Performance Analysis for the Exascale Era: From Measurements to Insights

Martin Schulz
Lawrence Livermore National Laboratory

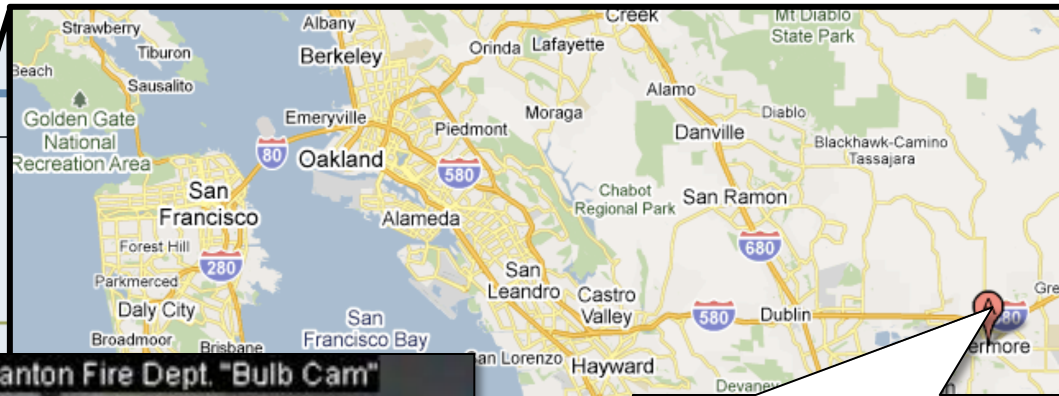
TU-Dresden, March 10th, 2016

http://scalability.llnl.gov/

LLNL-PRES-686269
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Where is Livermore?

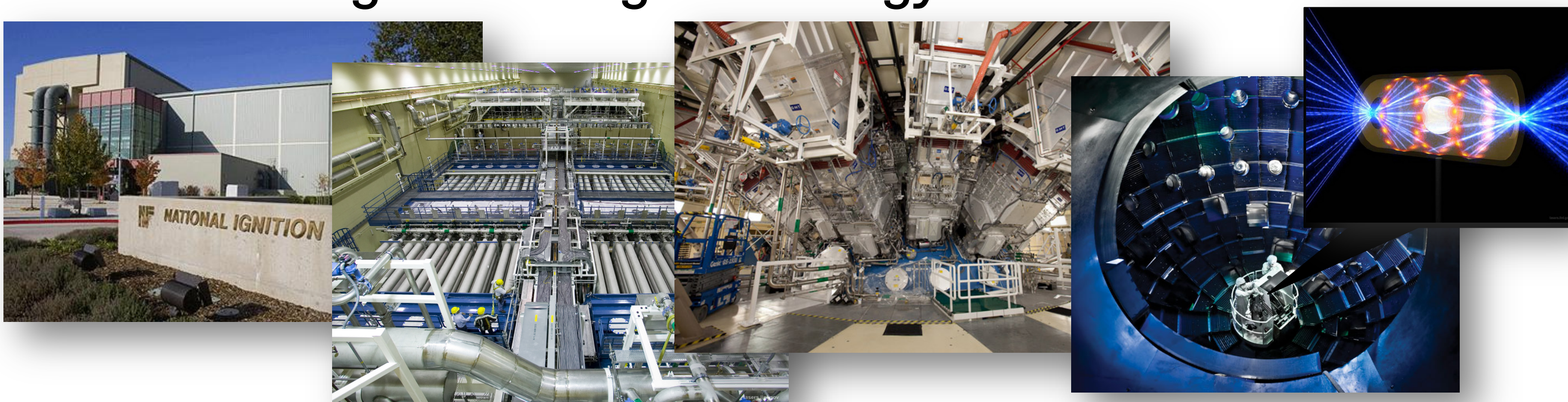


rancisco

- Wine
- Windmills
- A Lightbulb

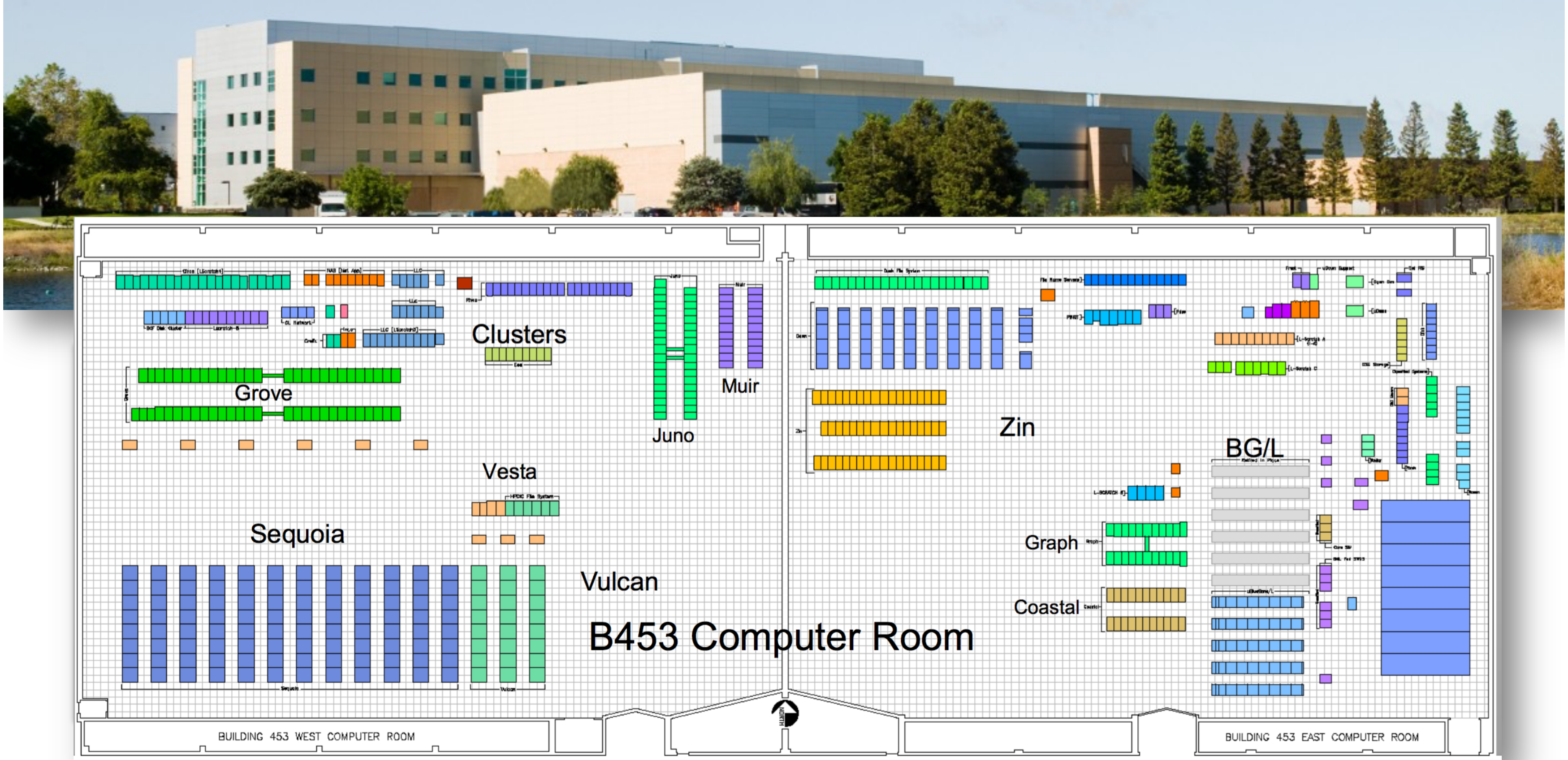
National Ignition Facility & Livermore Computing

- World's largest and highest-energy laser: Fusion research



- LC: more than 40 dedicated HPC systems in 4 rooms

Livermore Computing Complex



- 48,000 square feet of server floor space
- Up to 30 MW power available
- Liquid cooling for Blue Gene machines
- Power Usage Effectiveness (PUE) = 1.27

LLNL's BG/Qs: 20 PF Sequoia (plus 5 PF Vulcan)

New Machine: Sierra

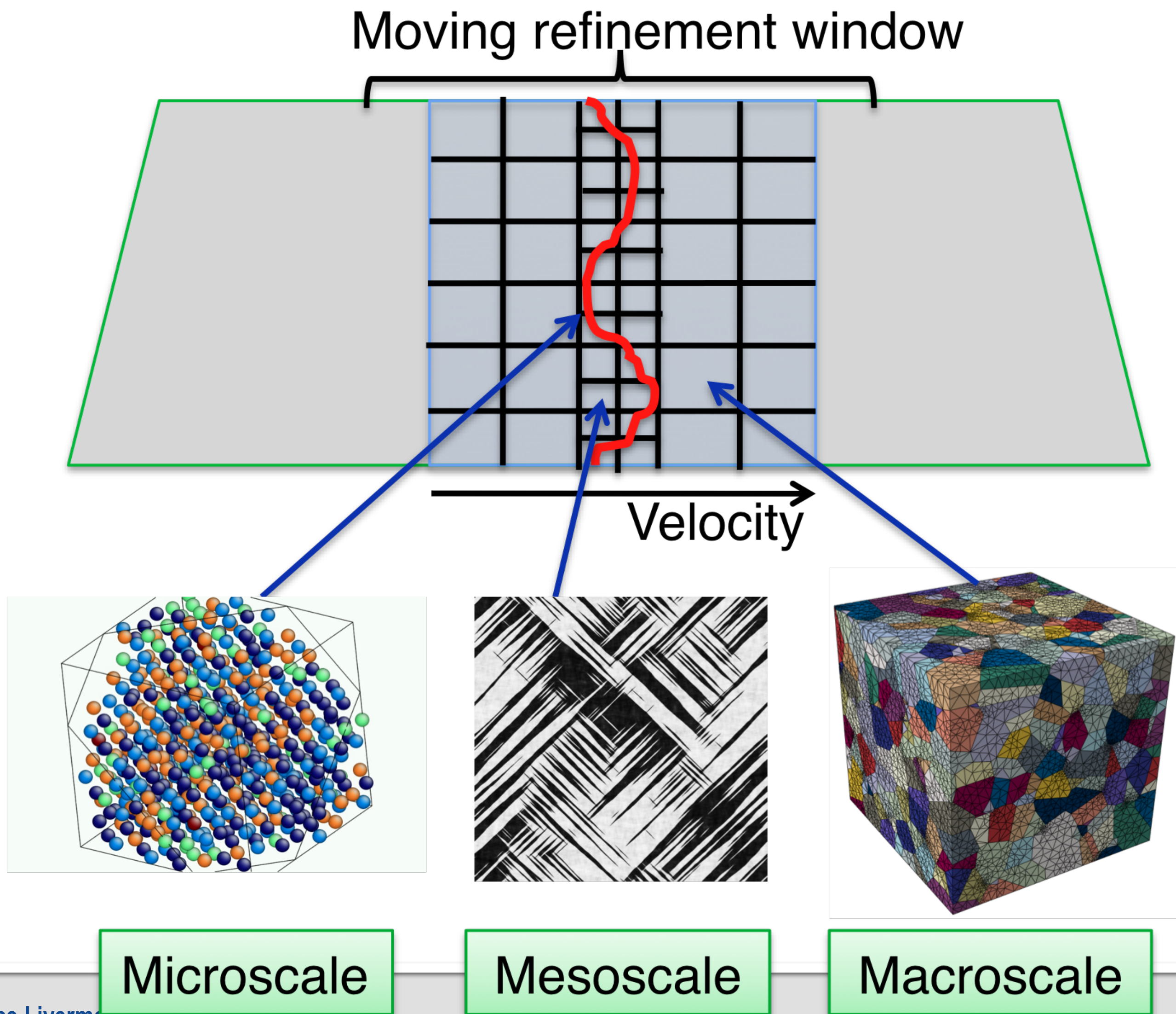
- Targeted for 2017/2018
  - CORAL collaboration between LLNL, ANL and ORNL
  - LLNL's Sierra had the same basic architecture as ORNL's Summit
- Vendor: IBM plus NVIDIA and Mellanox
  - IBM Power nodes plus NVIDIA Volta GPUs
  - Local NVRAM
  - Fat tree interconnect
  - 120-150 Pflop/s
  - 11 MW
- Path forward from Sierra to Exascale

Even If We Had an Exascale Machine ...

