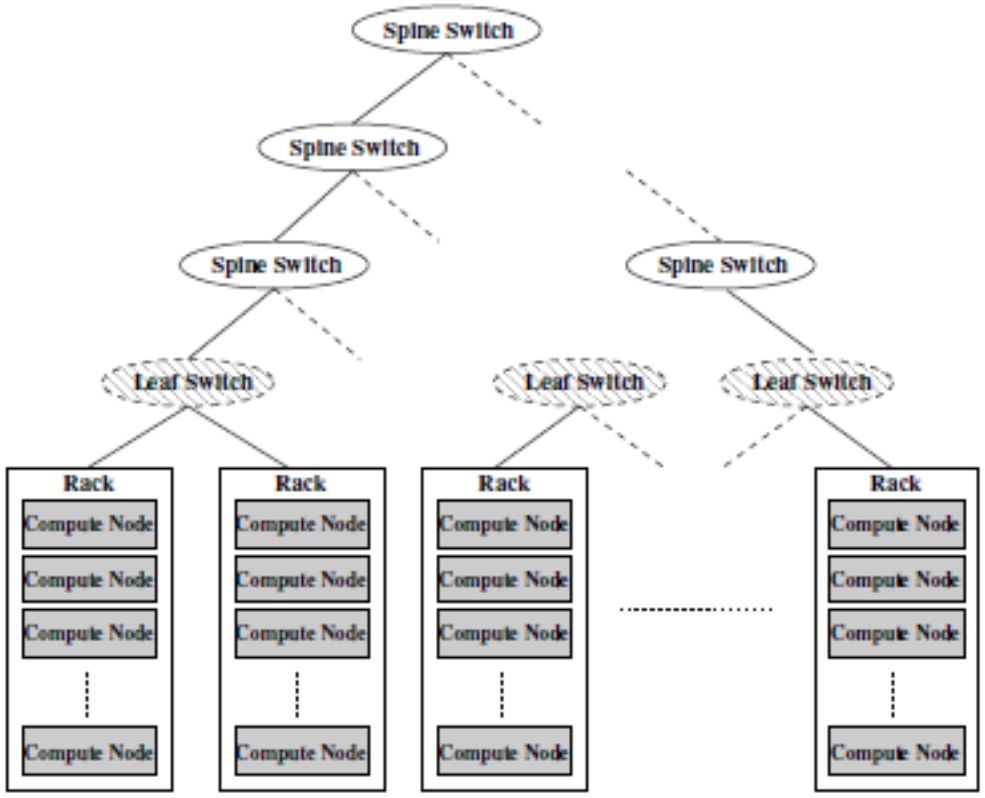
B_1 B_2 B_3

Contribution: Topology-aware Collectives

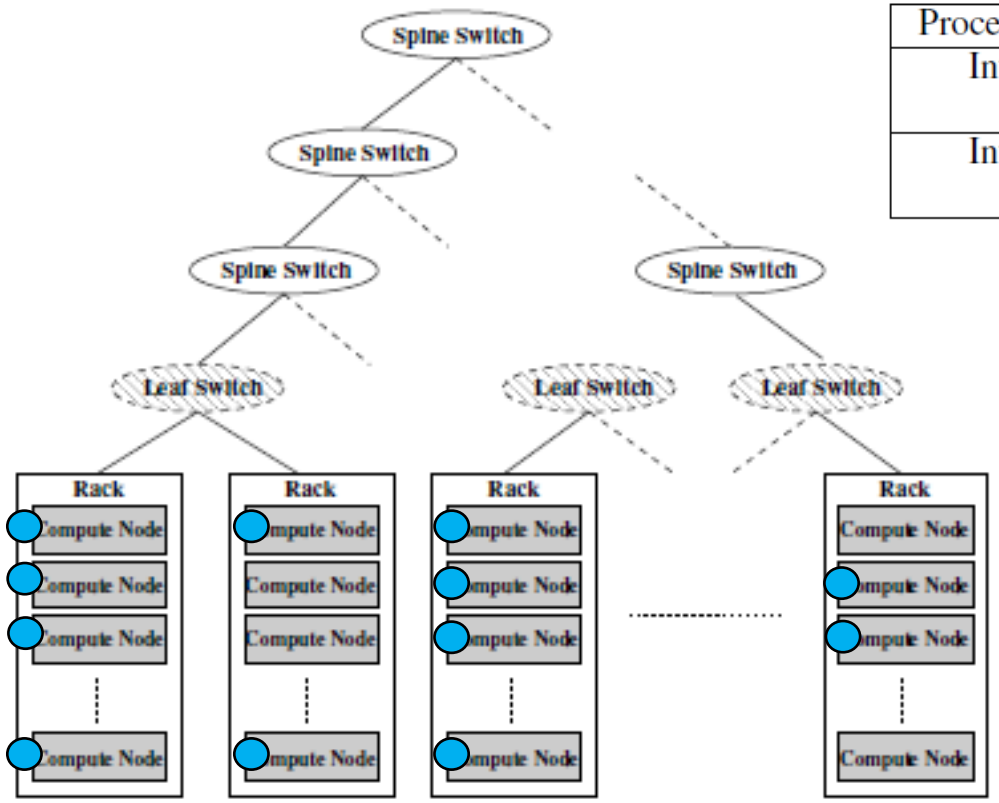
- Topology-aware Gather and Scatter
- Modified communication model
- 54% improvement on micro-benchmarks

Topology-aware Gather

- Designate a rack leader process
- Rack-leader processes independently perform intra-switch gather
- R rack leaders perform inter-switch gather
- Reduced L and B terms (due to reduction in inter-switch exchanges)



Create Sub-communicators



Process Location		Number of Hops	MPI Latency (*us*)
Intra-Rack	Intra-Chassis	0 Hops in Leaf Switch	1.57
	Inter-Chassis	1 Hop in Leaf Switch	2.04
Inter-Rack		3 Hops Across Spine Switch	2.45
		5 Hops Across Spine Switch	2.85

- intra-chassis communicators
- intra-switch communicators
- chassis-leader communicators
- switch-leader communicators

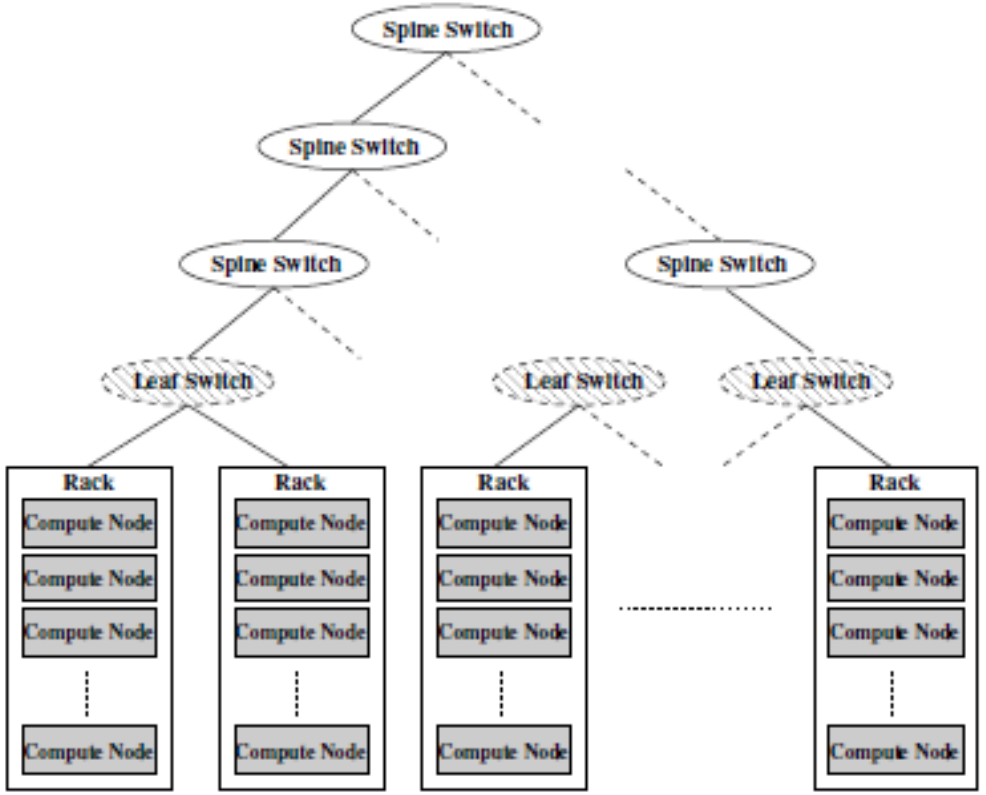
Discover Topology

- Infiniband tools
  - ibnetdiscover – outputs the switch connections / identifiers
  - One-time discovery (in general)

- MPI_Init
  - Create intra-chassis communicators – all nodes in the same chassis
  - Create intra-switch communicators – all nodes in the same leaf switch
  - Assign one chassis-leader and one switch-leader
  - Create switch-leader and chassis-leader communicators

Cost of Communication

t_s-intra-node < t_s-intra-switch < t_s-inter-switch
t_w-intra-node < t_w-intra-switch < t_w-inter-switch



Cost involved for communication within the same node
L: t_s-intra-node
B: t_w-intra-node

Cost of communication within the same leaf switch
L: t_s-intra-switch
B: t_w-intra-switch

Cost involved for an inter-switch communication
L: t_s-inter-switch
B: t_w-inter-switch

