Performance and Optimization: A Case for more Modular and Intuitive Tools

Martin Schulz

Lawrence Livermore National Laboratory

INT Exascale Workshop ◆ June 29, 2011

LLNL-PRES-490045

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Complexity is on the Rise

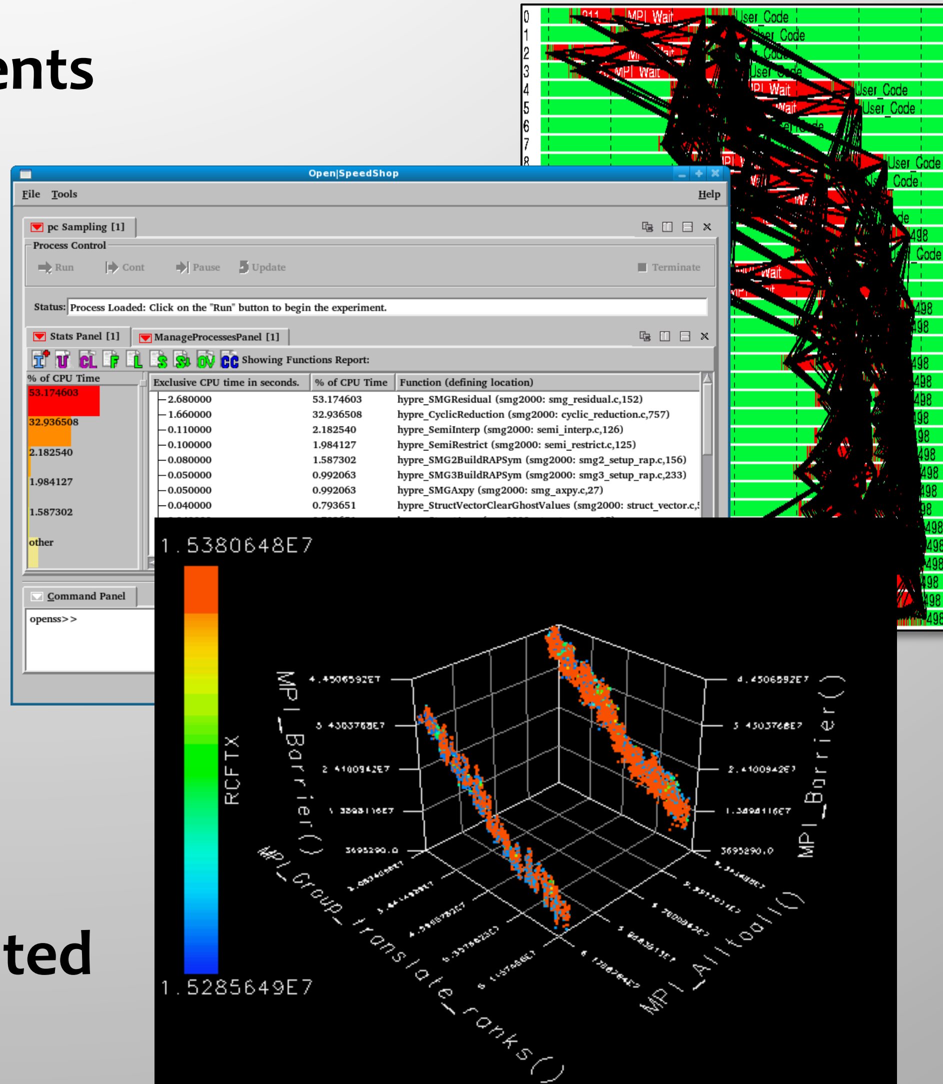
- **Architectures are getting more complex**
  - Huge process and/or thread counts
  - High dimensional network topologies
- **More constraints**
  - Less memory per core
  - Power limitations
  - Resiliency/Fault tolerance
- **Applications are getting more complex**
  - Multiphysics/Multiscale codes
  - Integration of UQ
- **Complex interactions that are not well understood**