- We need applications that can exploit an exascale system
  - Utilize system resources
  - Perform in resource constraint environments (e.g., power)
  - Survive higher failure rates (silent and fail/stop)

- New applications will pose additional challenges
  - Not only larger scale, but new physics
  - More complex numerical algorithms
  - Uncertainty Quantification (UQ) and Scale-bridging

Scale Bridging Example: Material Science



ExMatEx
Source: Jim Belak

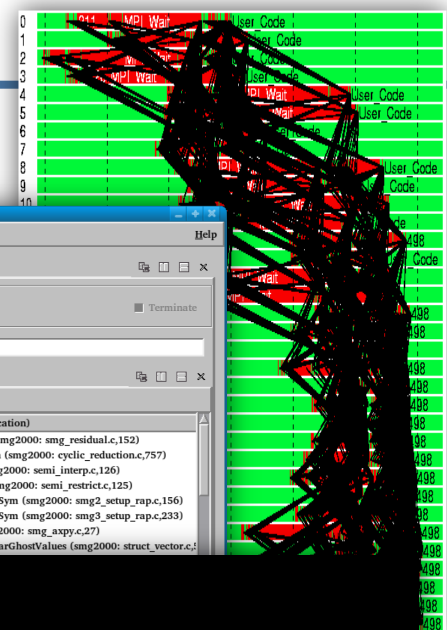
Even If We Had an Exascale Machine …

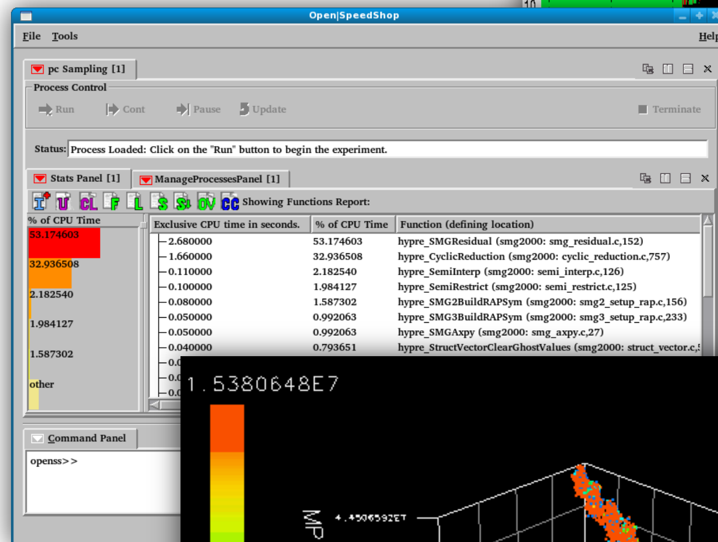
- We need applications that can exploit an exascale system
  - Utilize system resources
  - Perform in resource constraint environments (e.g., power)
  - Survive higher failure rates (silent and fail/stop)
- New applications will pose additional challenges
  - Not only larger scale, but new physics
  - More complex numerical algorithms
  - Uncertainty Quantification (UQ) and Scale-bridging
- Much will be left to the developer
  - New programming models
  - Complex heterogeneous architectures
  - High adaptivity at all system layers

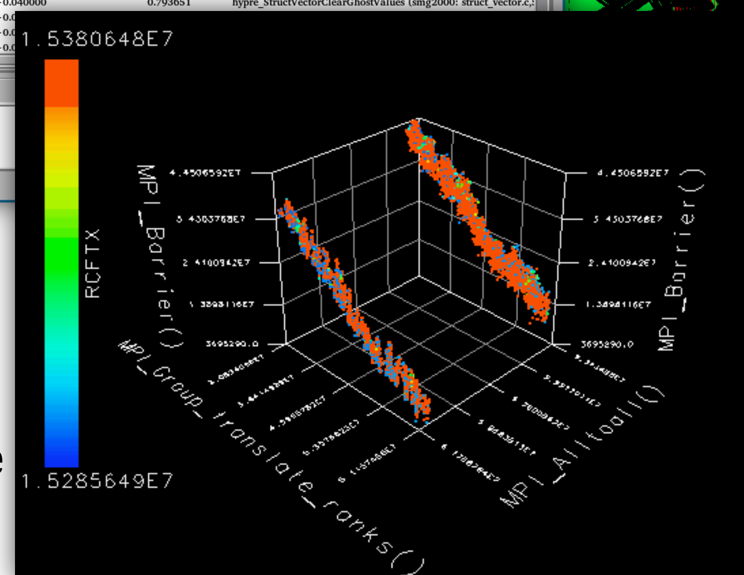
- Code developers will need sophisticated performance tools

Long History of Performance Tools

- Many tools can collect lot's of app. data
  - "Classic perf. tools" like OpenlSpeedShop, TAU, mpiP, HPCToolkit, Scalasca, Paraver, ompP or Vampir
  - HWC access (e.g., PAPI)
  - Architectural simulators
  - Performance models
- But …
  - Data volumes are increasing
  - Can't handle adaptivity (faults, tuning, OS, …)
  - System variability can invalidate results
- Second But …
  - Information often low level
  - Hard to match with application structure
  - Hard to understand for code developers



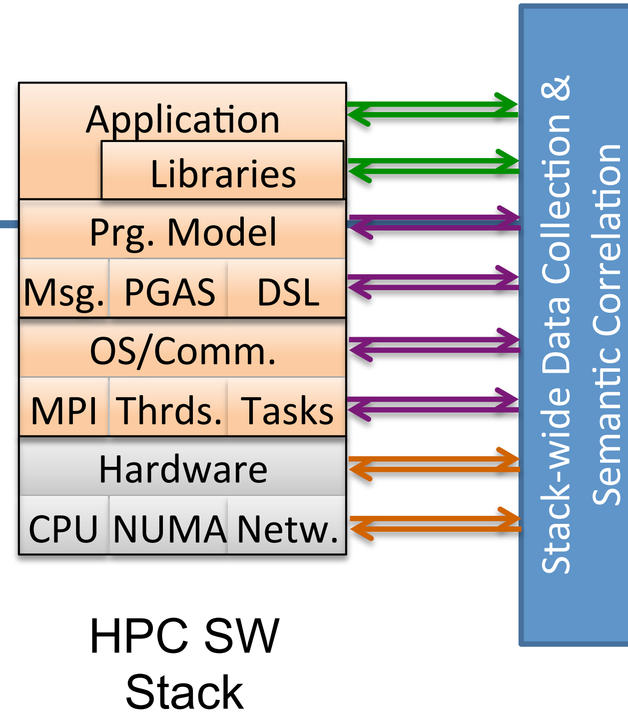




Need for a New Generation of Performance Tools

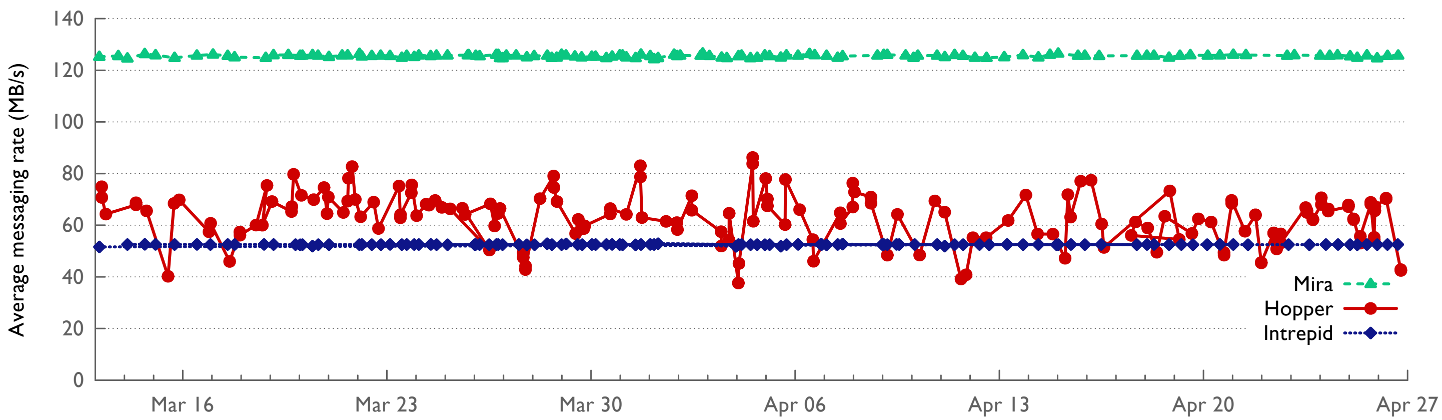
- Comprehensive data acquisition
  - Capture holistic view of the status of the software stack
  - Track system and application adaptations
  - Scalable data preprocessing and storage
  - Inclusion of facility and system data
- More intuitive ways to show data to end users: visualization
  - Mapping of performance data to application semantics
    - Using basic application information
    - Across new programming abstractions
  - Multiple views on the same data to allow for correlations
  - Close collaborations with the InfoVis/Vis communities helpful
- Critical pieces
  - Extract the necessary context
  - System/facility wide monitoring
  - Visualize context to provide new views on performance data

Holistic Data Acquisition

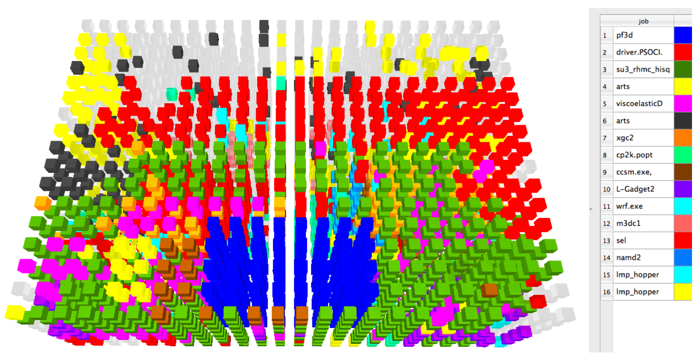


- Capture data in entire stack
  - Metadata to explain results
  - Capture adaptivity in the system
  - Information to map measurements
  - Correlation across layers
- Low-level information
  - From CPU/MSR, board, accelerator
  - OS can provide valuable data
- Extract information from programming model/runtime
  - Need ability to map performance data to programming constructs
    - Programming model specific APIs (OMPT, MPI_T, OCR-T, …)
  - Need interfaces into the runtime stack
    - Introspection abilities, especially for dynamic adaptations
- Need facility wide and continuous monitoring
  - Single performance experiments from limited sources are no longer reliable

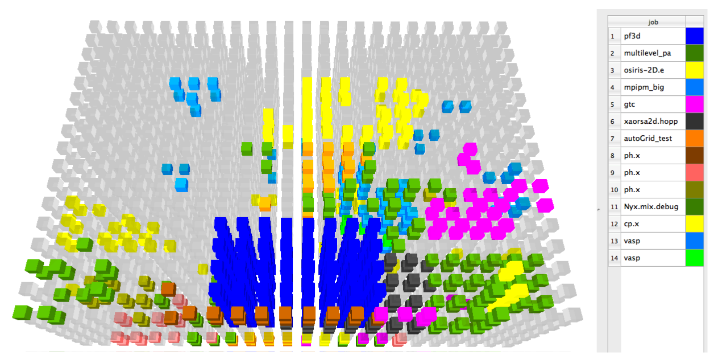
Example of Variability: Network contention



Performance variability over time with and without network congestion. Blue Gene systems (Mira & Intrepid) have isolated per-job network partitions, while Cray XE6 systems use a shared network.



Slow run of pf3d on Cray XE6 system.