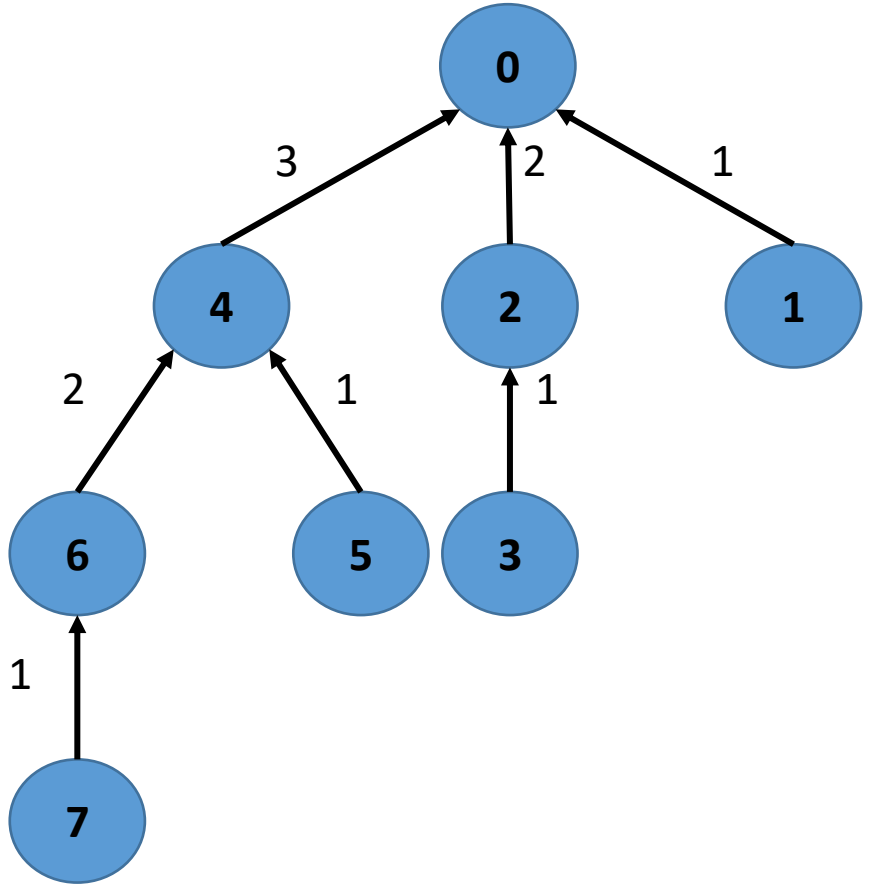
1. Actual cost depends on the #hops based on the actual placement of processes
2. Contention for intra-node/switch << inter-switch

Gather Communication Cost

Binomial tree

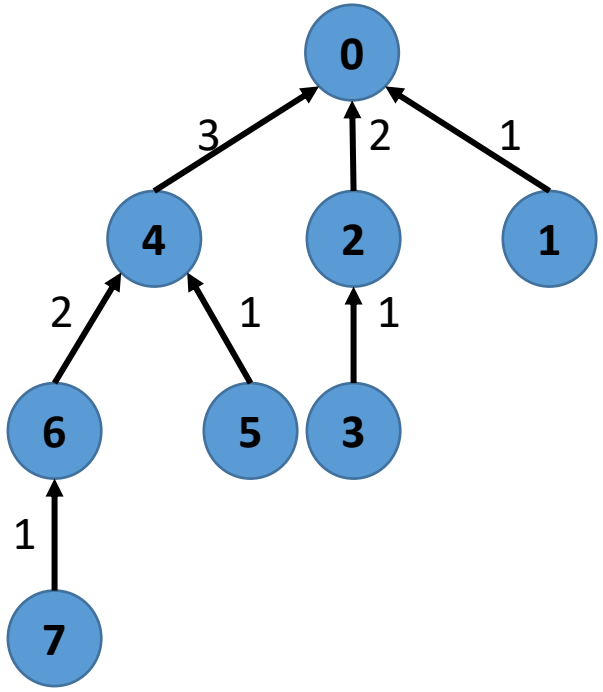
Time: (log p)*L + (p-1)/p*(n/B)

L = latency, B = bandwidth



Cost Model (Original Gather - Binomial)

Number of racks = R
Number of processes = P
Message size = N



Number of exchanges at i^{th} level: C_i
C_1 = Number of intra-node transfers
C_2 = Number of intra-switch transfers
C_3 = Number of inter-switch transfers
$C_1 + C_2 + C_3 = \log P$
Switch-level contention: α

There's a typo in the paper. Read this as ts-intra-node

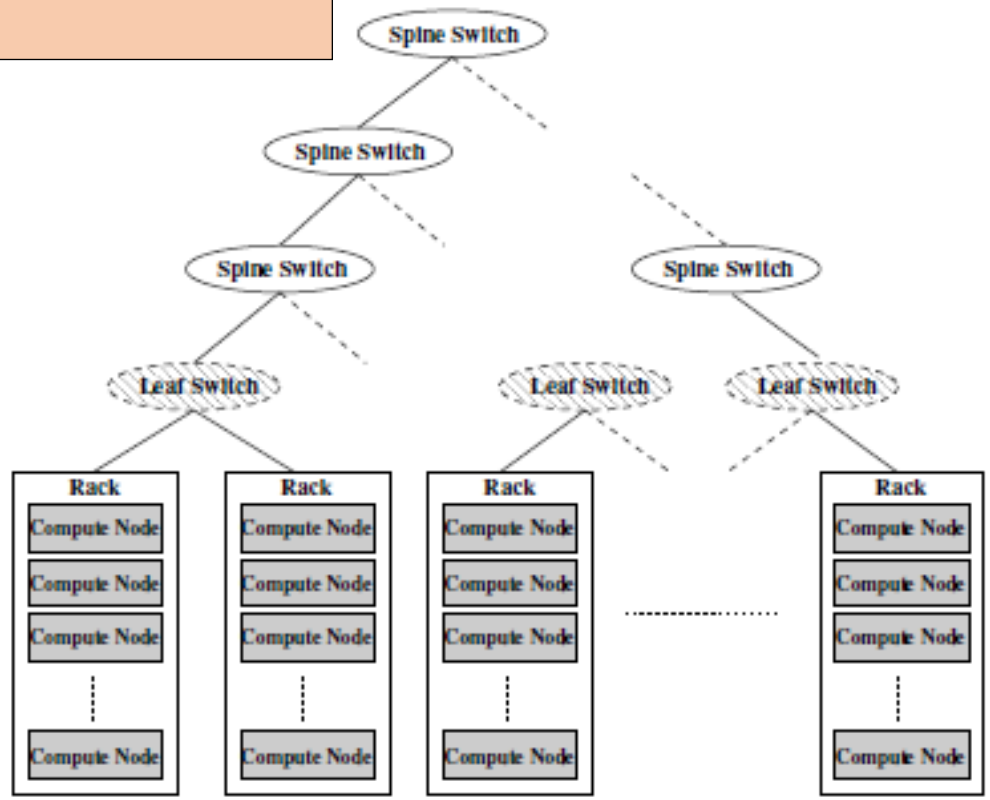
$$T_{binomial} = (t_s\text{-}inter\text{-}node * C_1 + t_s\text{-}intra\text{-}switch * C_2 + \alpha * t_s\text{-}inter\text{-}switch * C_3) + t_w\text{-}intra\text{-}node * (C_1) * (N * \gamma) + t_w\text{-}intra\text{-}switch * (C_2) * (N * \beta) + \alpha * t_w\text{-}inter\text{-}switch * (C_3) * (N * \delta)$$

Communication Cost for Gather (Binomial)

- The bandwidth term is obtained by adding costs at each level
- $[C_1 * \gamma + C_2 * \beta + C_3 * \delta] * N = (p-1)/p * N$

There's a typo in the paper. Read this as ts-intra-node

$$T_{binomial} = (t_s\text{-}inter\text{-}node * C_1 + t_s\text{-}intra\text{-}switch * C_2 + \alpha * t_s\text{-}inter\text{-}switch * C_3) + t_w\text{-}intra\text{-}node * (C_1) * (N * \gamma) + t_w\text{-}intra\text{-}switch * (C_2) * (N * \beta) + \alpha * t_w\text{-}inter\text{-}switch * (C_3) * (N * \delta)$$



Communication Cost Comparison

$$T_{binomial} > (\alpha * t_s\text{-}inter\text{-}switch * C_3) + (\alpha * N * t_w\text{-}inter\text{-}switch * C_3 * \delta)$$

$$T_{topo} > (\alpha * t_s\text{-}inter\text{-}switch * log(R)) + (\alpha * (1 - 1/R))/(M * t_w\text{-}inter\text{-}switch)$$

There's another typo in the paper !! Read / as*

$$T_{binomial}/T_{topo} = N * \delta * C_3/(M * (1 - 1/R))$$

$$T_{binomial}/T_{topo} = f(R)$$

1. The rack-leader processes independently perform an intra-switch gather operation. This phase of the algorithm does not involve any inter-switch exchanges.
2. Once the rack-leaders have completed the first phase, the data is gathered at the root through an inter-switch gather operation performed over the *R* rack-leader processes.