Complexity Directly Impacts Programmers

- **Programmers need a new way to program at Exascale**
  - … and need to do so efficiently
- **It will be a challenge to achieve efficiency**
  - Load balance will be key at billions of threads
  - Manual adaptation to architectures may be necessary
  - Memory and network architectures will require layout optimizations
- **Definition of efficiency needs to be revisited**
  - Heterogeneous systems/nodes/chips/units/…
  - Multiple optimization targets (power vs. reliability vs. memory vs. speed)
  - Self-adaptive systems at all layers
  - Baselines are no longer obvious
- **Programmers will need tools more than ever**
  - Identify critical regions and bottlenecks in the code
  - Track down root causes of code behavior

Existing Tools Provide Comprehensive Measurements

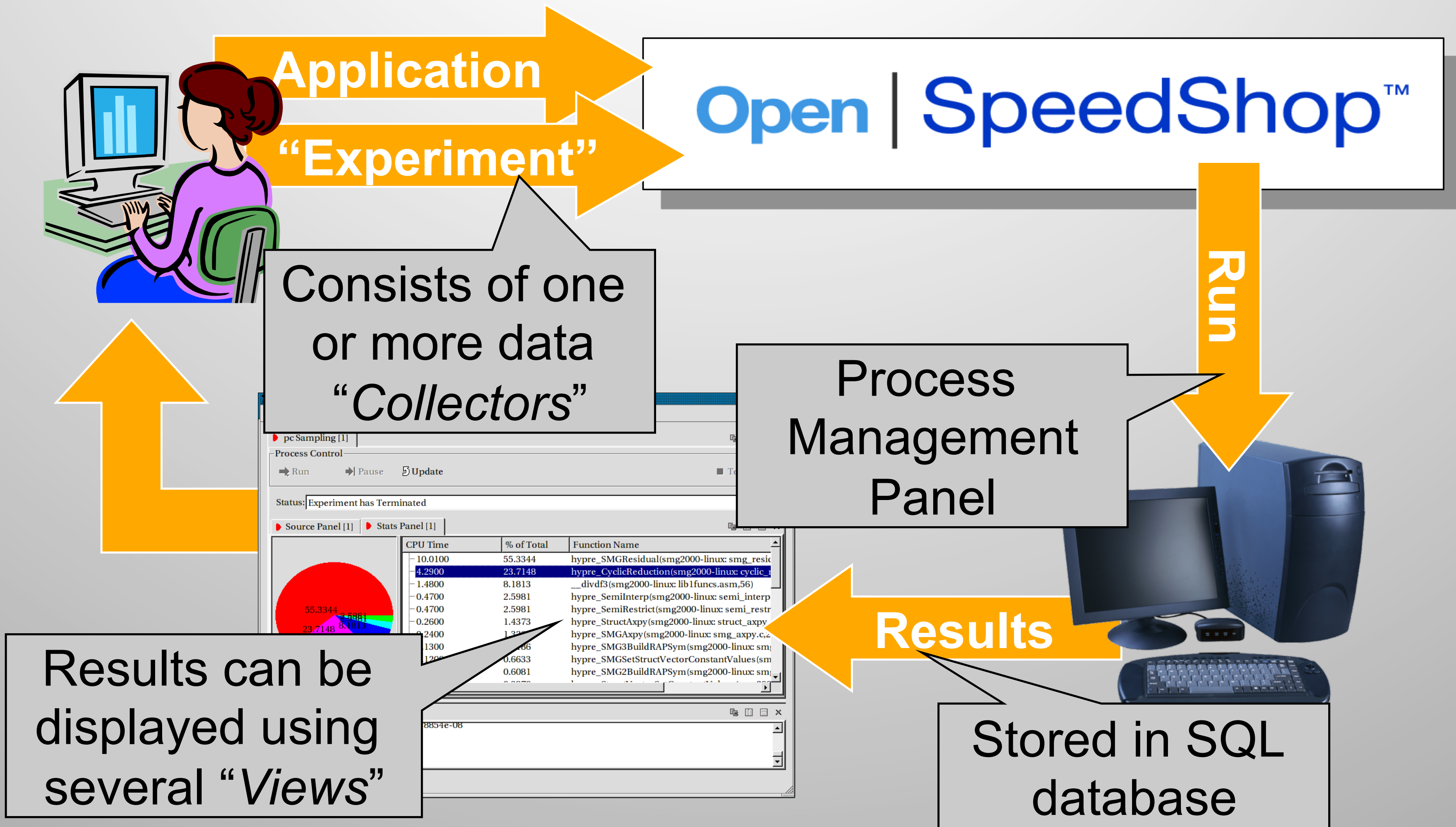
- **Large variety of measurements**
  - Sampling/Tracing
  - Timings/Counters
- **Attribution to source code**
  - Static: Debug information
  - Dynamic: Stack traces
- **Instrumentation options**
  - Transparent instrumentation
  - Binary rewriting
  - Source code annotation
- **Some tools allow sophisticated multi-metric visualizations**