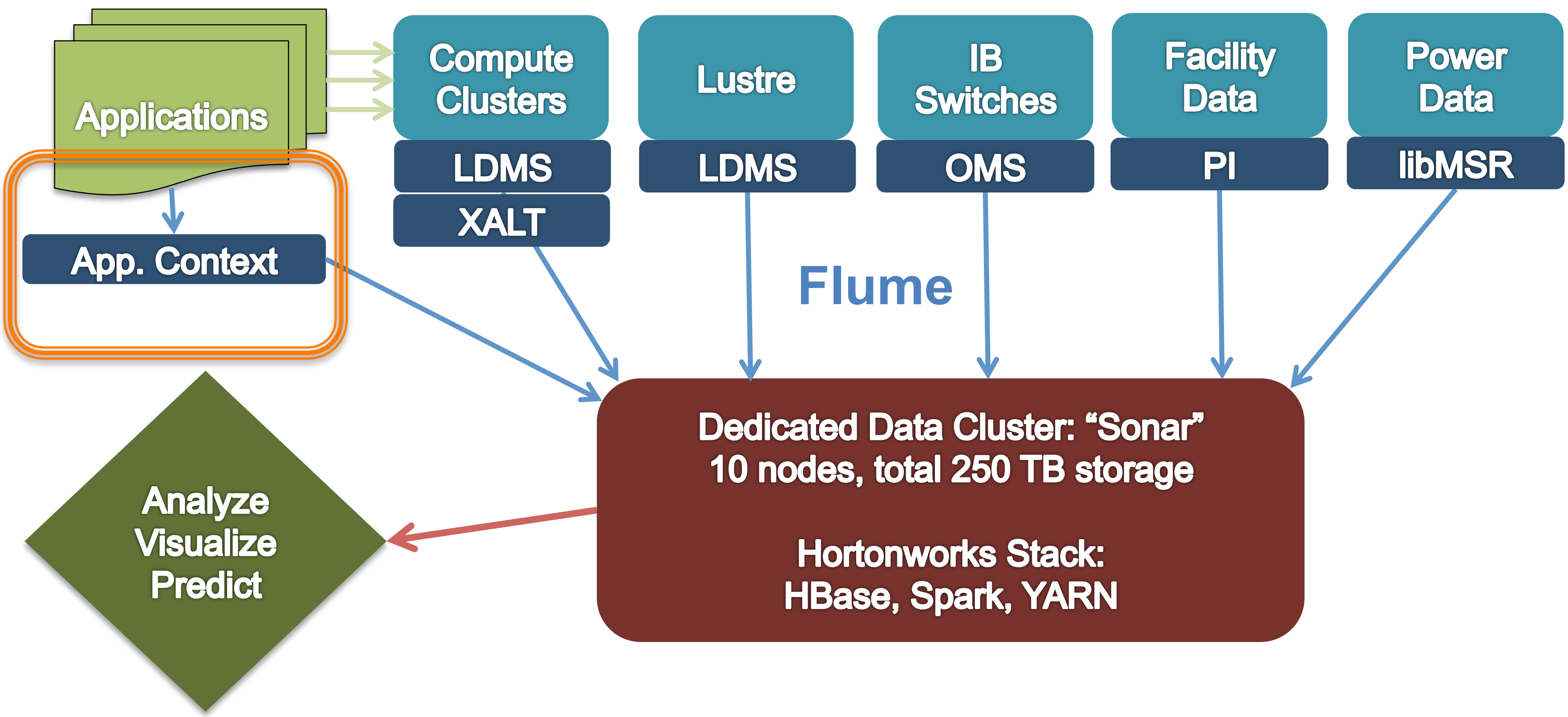
25% faster messaging rate without congestion.

Variability Concerns

- Network contention
- OS Noise
  - Non reproducible runs
  - Memory layout
- Manufacturing variability leads to power variations
  - Under a power cap these lead to performance variability
  - ~10% on Sandybridge, up to 25% on Ivybridge
- External factors
  - Temperature fluctuations
- File system performance

➢ Makes comparing two runs increasingly hard
  - ➢ Performance analysis is turning into statistical analysis
  - ➢ Small improvements in performance eaten up by variability
  - ➢ Need to understand and track execution context for many runs

Multi-level, Site-wide Monitoring is Necessary to Accurately Characterize Behavior



Clusters send data to the database to be analyzed, visualized, and used to make predictions for future runs.

Capturing Application Context

- Context: program and system state
  - Spread across the software stack
  - Must be contributed independently by different modules
  - Should be used to annotate measurements

Context

Application

- call path, iteration count, calculation phase

Library

- AMR level, library phase, solver level

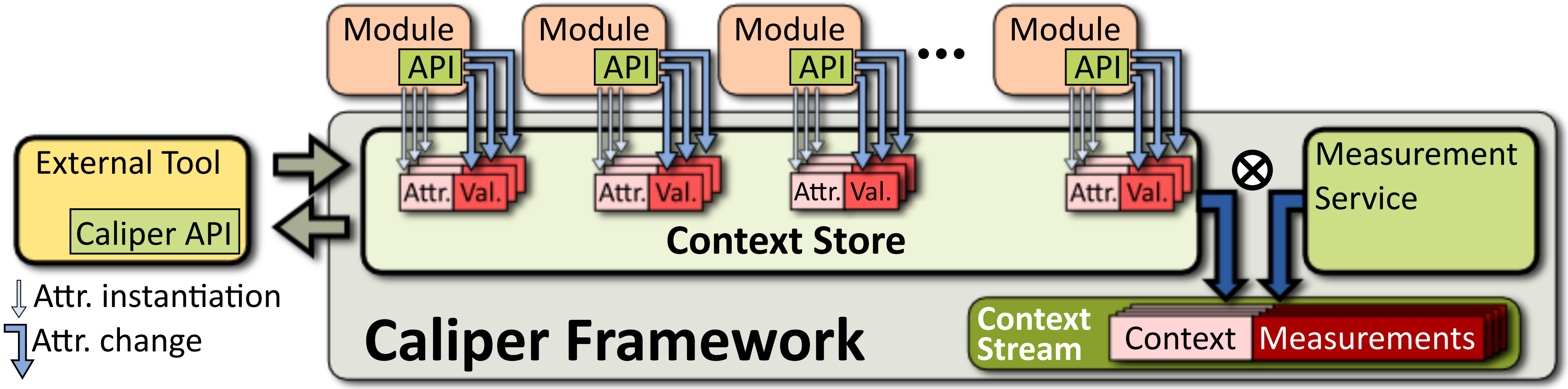
Runtime system

- thread / task id, runtime state

Measurement data

FLOPS, time, memory accesses

The Caliper Approach



- Modules define and update attributes independently
  – Attribute:Value pairs
- Caliper maintains global context buffer
  – Process global
- Caliper takes *snapshots* of current context + measurements
  – Written to context stream or given to third-party tool

Annotation Interface

- `cali::Annotation`
  - Encapsulates attribute
- `begin()`
  - Append new value
- `set()`
  - Set (overwrite) value
- `end()`
  - Remove last value

```
#include <Annotation.h>

int main(int argc, char* argv[])
{
    cali::Annotation phase_ann("phase");

    phase_ann.begin("main");
    phase_ann.begin("init");
    // Perform initialization
    initialize();
    phase_ann.end(); // ends "init"

    phase_ann.begin("loop");

#pragma omp parallel for
   for (int i; i < MAX; ++i) {

cali::Annotation("iteration").set(i);
        do_work(i);
    }

    phase_ann.end(); // ends "loop"
    phase_ann.end(); // ends "main"
}
```

Measurement Services

Timer

- Timestamps, time durations

Ompt

- OpenMP tools interface: get OpenMP runtime status

Callpath

- Get call path using stack unwinding

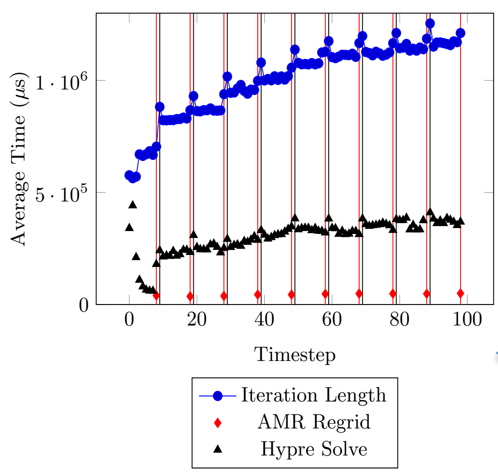
perf event

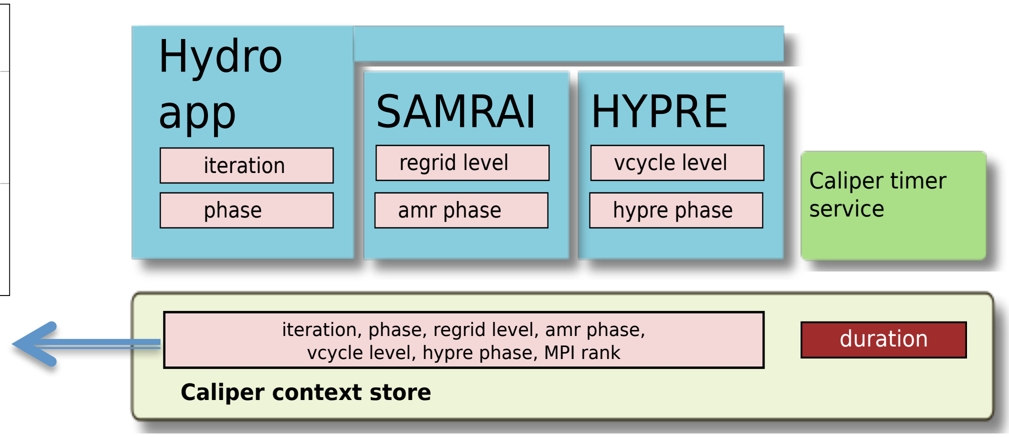
- Memory access info from Intel PEBS counters

Caliper Use Cases

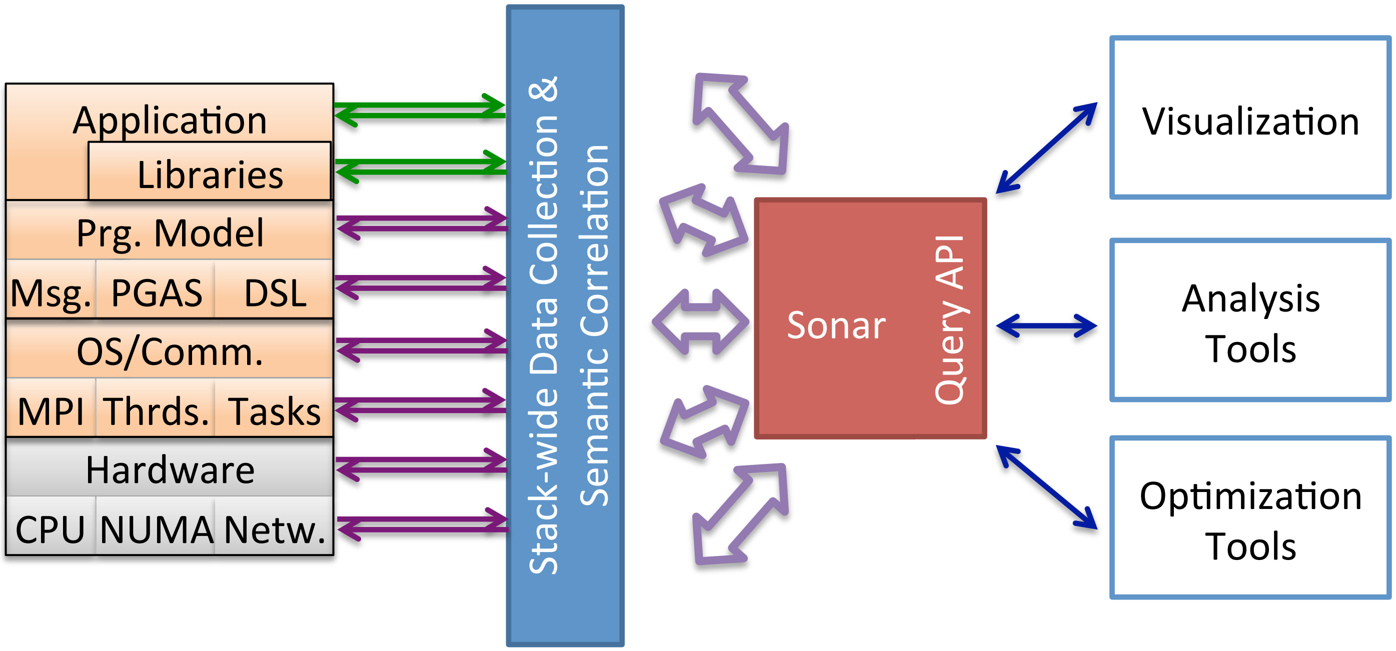
- Replace code specific timer libraries
  - Expose measurement intervals via Caliper
  - Simple timing service provide day to day metrics
  - More complex tools can pick up the same context
- Example: large physics at LLNL
  - Multiple libraries independently instrumented
  - Correlations across modules/libraries