Experimental Setup

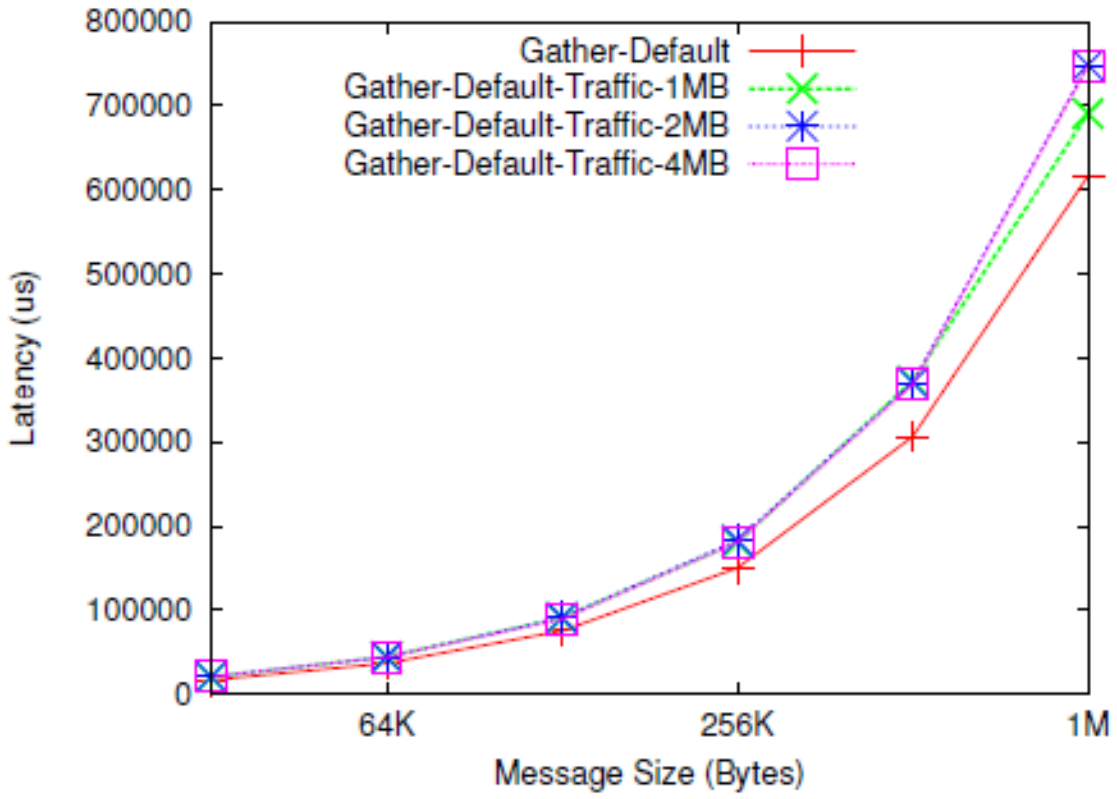
- Three InfiniBand DDR switches A, B and C to create a tree topology.
- Switch A is connected to 8 nodes based on the quad-core, quad-socket AMD Barcelona architecture (4 nodes /64 processes used)
- Switch B is connected to 32 nodes based on the quad-core, dual-socket Intel Clovertown architecture (29 nodes/ 232 processes used)
- Switches A and B are connected to Switch C with two InfiniBand DDR links each.

Benchmark

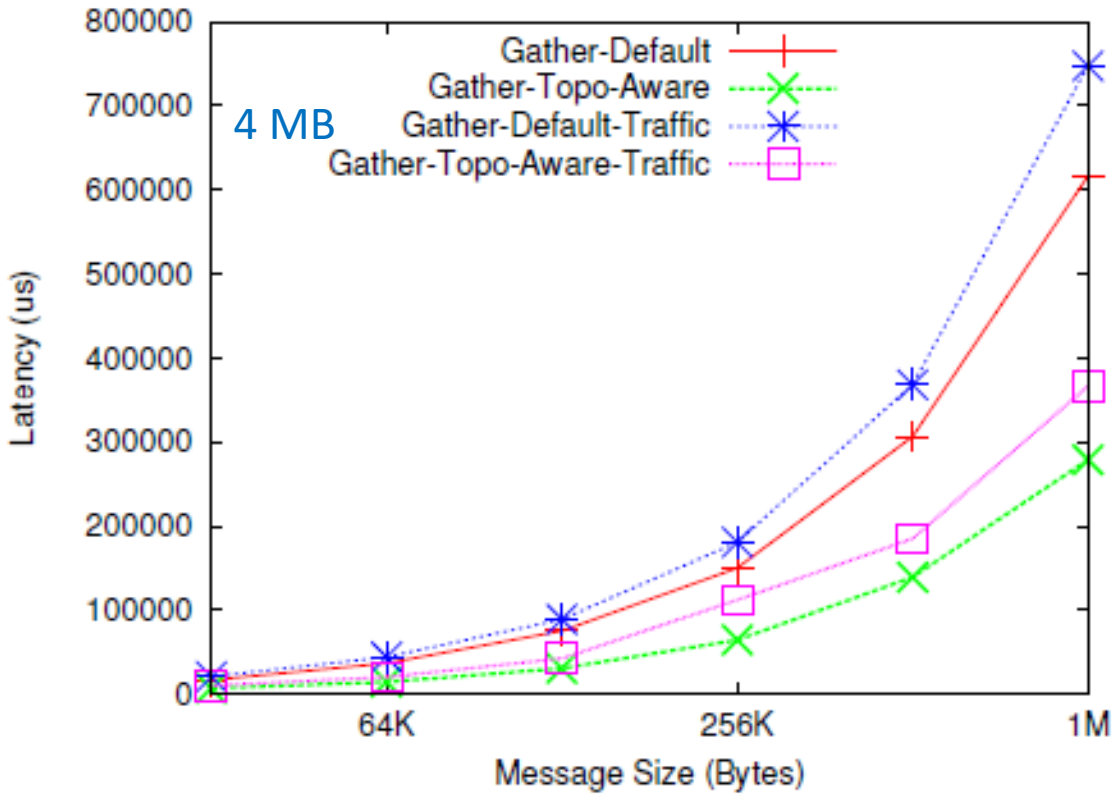
- A simple benchmark code iterates through various message sizes (0 – 1 MB) and invokes a collective call several times in a loop.
- Measured average time when system is quiet
- AlltoAll is used to create background traffic (K/2 in each switch)
  - Constant traffic K*M bytes over the switches

Gather Results (With and Without Traffic)