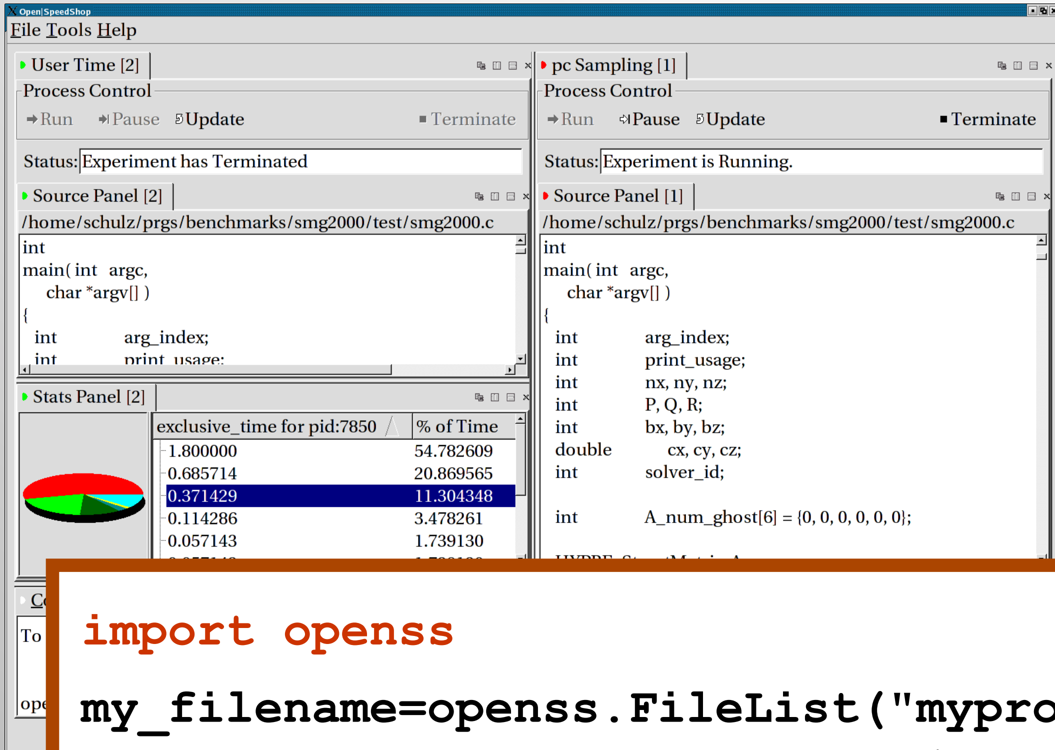
Example: Open|SpeedShop (Krell & ASC/NNSA Trilabs)