Example: Combine data from multiple components in an IC code

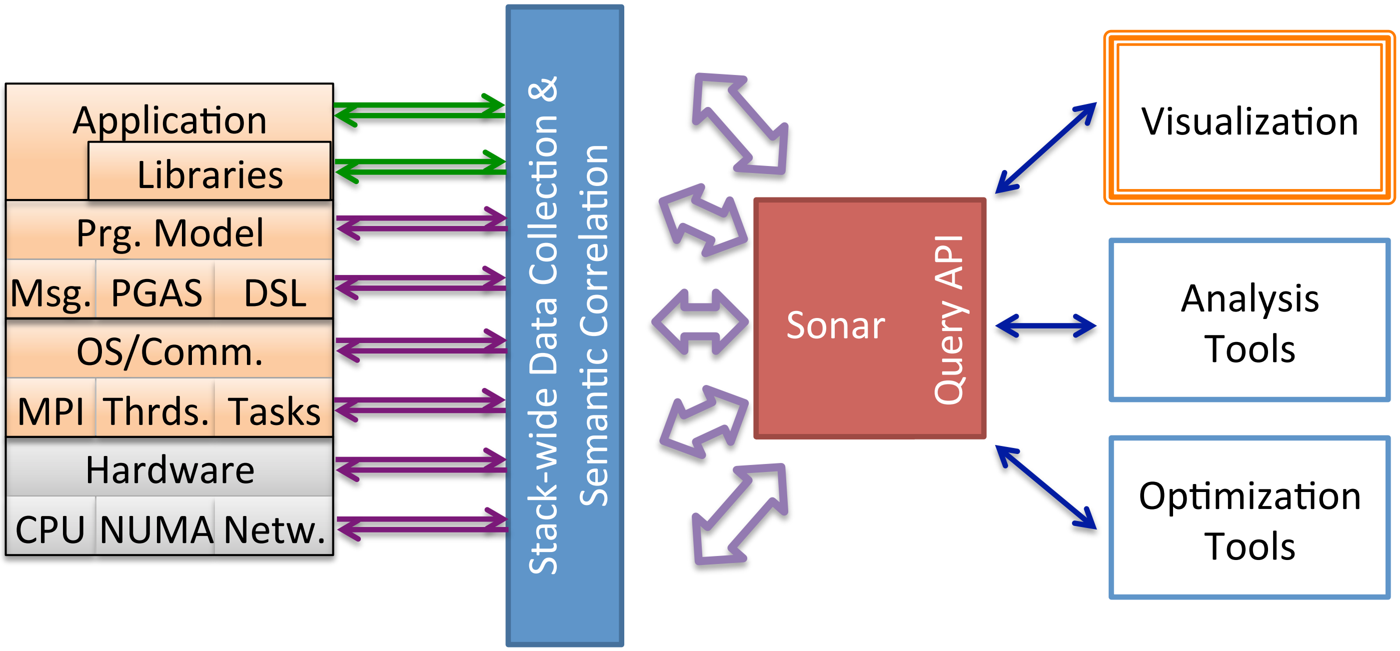




From Information to Insight



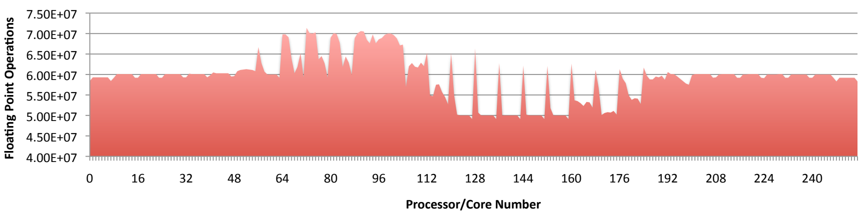
From Information to Insight



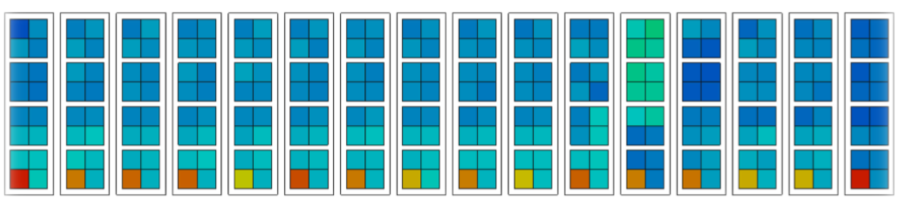
- Visual exploration useful to find new phenomena
  - Collaboration with SciVis and InfoVis communities
  - Goal: increase intuition for tool user
  - Map data from measurement to analysis/visualization domain

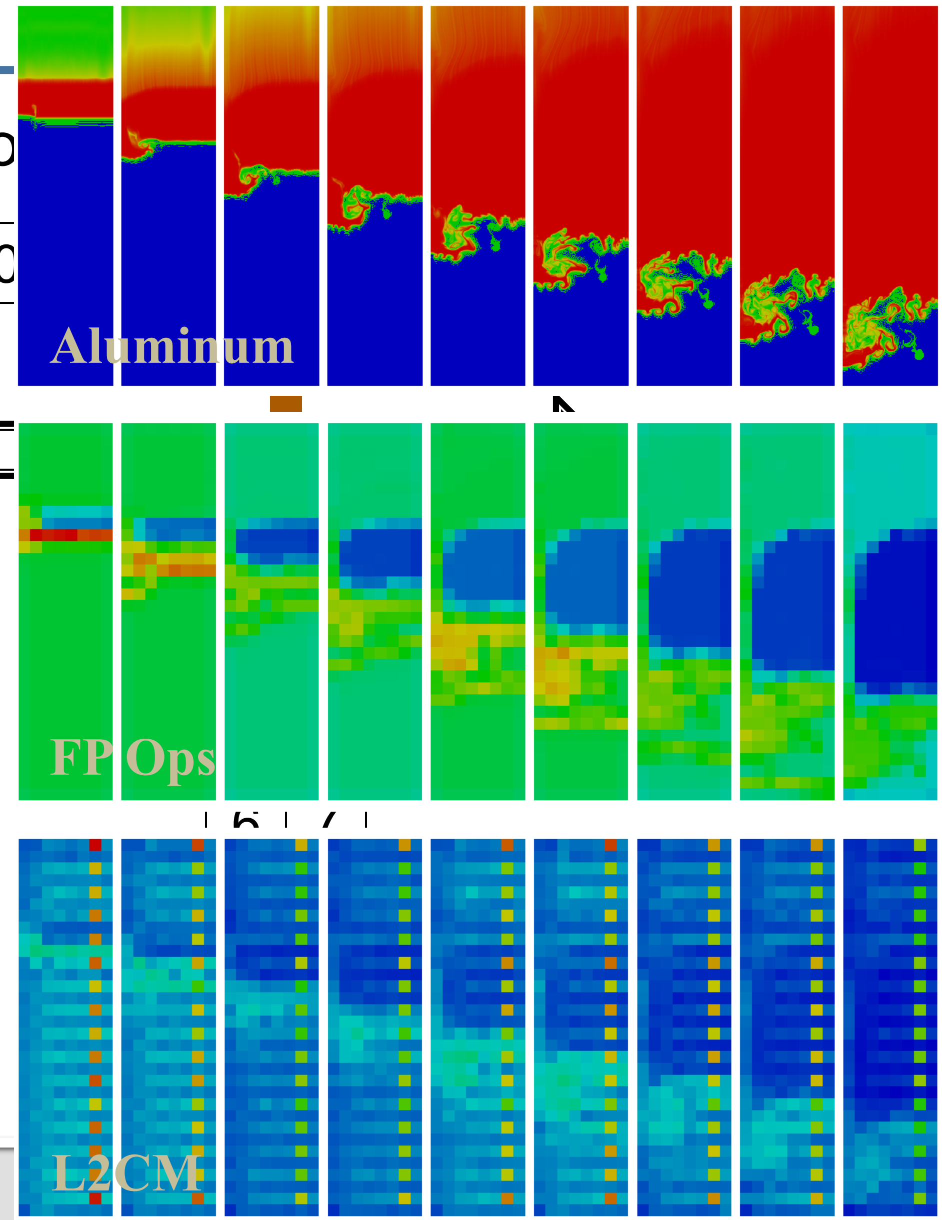
Picking the Right Analysis/Visualization Domain

- Example: Performance data o
  - Dense matrix on 8x32 cores
  - Floating point operations



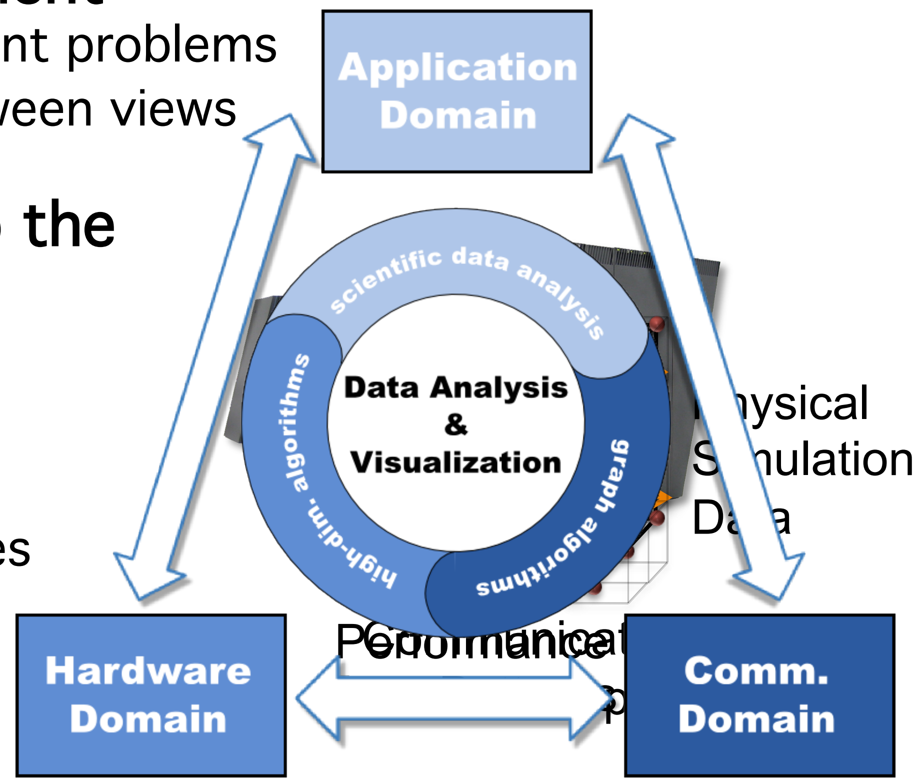
- **Second Effect**
  - Visible in dots in L2CM
  - Not related to physics
  - Map to same core on each node





Correlating Performance Domains

- **Single view on data is insufficient**
  - Different perspectives for different problems
  - Need to support correlation between views
- **Map data from one domain to the one of the other domains**
  - Comparable data
  - Enable correlation
  - Understand interactions
  - Access to visualization techniques
- **Increase intuition for users**
  - Display data in domains familiar to users
  - Make abstract measurements tangible