21% degradation

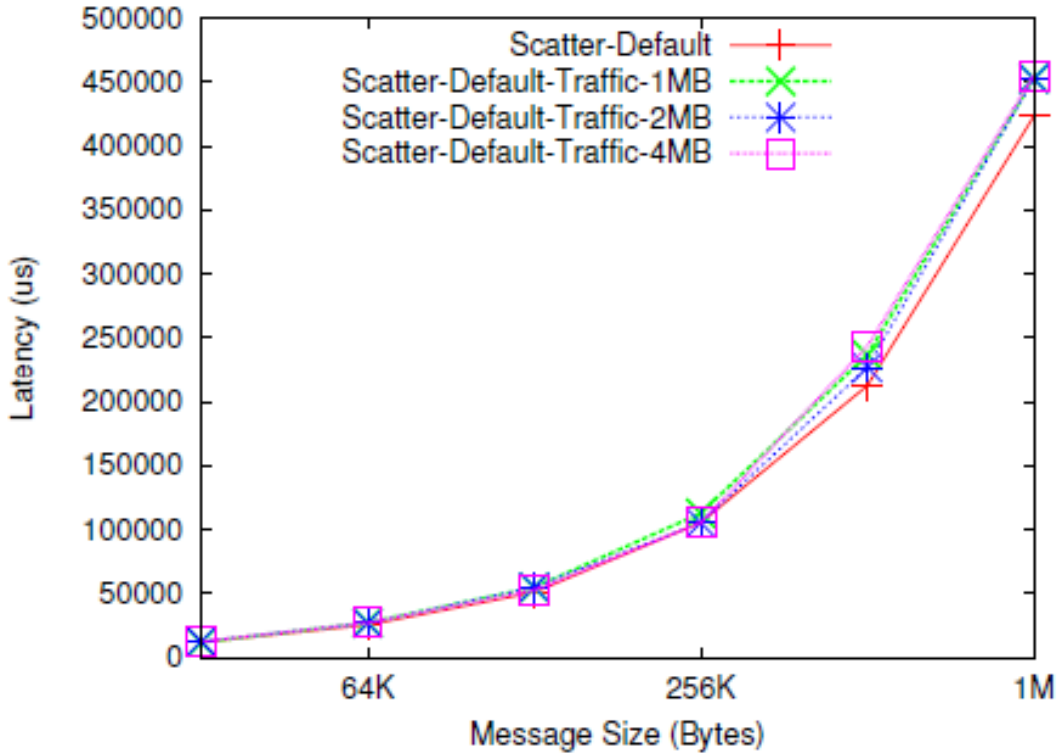


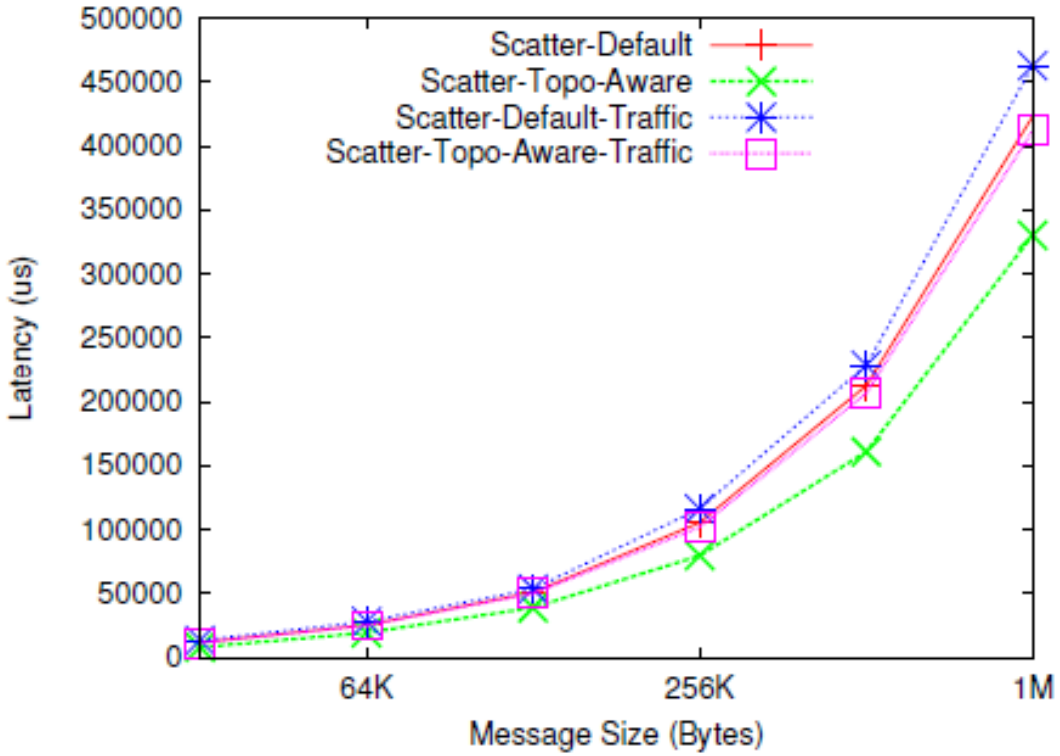
50% reduction



Scatter Results

Q: Why is T_Scatter < T_Gather?



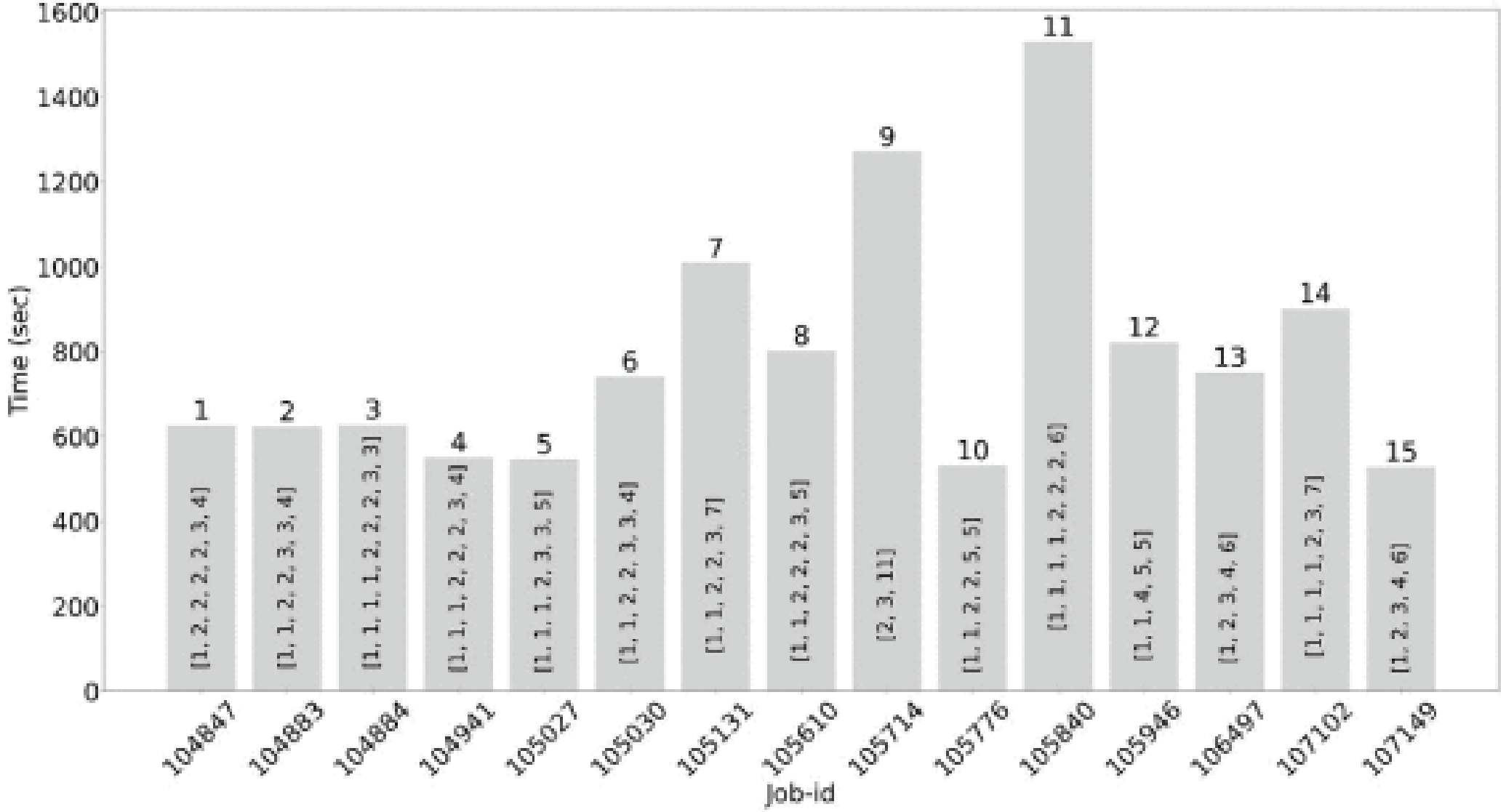


Conclusions

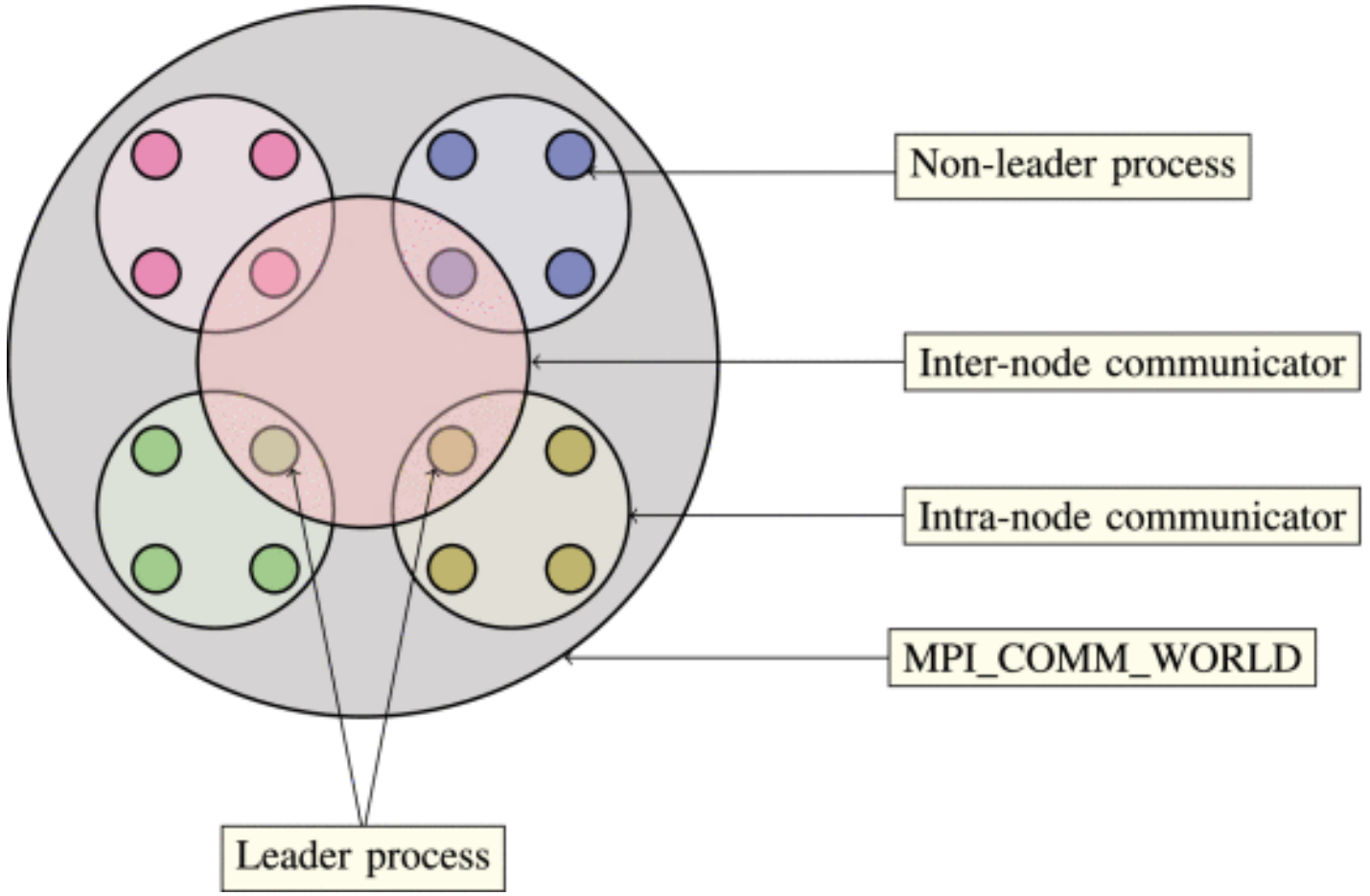
- Presence of background traffic may lead to performance degradation
- Efficient software stack may improve performance
- Topology-aware Gather and Scatter was shown to perform better