- **Open Source Performance Analysis Tool Framework**
  - Most common performance analysis steps ***all in one tool***
  - Supports ***sampling*** and ***tracing*** of various metrics
- **Flexible and Easy to use**
  - User access through ***GUI***, ***Command Line***, and ***Python Scripting***
- **Several Instrumentation Options**
  - All work on ***unmodified application binaries***
  - ***Offline*** and ***online data collection*** / ***attach*** to running codes
- **Supports a wide range of systems**
  - Extensively used and tested on a variety of ***Linux clusters***
  - Support for ***Cray XT/XE*** and ***Blue Gene/P***
- **http://www.openspeedshop.org/**

Open|SpeedShop Workflow



O|SS Analysis Interfaces for Ease of Use




```
Experiment Commands
    expAttach
    expCreate
    expDetach
    expGo
    expView

List Commands
```

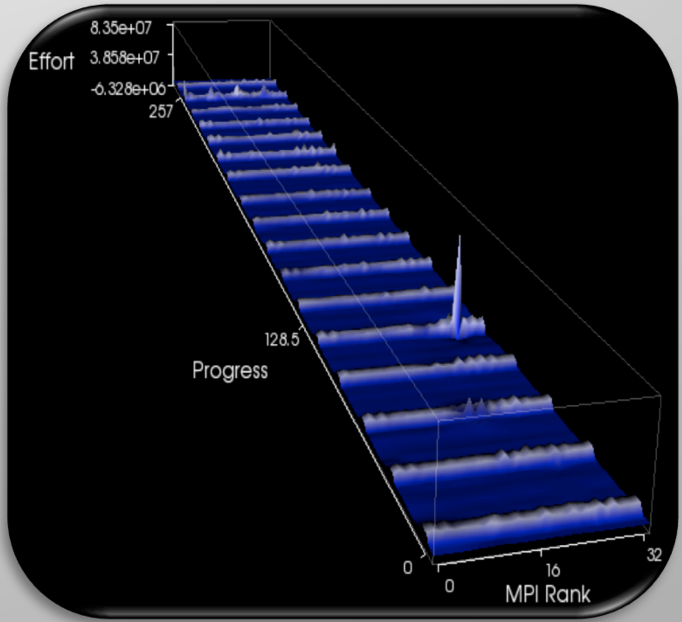
```
import openss

my_filename=openss.FileList("myprog.a.out")
my_exptype=openss.ExpTypeList("pcsamp")
my_id=openss.expCreate(my_filename,my_exptype)

openss.expGo()

My_metric_list = openss.MetricList("exclusive")
my_viewtype = openss.ViewTypeList("pcsamp”)
result = openss.expView(my_id,my_viewtype,my_metric_list)
```

Selected Other Tool Sets

- **HPCToolkit (Rice Univeristy)**
  - Specialized on sampling techniques
- **TAU (University of Oregon)**
  - Large toolkit of performance analysis techniques
- **Kojak/Scalasca (JSC)**
  - Profiling and Tracing combined with automatic trace analysis
- **Vampir NG (TU Dresden)**
  - Trace visualization
- **Paraver (UPC/BSC)**
  - Trace visualization and phase detection
- **Vendor tools**

Emphasis on Scaling

- **Tools need to scale with machine and application**
  - Deal with a flood of data from all tasks/threads
  - Challenge to Acquire, Store, Analyze, and Visualize
- **Current tools do scale to 10K or even 100K nodes**
  - Often achieved with "brute force", though
- **New techniques**
  - Tree-based Reduction Networks
  - Data compression
  - Consequence: tools are no longer free!
- **Example: Libra**
  - Scalable Load Balance Analysis
  - Wavelet compression