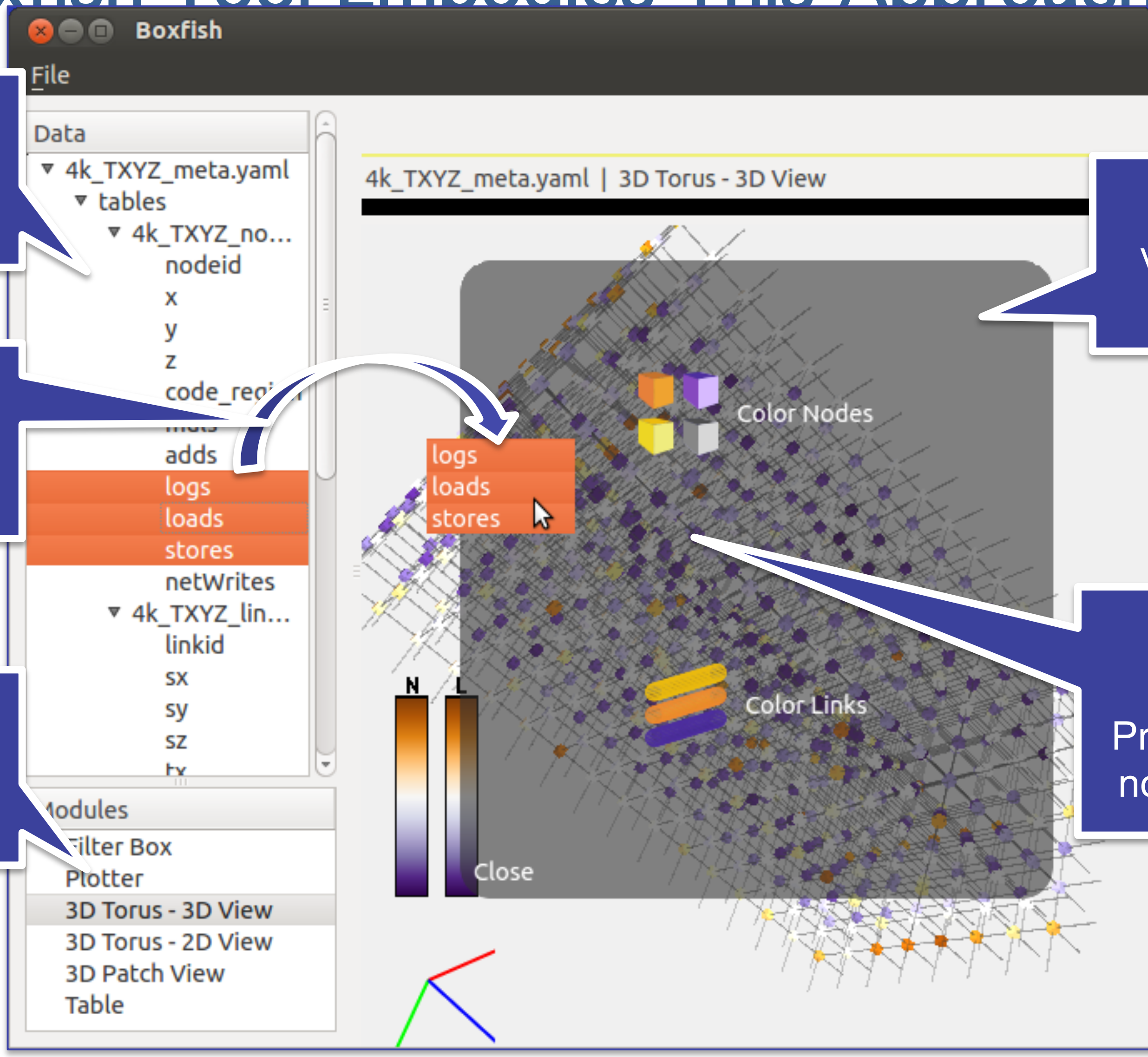


The Boxfish Tool Embodies This Approach

Input data from multiple measurements

Drag selection to map data to visualization

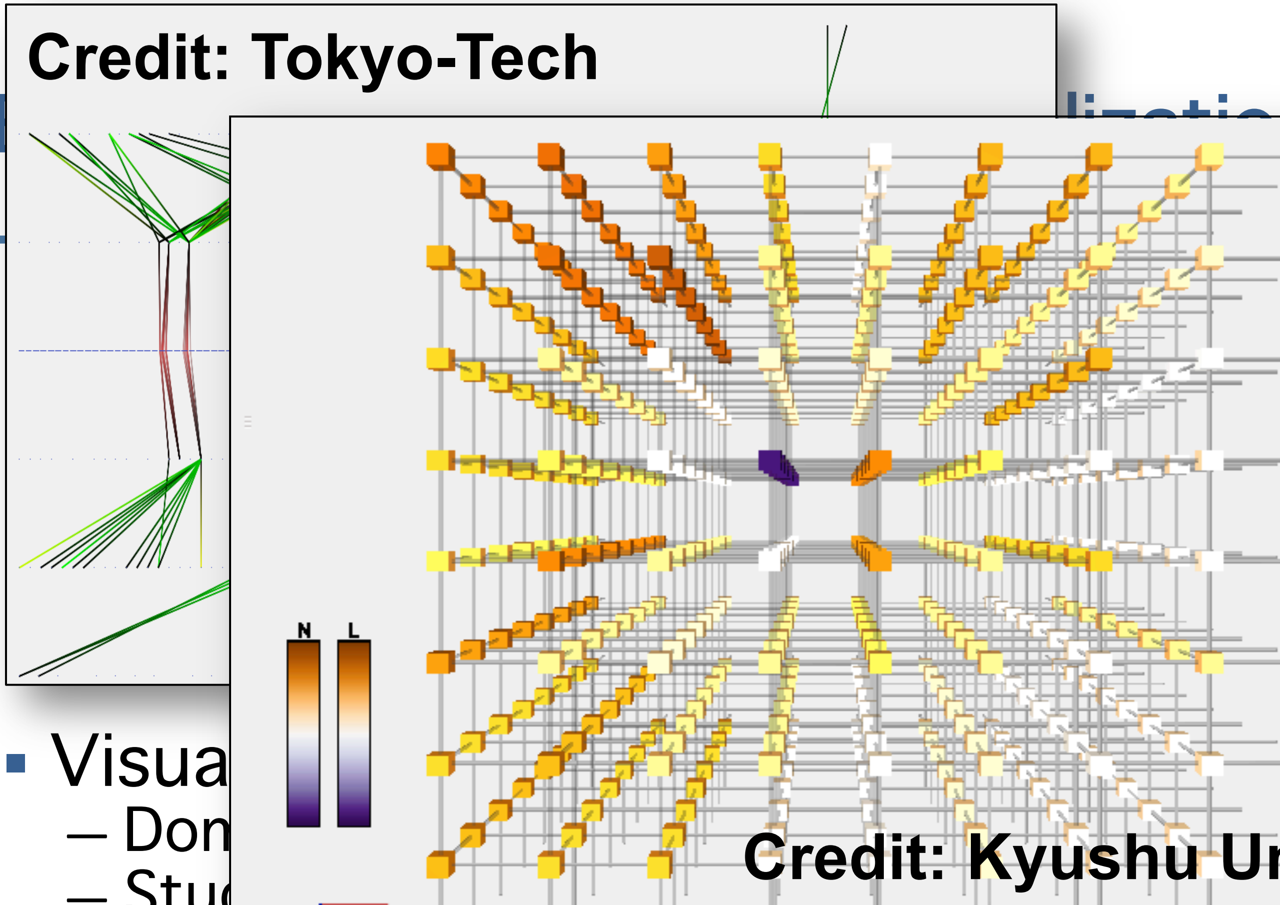
Available Visualization domains

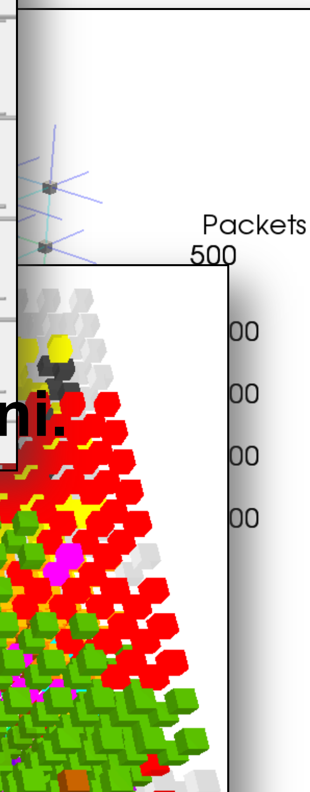


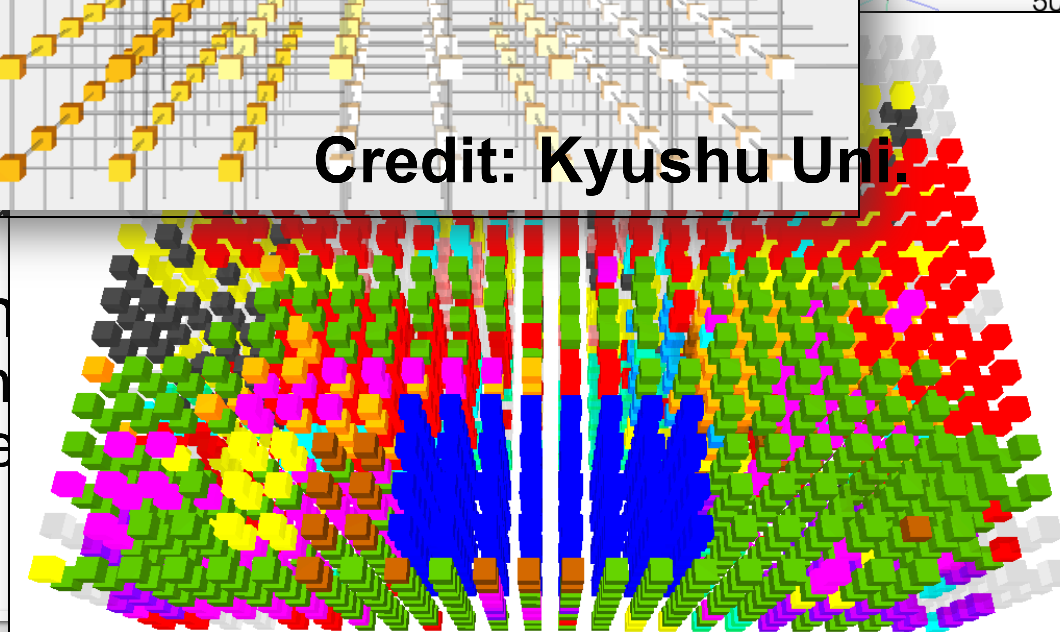
Selected visualization: 3D Torus

Choice of mappings: Present data on nodes or links?