Hierarchical Communication Optimization for FFT, M Kumar, P Malakar, SC W HiPar 2022

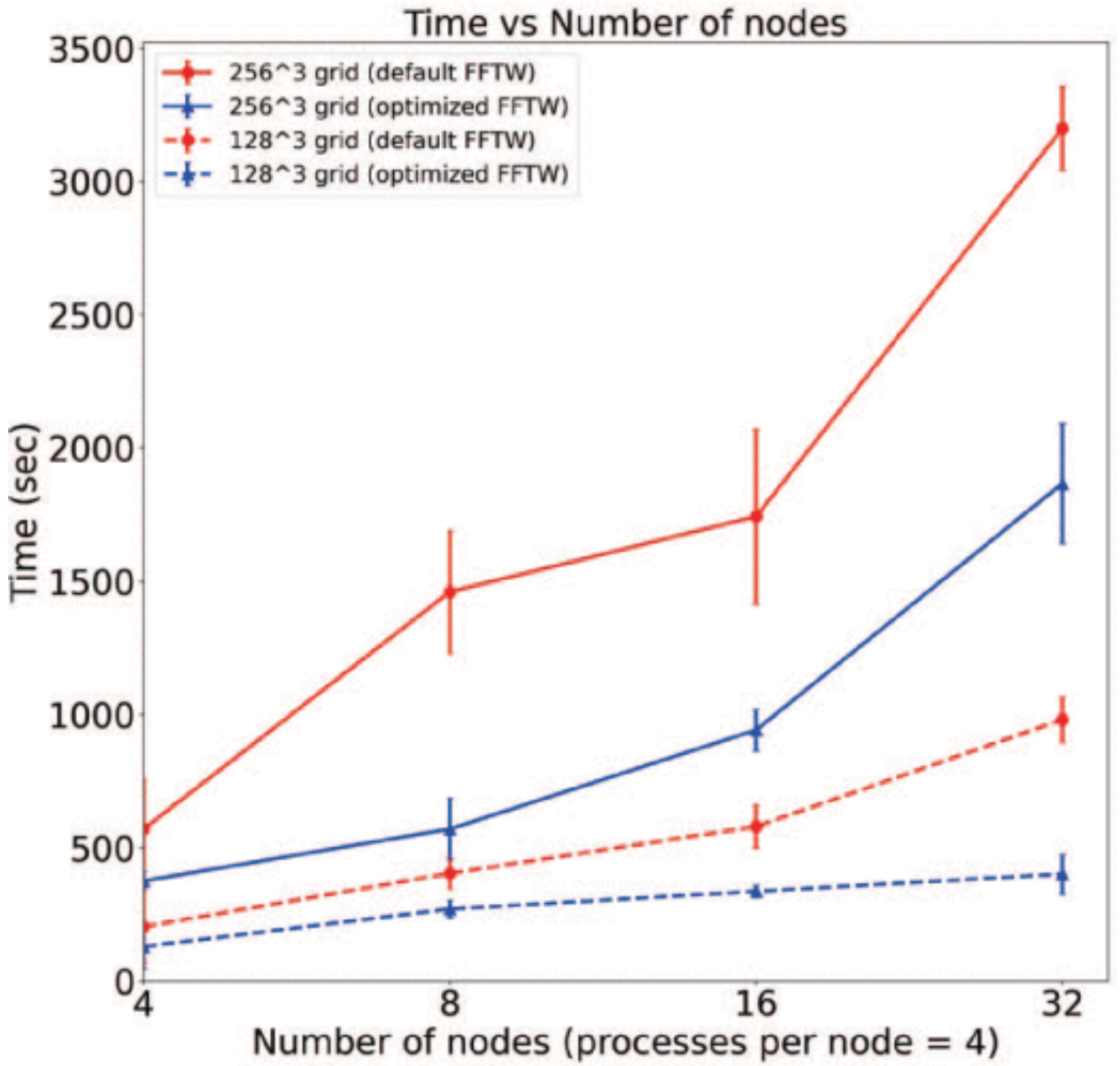
Variation in Job Runtimes



Hierarchical AlltoAll/Sendrecv



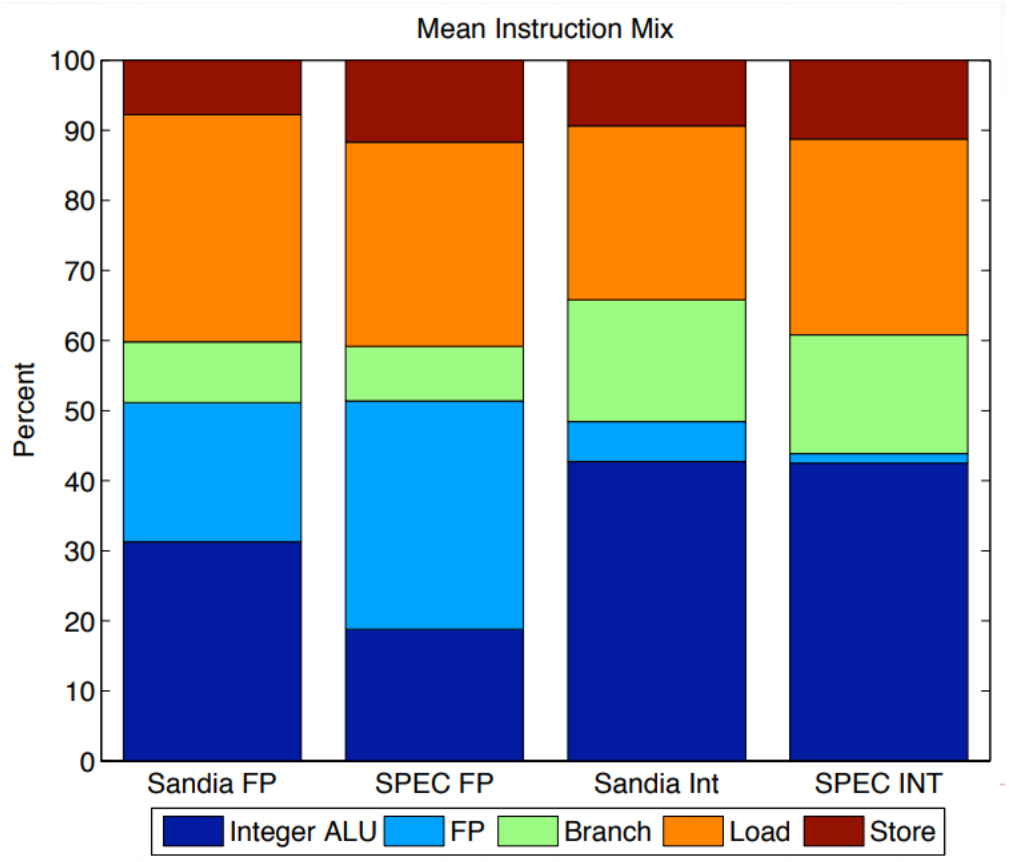
Results


Time vs Number of nodes
256^3 grid (default FFTW)
256^3 grid (optimized FFTW)
128^3 grid (default FFTW)
128^3 grid (optimized FFTW)
Time (sec)
0
500
1000
1500
2000
2500
3000
3500
4
8
16
32
Number of nodes (processes per node = 4)

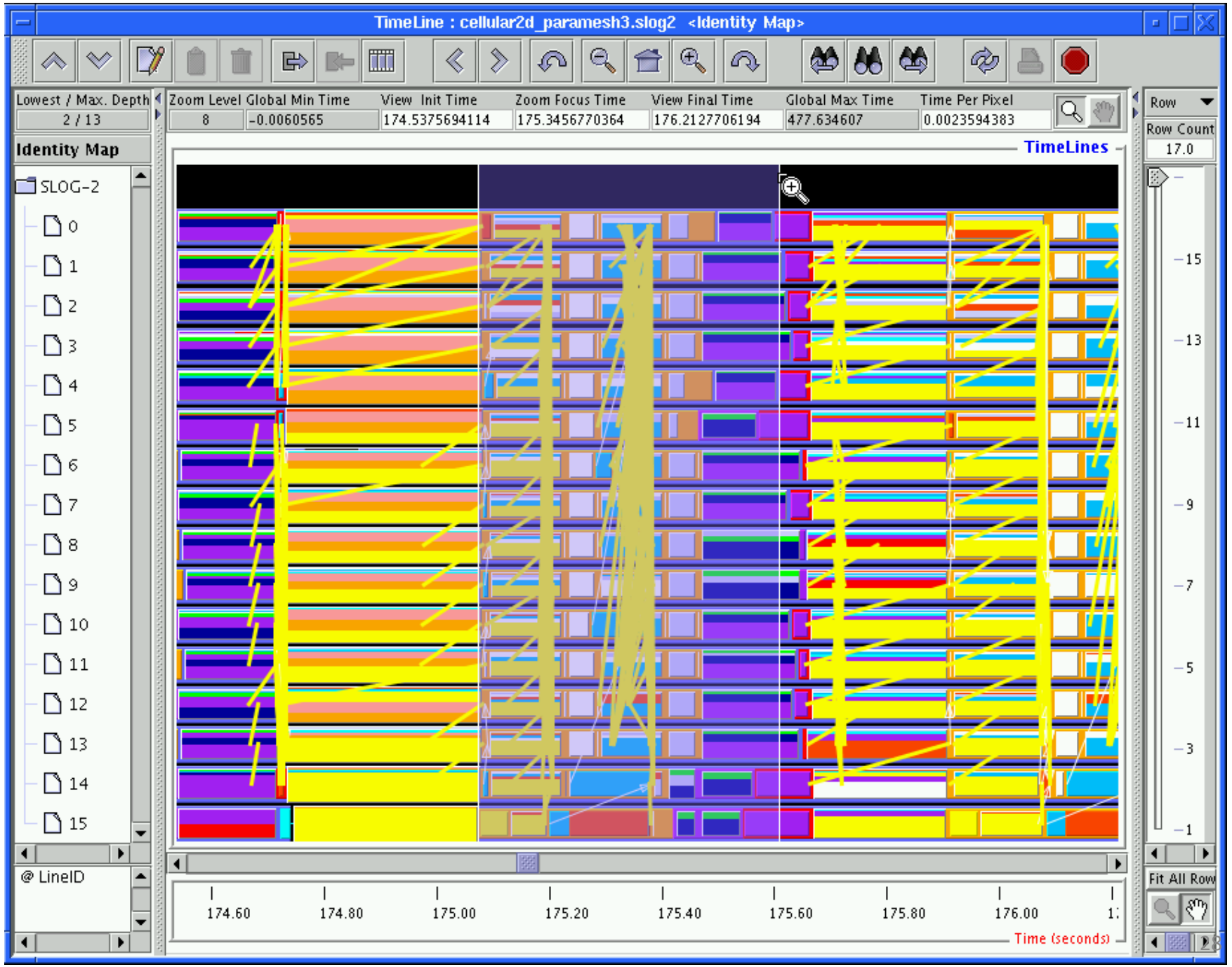
Profiling

Analyzing Performance

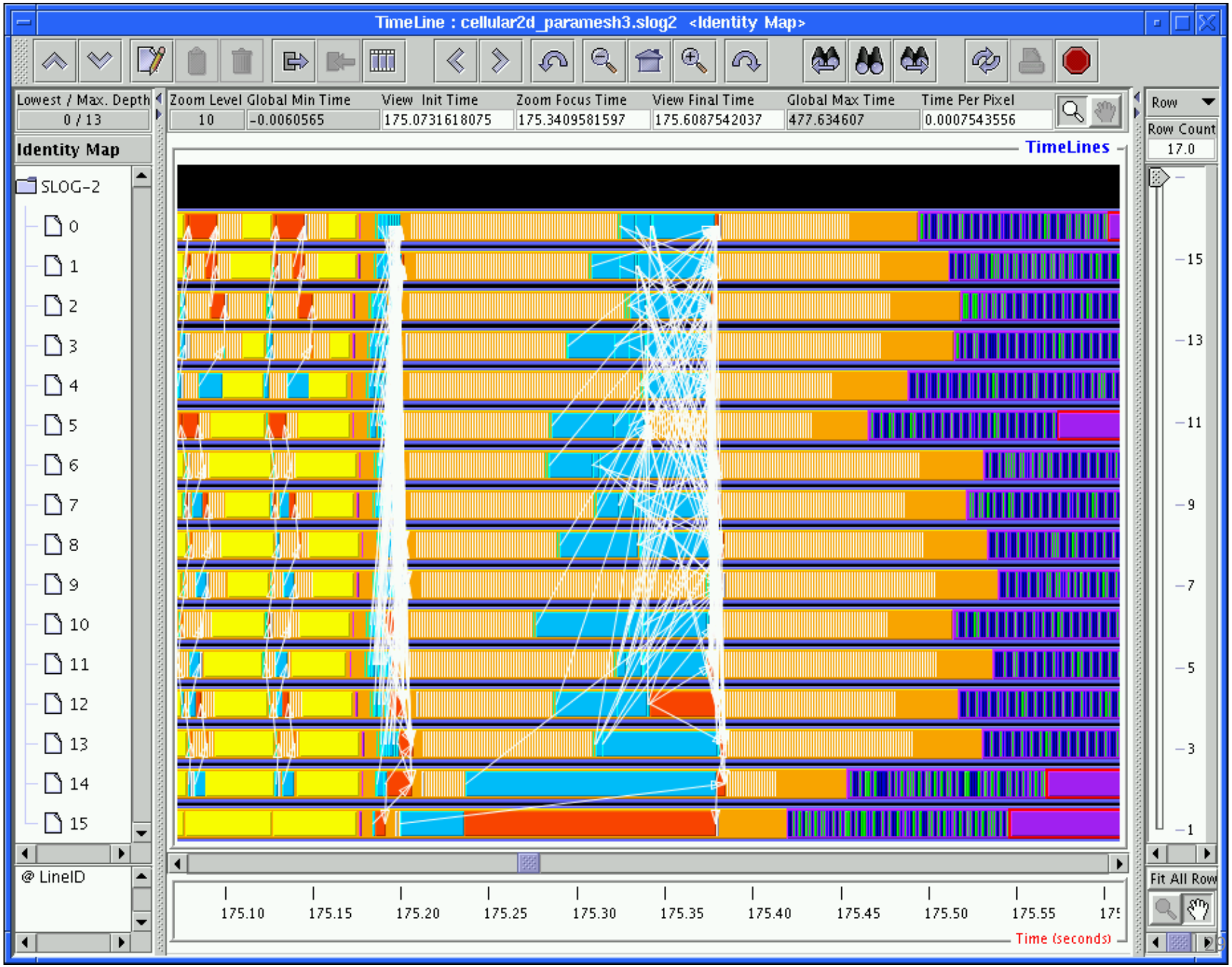
Mean Instruction Mix of Different Workloads



Reference: **On the Memory Access Patterns of Supercomputer Applications: Benchmark Selection and Its Implications, Murphy and Kogge, IEEE Transactions on Computers, 2007**



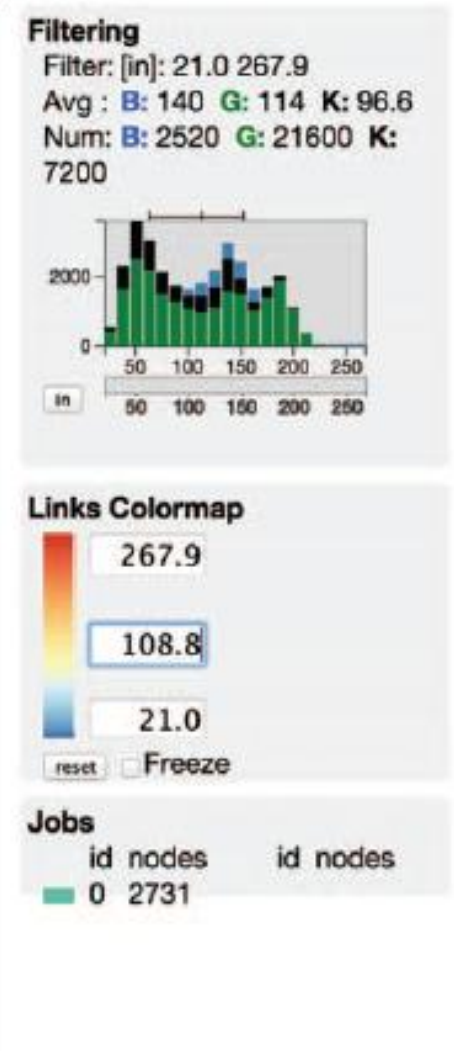
https://www.mcs.anl.gov/research/projects/perfvis/software/viewers/