lization

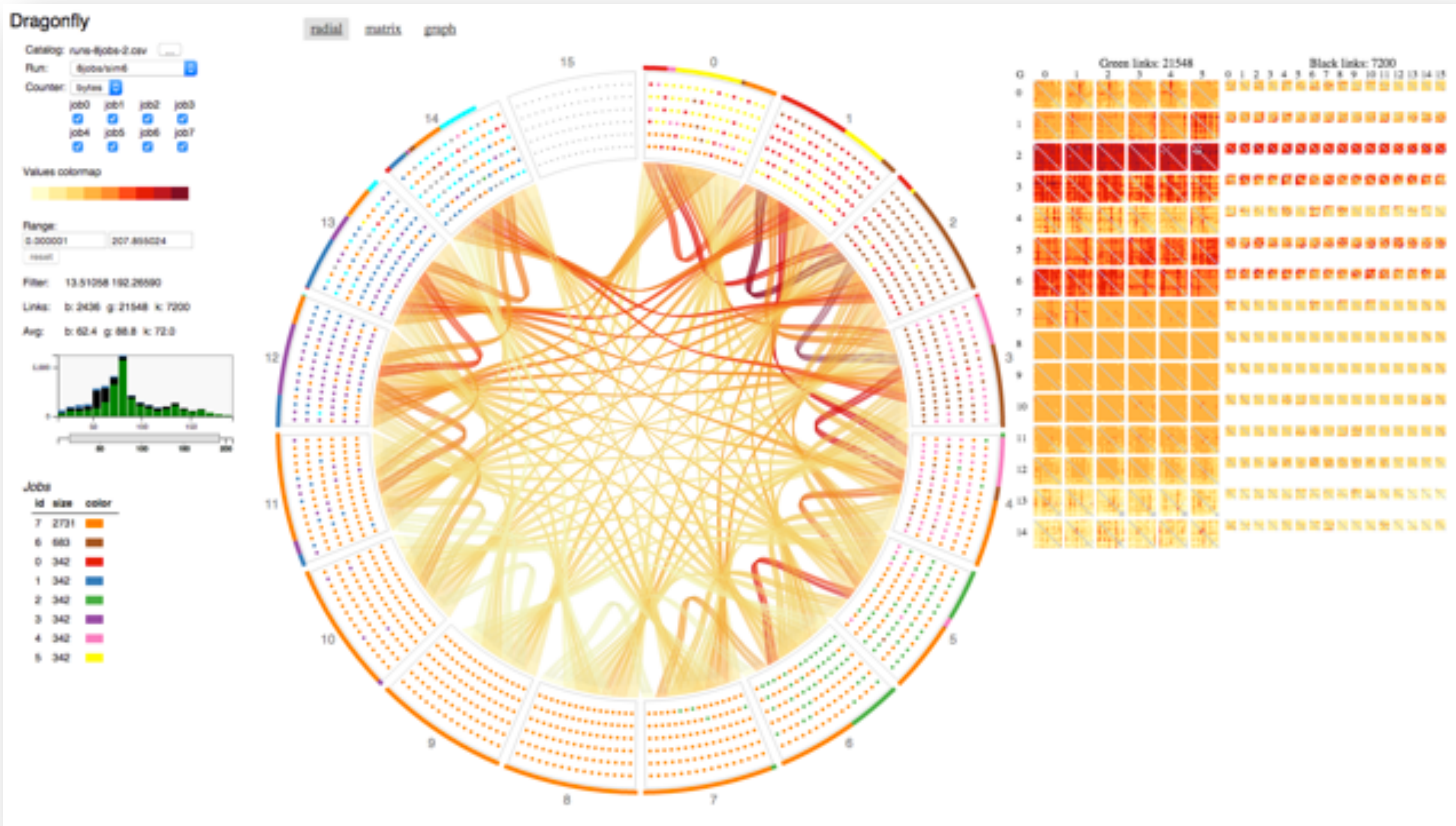




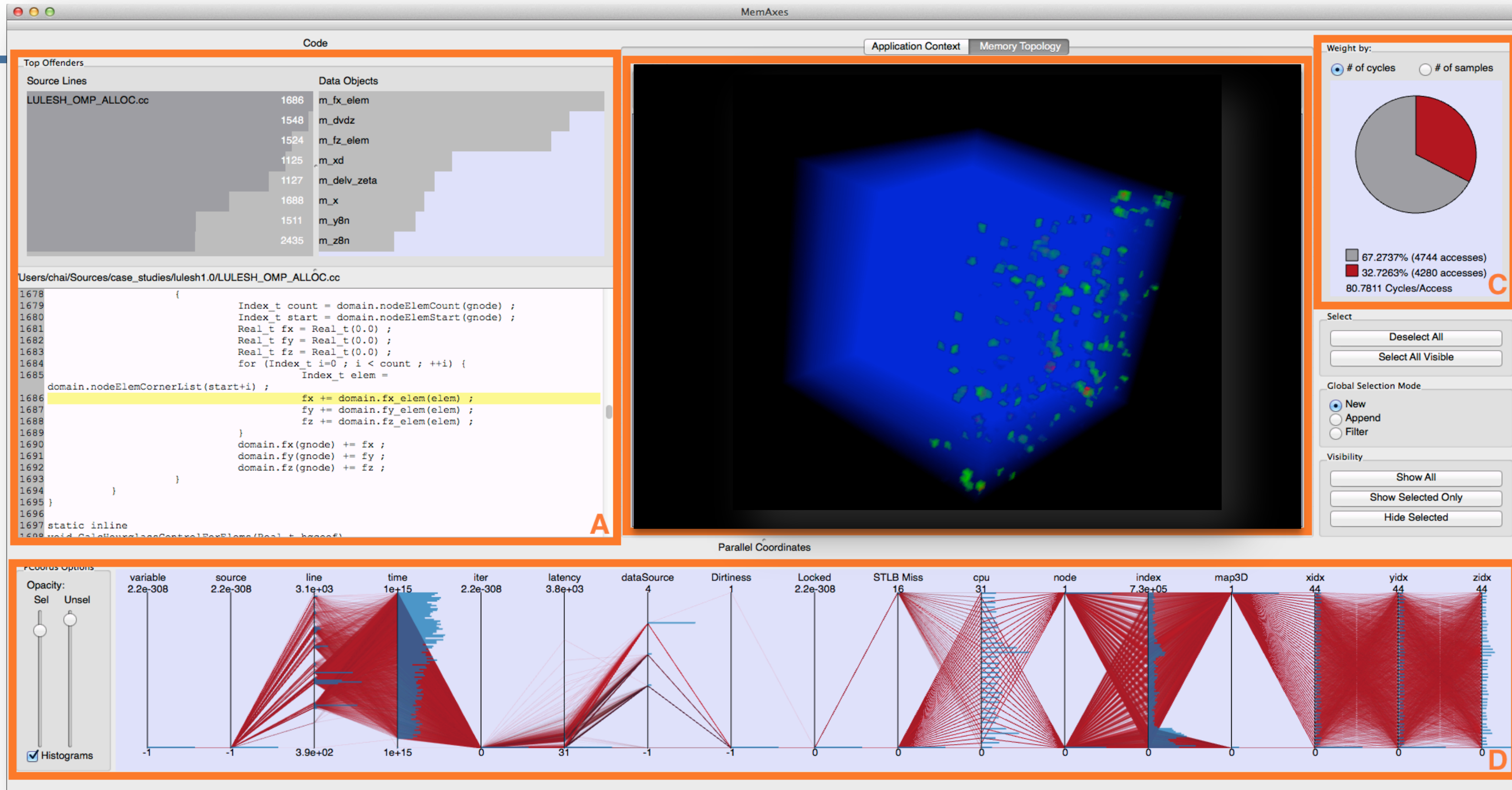


- Visua
  - Don
  - Stud
- Power optin
  - Turn off un
  - Boxfish use

Visualizing Dragonfly Network



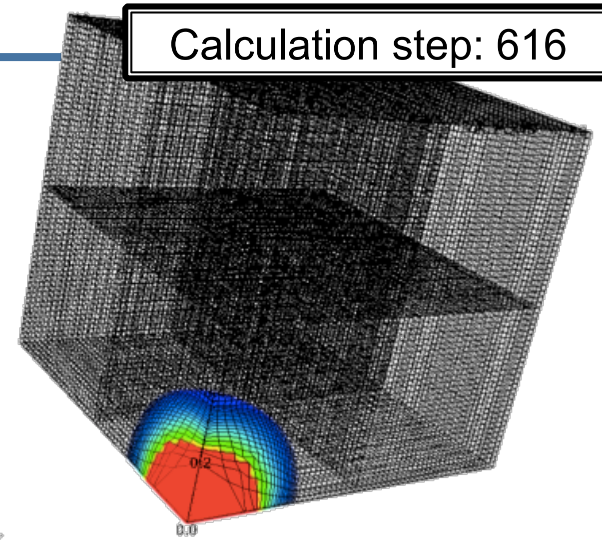
MemAxes: Visualizing Memory Traffic

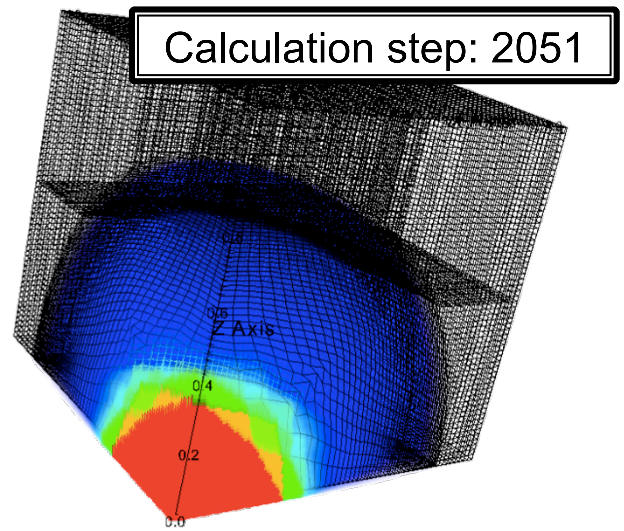


- **Shows data mapped to of code and machine characteristics**
  - Hardware topology
  - Location within the mesh
  - Code locations

MemAxes: Details and Case Study

- Efficient Sampling using PEBS
  – Access to cache miss address
  – Ability to map to data structures (and more)
- Collection of application metadata
  – Tracking of user allocations
  – Parsing of debug symbols for code mappings
  – Integration with Caliper context
- Case Study: LULESH
  – Shock Hydrodynamics challenge problem
  – Solves Sedov problem
  – Unstructured hex mesh
  – Implemented in a wide range of models (incl. OpenMP, which we use here)