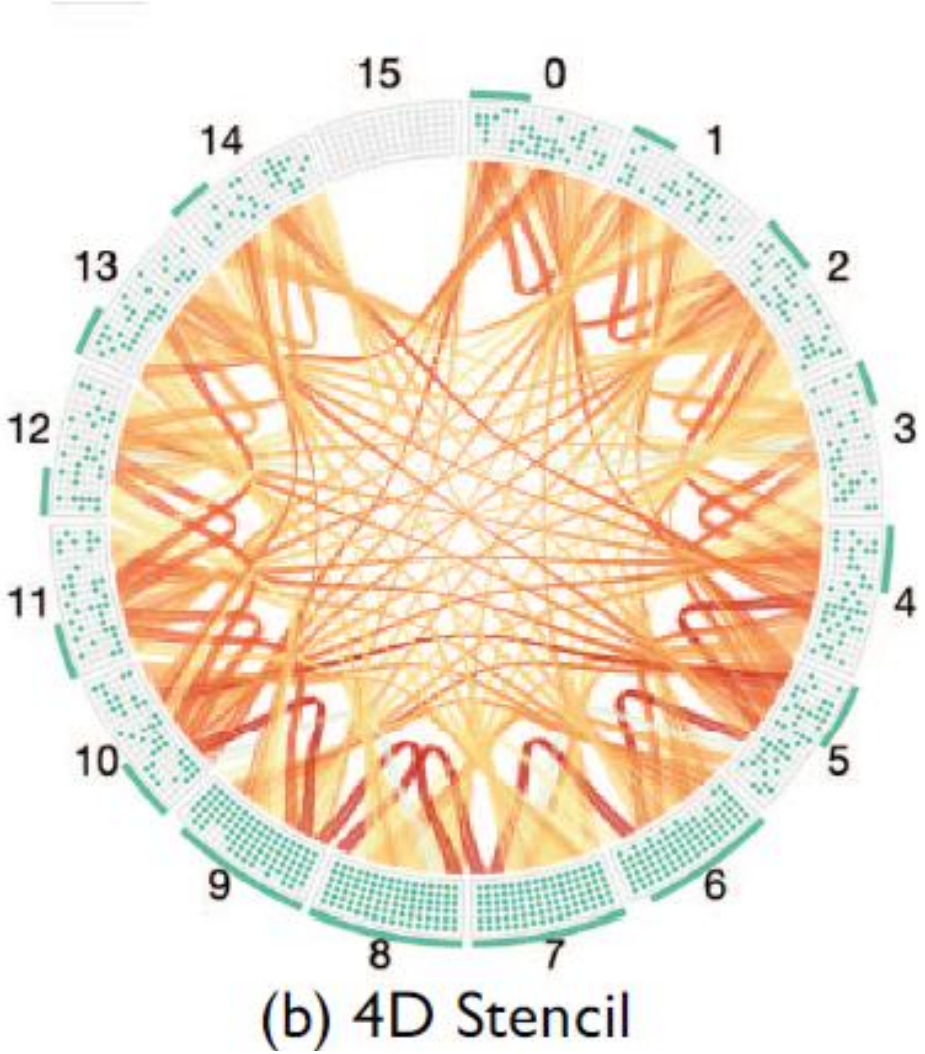

TimeLine : cellular2d_paramesh3.slog2 <Identity Map>
Lowest / Max. Depth
0 / 13
Zoom Level
10
Global Min Time
-0.0060565
View Init Time
175.0731618075
Zoom Focus Time
175.3409581597
View Final Time
175.6087542037
Global Max Time
477.634607
Time Per Pixel
0.0007543556
Identity Map
SLOG-2
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
@ LineID
TimeLines
Row
Row Count
17.0
15
13
11
9
7
5
3
1
Fit All Row
175.10
175.15
175.20
175.25
175.30
175.35
175.40
175.45
175.50
175.55
Time (seconds)

Performance Analysis

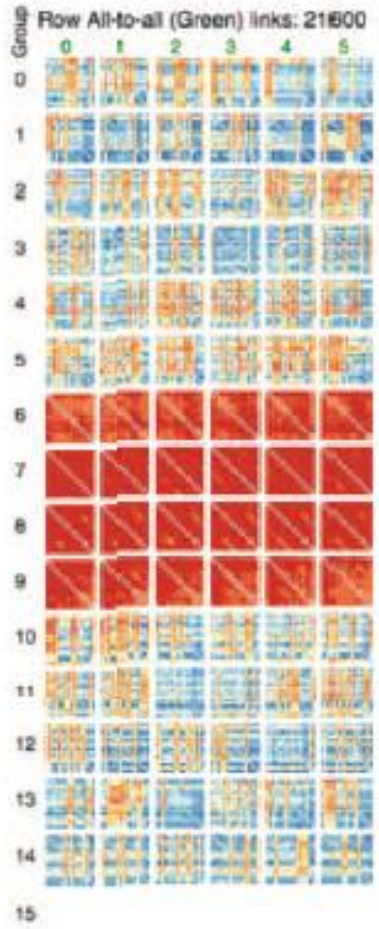
- Identify hot spots
- Identify where resources are being used inefficiently
- Difference between peak and average performance
- What matters is proportion of events
  - Floating point arithmetic vs. memory operations
- Performance analysis tools must collect and present relevant information on application performance in a scalable manner
  - Enable developers to easily identify and determine the causes of performance bottlenecks
  - Crucial to be not immersed in sea of data

Example (Custom Analysis)





(b) 4D Stencil



Profiling

- Elapsed time for the program and procedures
- Collection of statistical summaries of various performance metrics broken down by program entities such as routines and nested loops
- Performance metrics include time as well as hardware counter metrics such as operation counts and cache and memory event counts
- Profiling can identify regions of a program that are consuming the most resources

Tracing

- Timestamp attached with each event
- Collection of a timestamped sequence of events
  - Entering and exiting program regions
  - Sending and receiving messages.
- Can help identify the causes of performance problems
- Event traces record the temporal and spatial relationships between individual runtime events
- Cons: Huge space requirement

Performance Counters

Operation count data - Hardware

- Number of cache misses
- Number of loads/stores
- Main memory stalls

Operation count data - Network

- Number of packets sent and received
- Number of packets waiting to be sent

Profiling

- Source instrumentation
  - Know the location of bottleneck a priori for source code instrumentation
  - Huge endeavor to instrument every procedure in a large application
- Compile-time instrumentation
- Run-time instrumentation

Source Code Instrumentation

```
start = MPI_Wtime()
…
…
…
end = MPI_Wtime()
```

Source Code Instrumentation

```
/* Initialize the PAPI library and get the number of counters available */
if ((num_hwcntrs = PAPI_num_counters()) <= PAPI_OK)
        handle_error(1);


/* Start counting events */
if (PAPI_start_counters(Events, num_hwcntrs) != PAPI_OK)
        handle_error(1);
```

Compile-time/Run-time Instrumentation

- Code recompilation
  - -L<path-to-lib> -l<libname>
- Code recompilation not required
  - LD_PRELOAD, LD_LIBRARY_PATH..

Some Factors

Scalability

- Large number of processes
- Many functions
- Different inputs

Data Format

- Some common log formats
- Most are tool-specific

Presentation

- Present data in a hierarchical fashion
- Prioritize – present first what happens to be important

THE TAU PARALLEL PERFORMANCE SYSTEM

Sameer S. Shende

Allen D. Malony

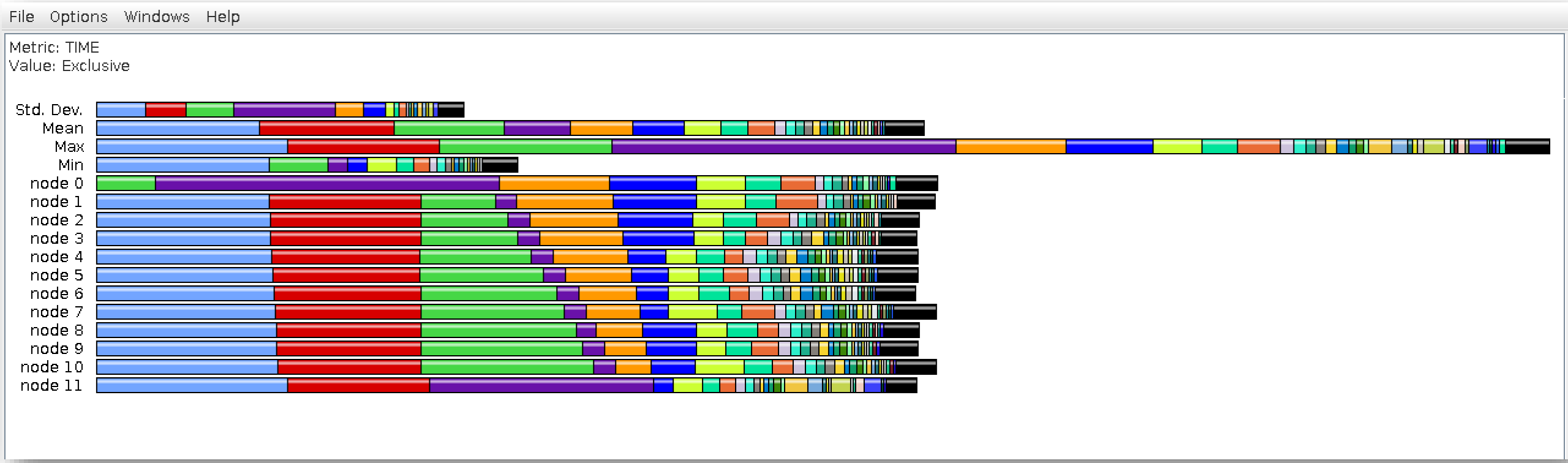
Tuning and Analysis Utilities (TAU)

- Supports tracing and profiling
- Supports different automated instrumentation options
  - Source code
  - Compiler based
- Uses call-path profiling to determine relevant routines for tracing
- Data management framework
  - Relational database
  - Graphical display
- Search traces to automatically identify inefficiency

TAU Profiling

- Performance measurements for functions, basic blocks, statements.
- Exclusive and inclusive time spent in functions (ns)
- #times each function was called, mean inclusive time per call, etc.