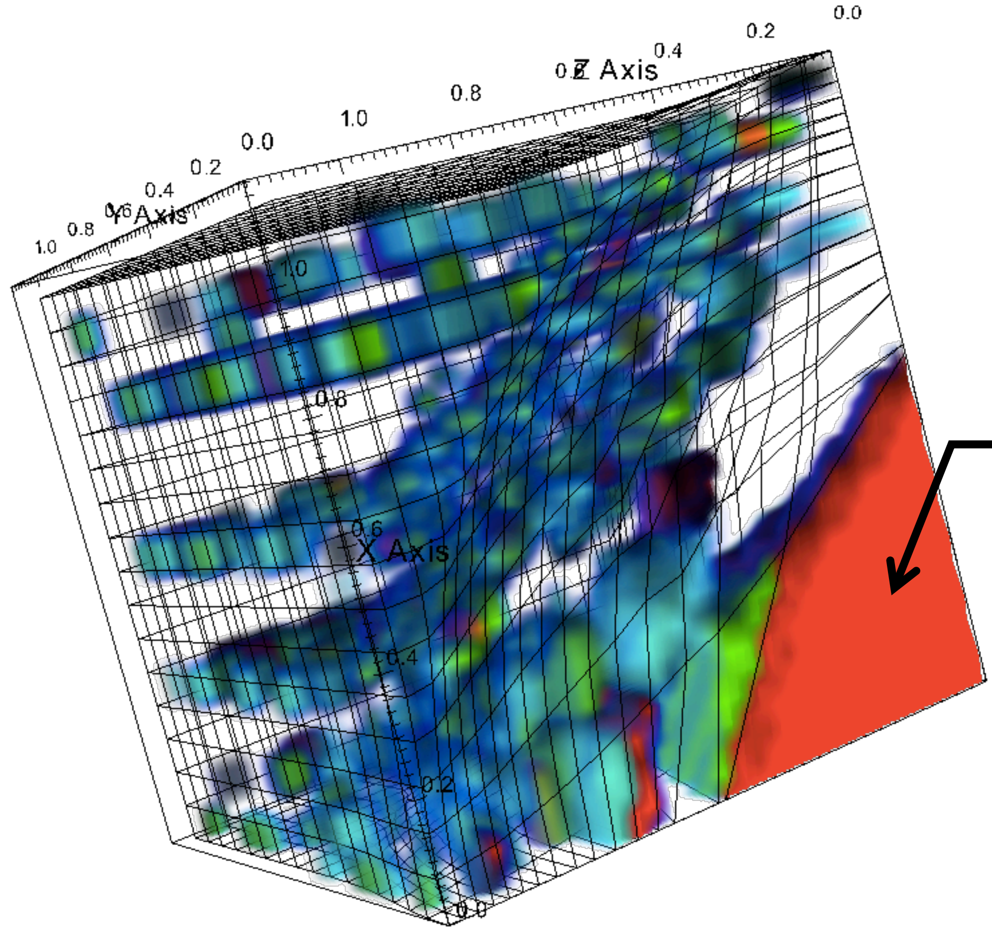




Cache Misses → LULESH Unstructured Grid

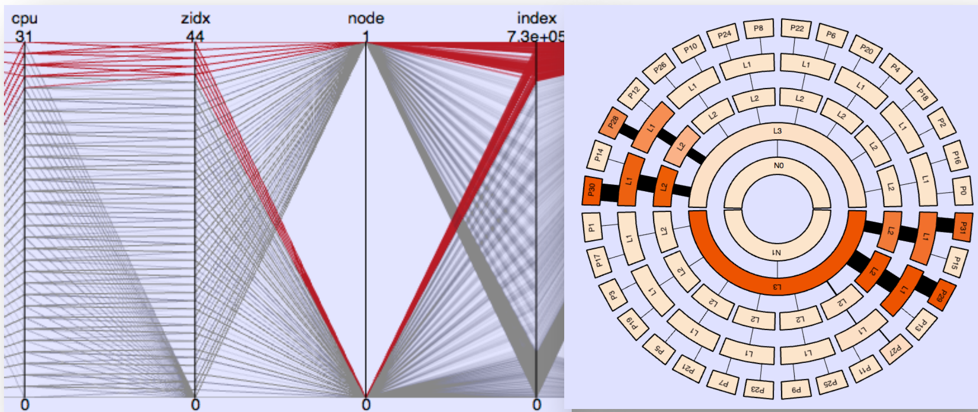
Total Cycles

1000.
889.3
740.4
511.9
0.000

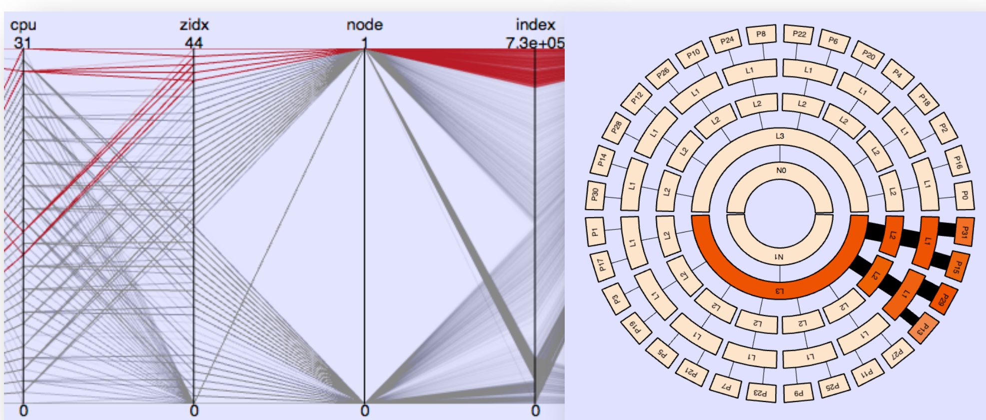


Compulsory cache misses at first element

Case Study: Optimization of On-node Locality



Default thread affinity with poor locality

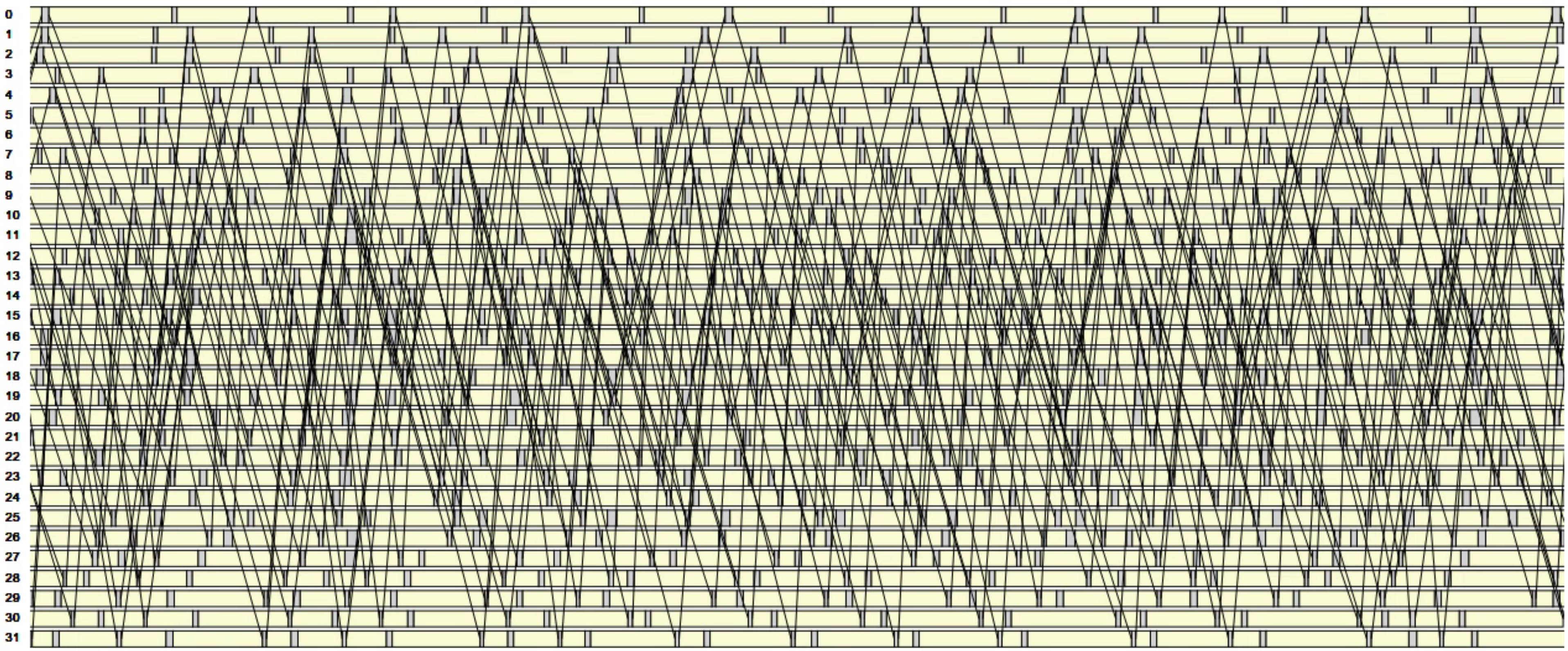


Optimized thread affinity with good locality

- Parallel coordinates view shows correlation between array index and core id in LULESH
- Linked node topology view shows data motion for highlighted memory operations
- A contiguous chunk of an array is initially split between threads on four cores
- Using an optimized affinity scheme, we improve locality
- Performance improved by 10%

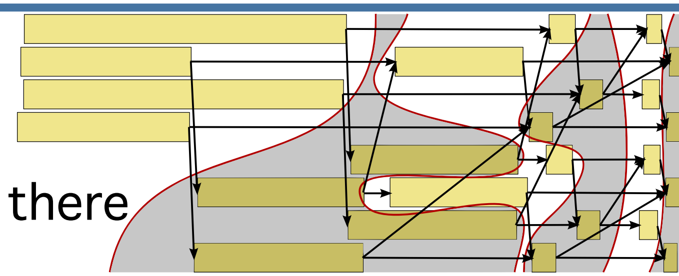
Ravel: Making Message Traces Readable

- Trace visualization is a helpful tool to show message details
  - Physical timeline view can create a hairball
  - We need new techniques to unravel this hairball -> virtual time

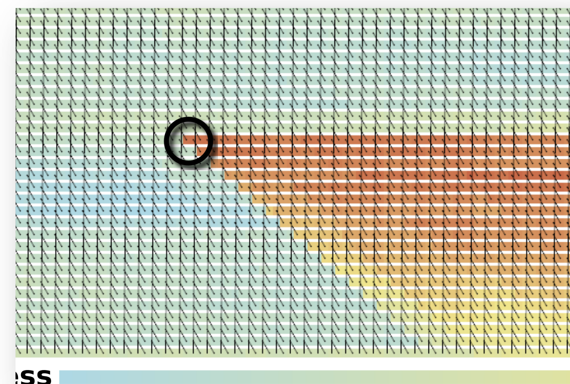


Ravel: Visualizing Traces in Virtual Time

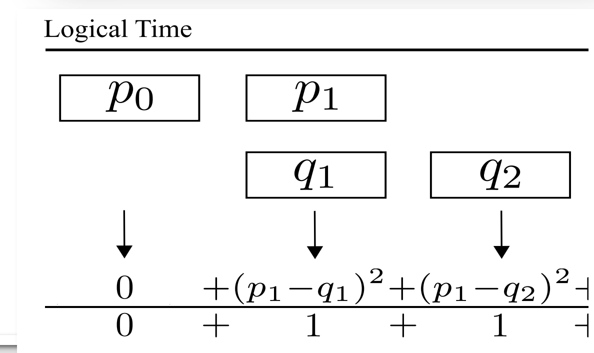
- Step 1: Identifying time slices
  - Concept of connected components
  - Start with send/recv pairs and grow from there
  - Heuristics on when to stop growing



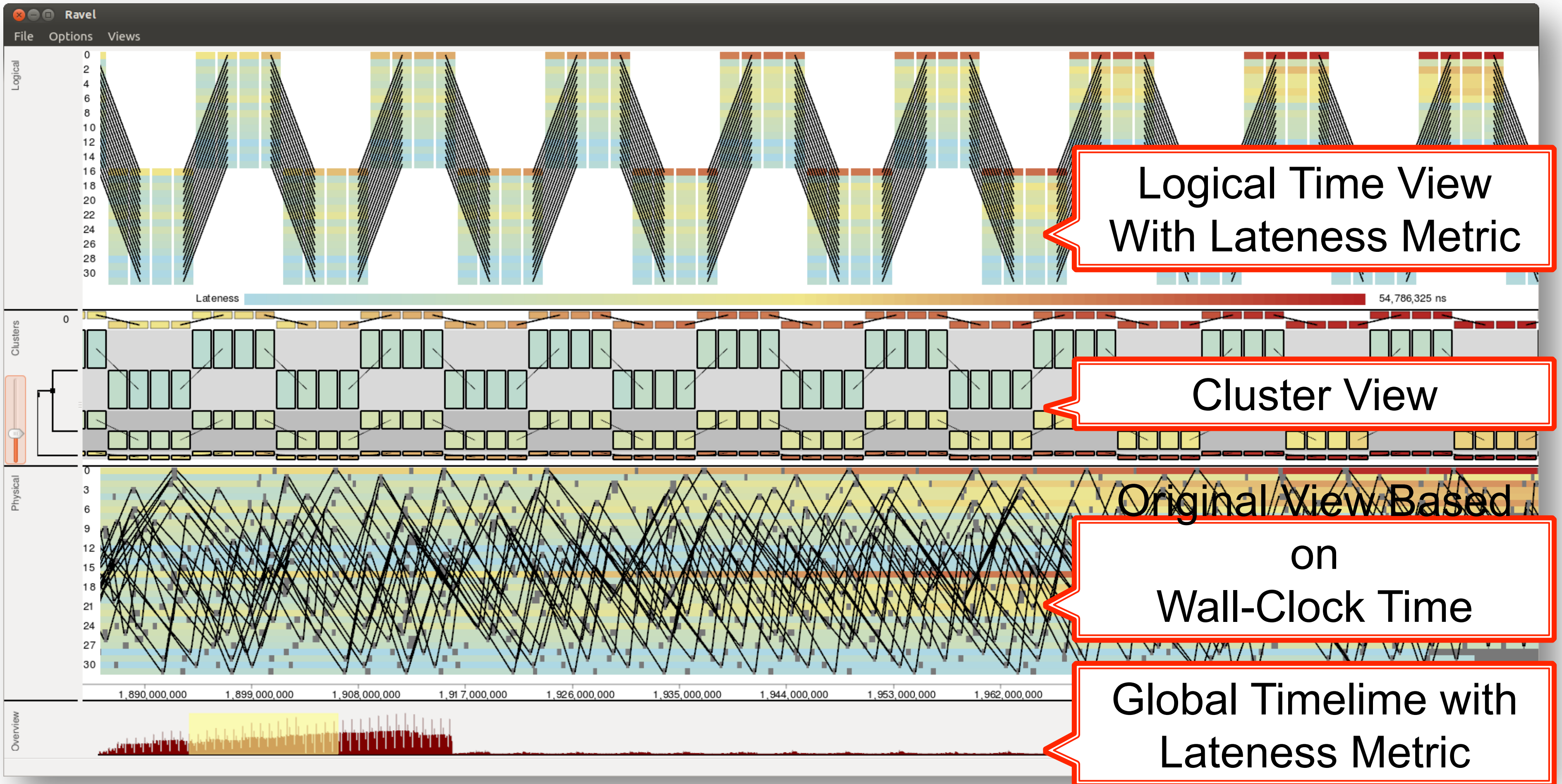
- Step 2: Mapping timing metrics
  - Mapping to virtual time loses physical time
  - Reintroduction of time using lateness metric
    - Time difference to end of aligned phase
    - Shows propagations of delays



- Step 3: Cross process clustering
  - Aggregate traces with similar lateness
  - Use of representative traces to show data

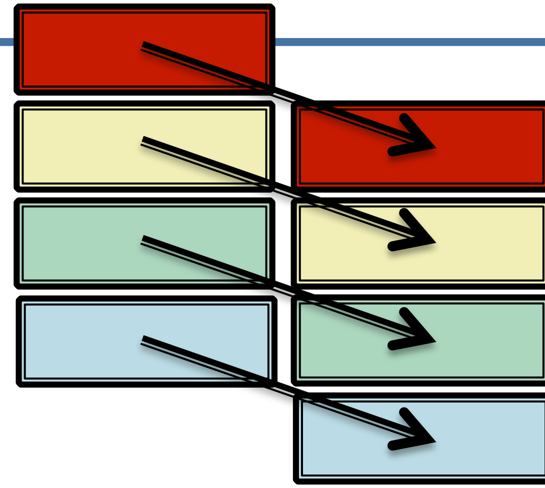


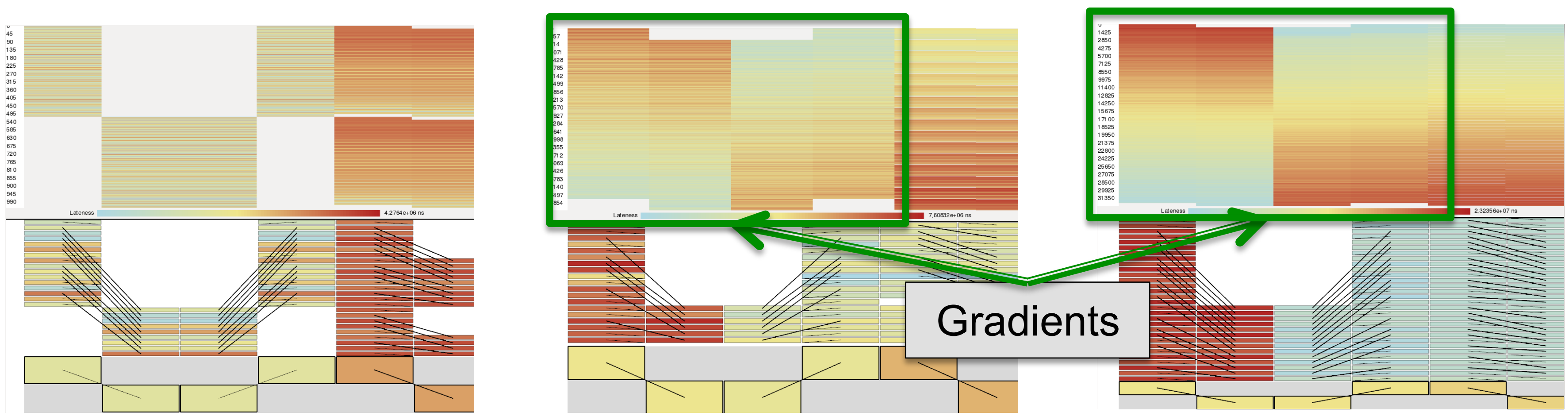
Ravel: Trace Visualization Using Logical Time