TAU Snapshot



HPCTOOLKIT: Tools for performance analysis of optimized parallel programs

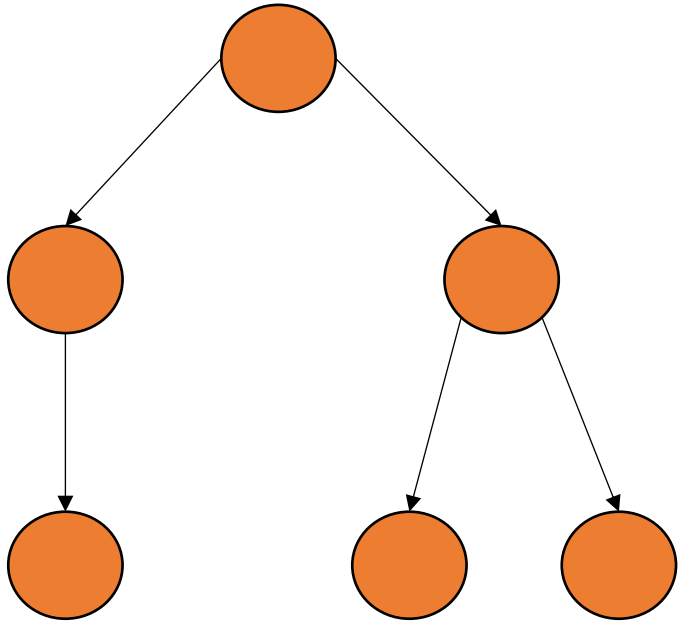
L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey , N. R. Tallent

HPCToolkit Overview

- Does not use instrumentation
  - gprof dilates by 82%
  - Intel's Vtune dilates by 31x
- Uses statistical sampling
- Supports call path profiling: attribute cost incurred to different contexts in which a procedure is called
- Supports collection, correlation of multiple performance metrics

Calling Context Tree

- Each vertex in a tree represents single procedure activation
  - Metrics recorded for that invocation
- Sample count recorded at each node
- Space-inefficient
- Tracing – Record the timestamp as well



Analysis

Correlation of performance metrics

- Mapping between object code and source code (“recovering program structure”)

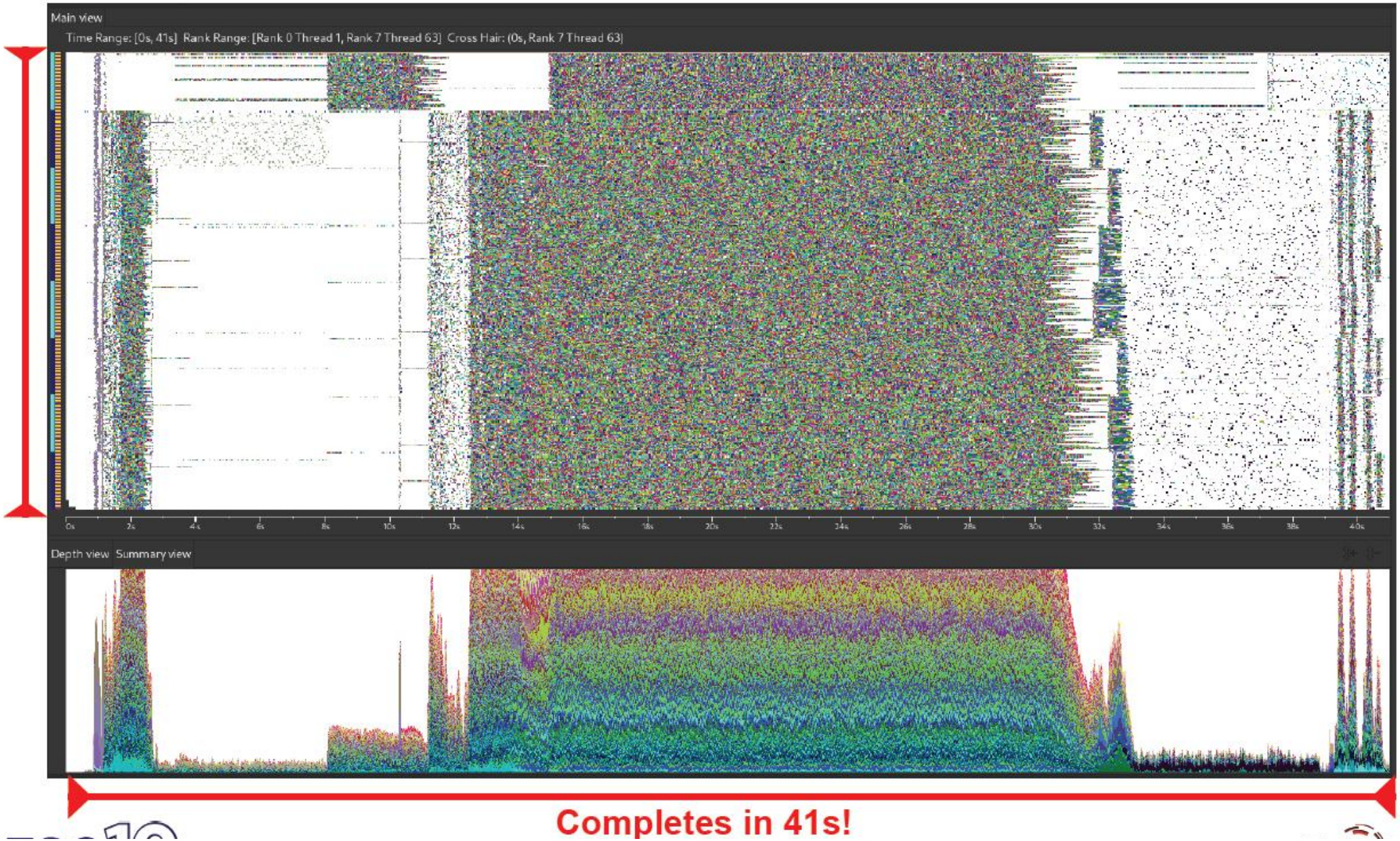
Compute useful metrics

- Knowing where the most cycles are spent
- Derived metrics

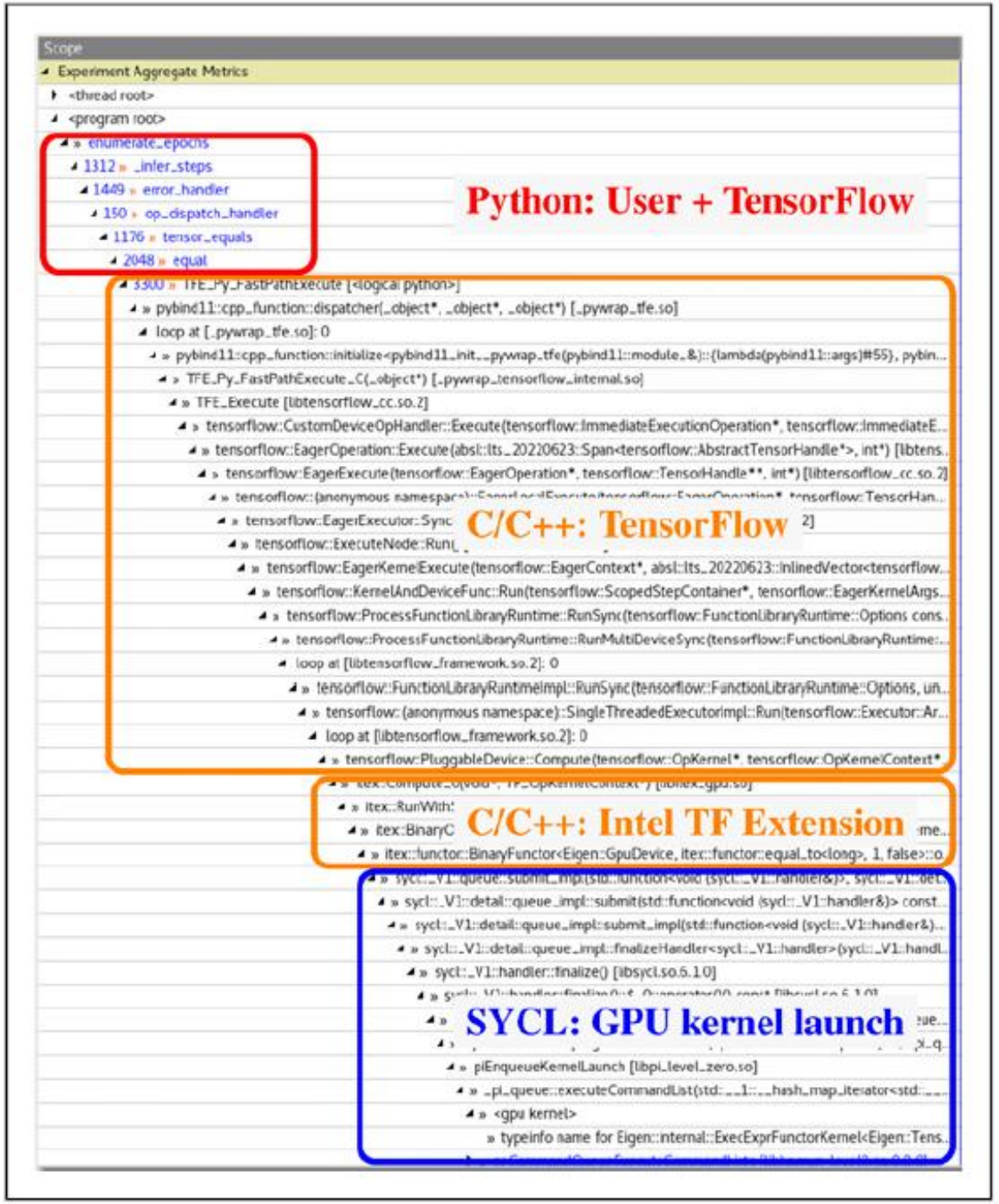
Scalability

- Combine call path profiles with program structure
- Estimate idleness, parallel efficiency etc.

hpcprof-mpi: Analyze Measurements of LAMMPS @ 2K threads + 2K GPUs



TensorFlow Application



hpcstruct Example: Analyze 7.7GB TensorFlow library (170MB text) in 77s