Case Study: Optimizing Communication Patterns

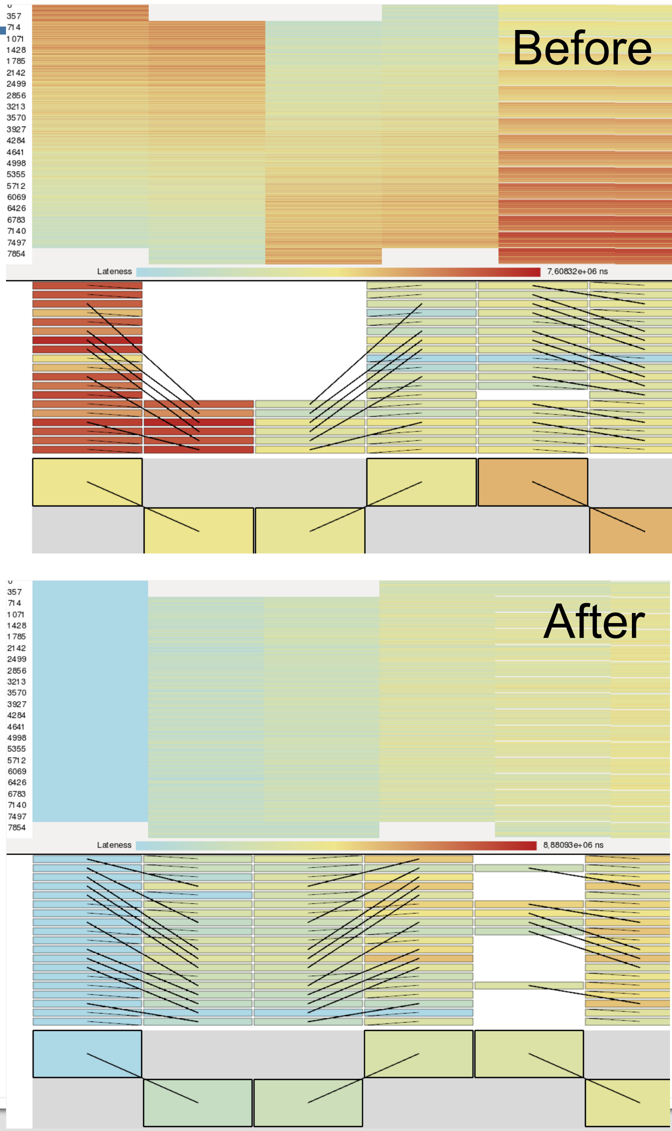
- Communication benchmark for physics simulation
  - Several process counts
  - Traces at process counts show inverting gradient of lateness

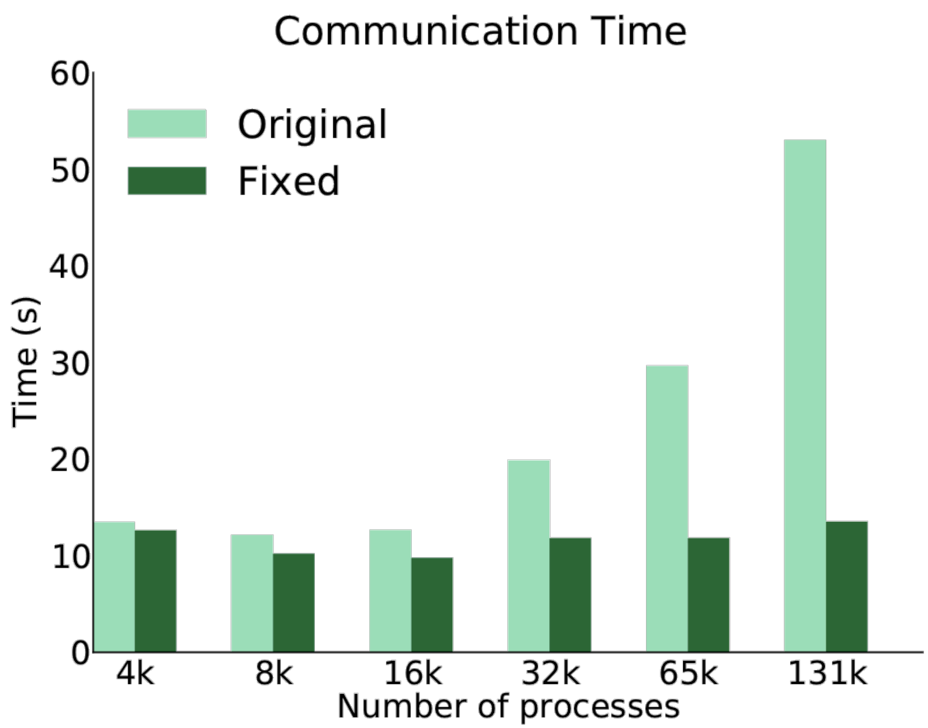




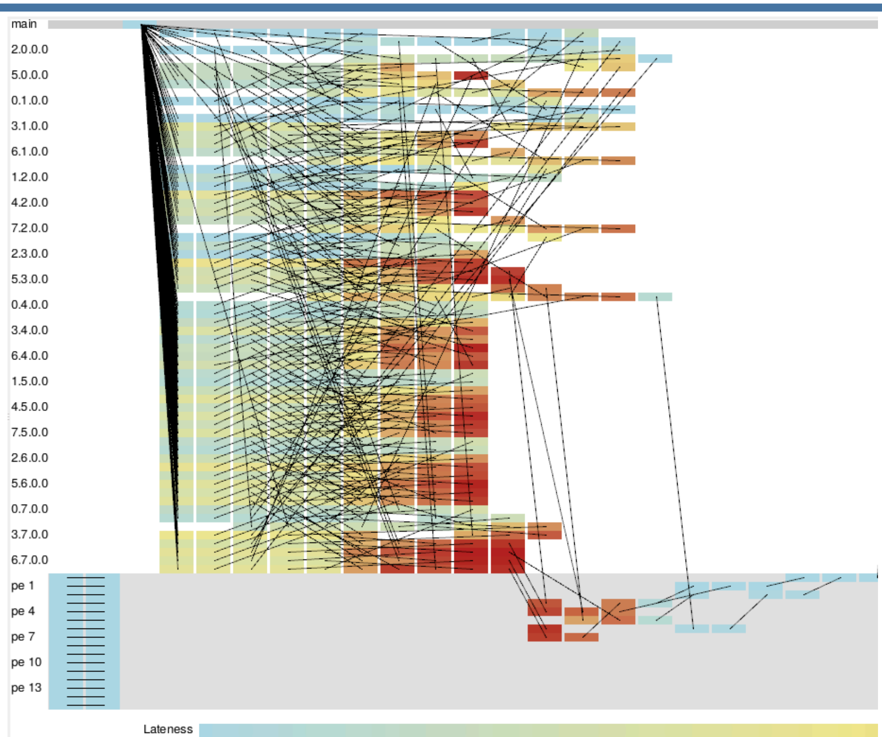
1k processes | 8k processes | 32k processes

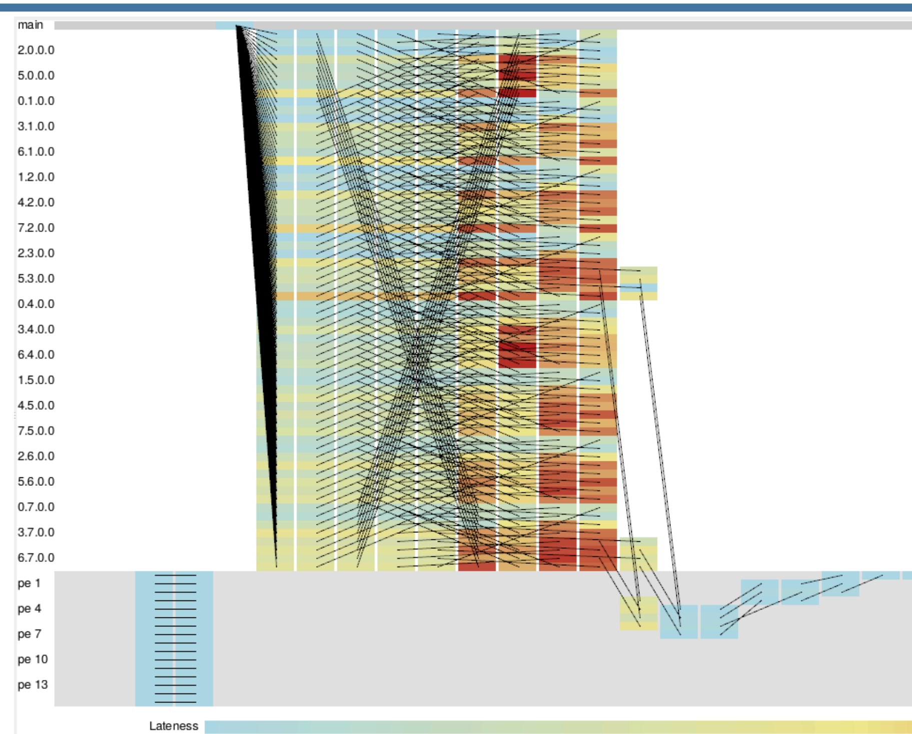
Case Study: Optimizing Communication Patterns





Unraveling Task Based Execution
An Example Based on Charm++





- Visualize tasks and their dependencies
- Left shows mess of tasks *considering* message receive order
- Right shows messages reordered to ignore nondeterminism, colored by lateness.

Conclusions

- We need more insights into performance data
  - Mappings between domains
  - Attribution and correlation with meta-data
  - Visualization, in particular InfoVis
  - Implicit and in-situ analysis of performance data
- Major steps necessary
  - Include more metrics (power, environmental, network, ...)
  - Continuous and facility wide monitoring
  - Extract the necessary context across the SW stack
  - Correlate and visualize context to provide new views on performance
- Examples that embody this approach:
  - Sonar: global NoSQL store and query interface
  - Caliper: flexible context annotation and storage
  - Boxfish: mapping performance data across domains
  - MemAxes: fine grained memory access visualization
  - Ravel: making message traces viable for analysis

The Scalability Team
http://scalability.llnl.gov/

Staff


Abhinav Bhatele


Todd Gamblin


Ignacio Laguna


Kathryn Mohror


Barry Rountree


Martin Schulz

Postdoc


David Beckingsale

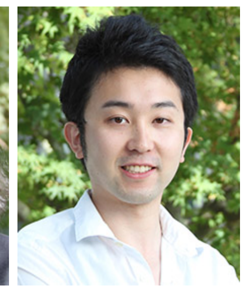

David Boehme

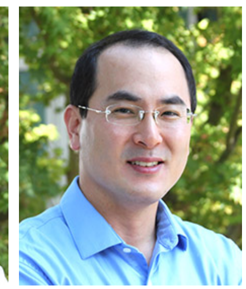

Murali Emani


Tanzima Islam


Aniruddha Marathe


Tapasya Patki


Kento Sato


Jae-Seung Yeom

- ❑ Performance analysis tools and optimization
- ❑ Correctness and debugging (incl. STAT, AutomaDeD, MUST)
- ❑ Power-aware and power-limited computing (incl. Adagio, Conductor)
- ❑ Resilience and Checkpoint/Restart (incl. SCR)

The Scalability Team
http://scalability.llnl.gov/

Staff


Abhinav Bhatele


Todd Gamblin


Ignacio Laguna


Kathryn Mohror


Barry Rountree


Martin Schulz

Postdoc


David Beckingsale

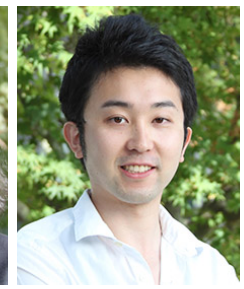

David Boehme

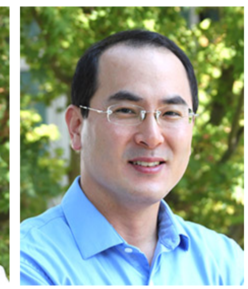

Murali Emani


Tanzima Islam


Aniruddha Marathe


Tapasya Patki


Kento Sato


Jae-Seung Yeom

- Performance analysis tools and optimization
- Correctness and debugging (incl. STAT, AutomaDeD, MUST)
- Power-aware and power-limited computing (incl. Adagio, Conductor)
- Resilience and Checkpoint/Restart (incl. SCR)

Conclusions

http://scalability.llnl.gov/

- We need more insights into performance data
  - Mappings between domains
  - Attribution and correlation with meta-data
  - Visualization, in particular InfoVis
  - Implicit and in-situ analysis of performance data
- Major steps necessary
  - Include more metrics (power, environmental, network, ...)
  - Continuous and facility wide monitoring
  - Extract the necessary context across the SW stack
  - Correlate and visualize context to provide new views on performance
- Examples that embody this approach:
  - Sonar: global NoSQL store and query interface
  - Caliper: flexible context annotation and storage
  - Boxfish: mapping performance data across domains
  - MemAxes: fine grained memory access visualization
  - Ravel: making message traces viable for analysis