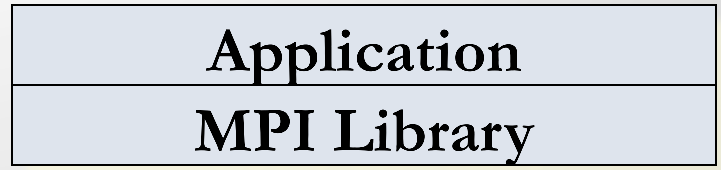


A Case for Modularity

- **The generality of current tools may be their Achilles' heel**
  - Great to get initial overview and to find basic bottlenecks
  - Bad to explore specific performance problems
  - New architectures will pose new and complex problems
- **Goal: application and scenario specific analysis tools**
  - Tools that understand and exploit application semantics
  - Tools that explore one particular machine aspect in detail
  - BUT: Can't develop new tools for every case
- **We need modular tool component frameworks**
  - Reusable and compatible components
  - Easy to maintain and distribute
  - Quick prototyping of new tools for new scenarios
  - Examples: P^NMPI (LLNL), CBTF (Krell/ASCR), PTP (IBM)

Quick Tool Prototyping with P^NMPI

- **PMPI interception of MPI calls**
  - Used by many MPI tools
  - Limited to a single tool



Quick Tool Prototyping with P^NMPI

- **PMPI interception of MPI calls**
  - Used by many MPI tools
  - Limited to a single tool

Application
PMPI Tool 1
MPI Library

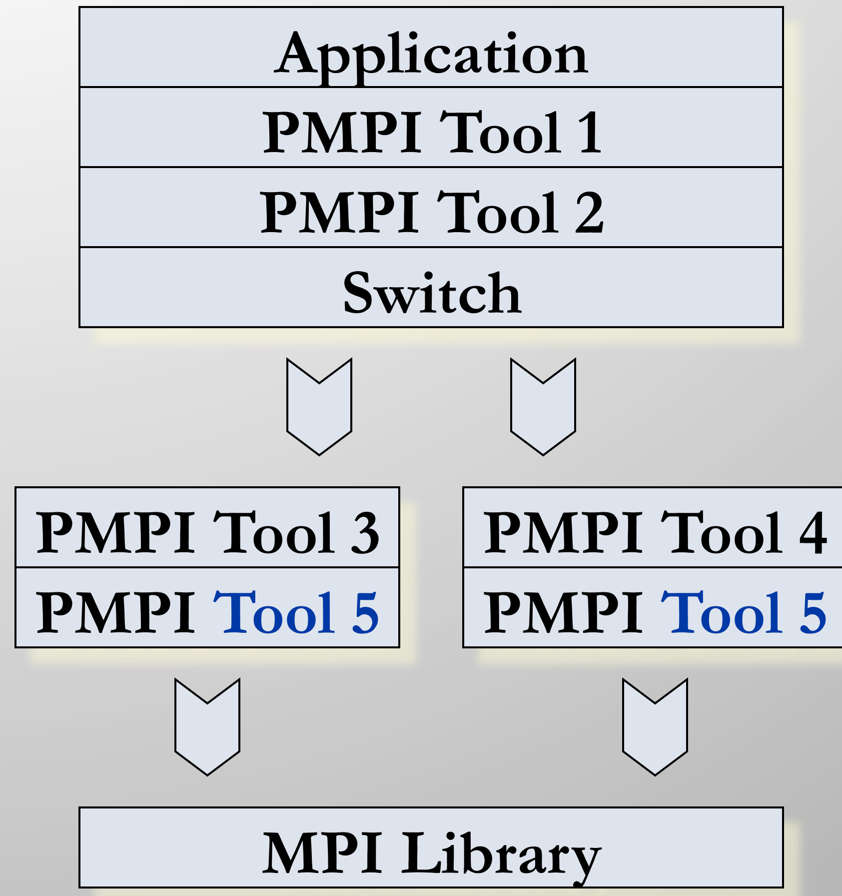
Quick Tool Prototyping with P^NMPI

- **PMPI interception of MPI calls**
  - Used by many MPI tools
  - Limited to a single tool
- **P^NMPI virtualized PMPI**
  - Multiple tools concurrently
  - Dynamic loading of tools
  - Configuration through text file
  - Tools are independent
  - Tools can collaborate

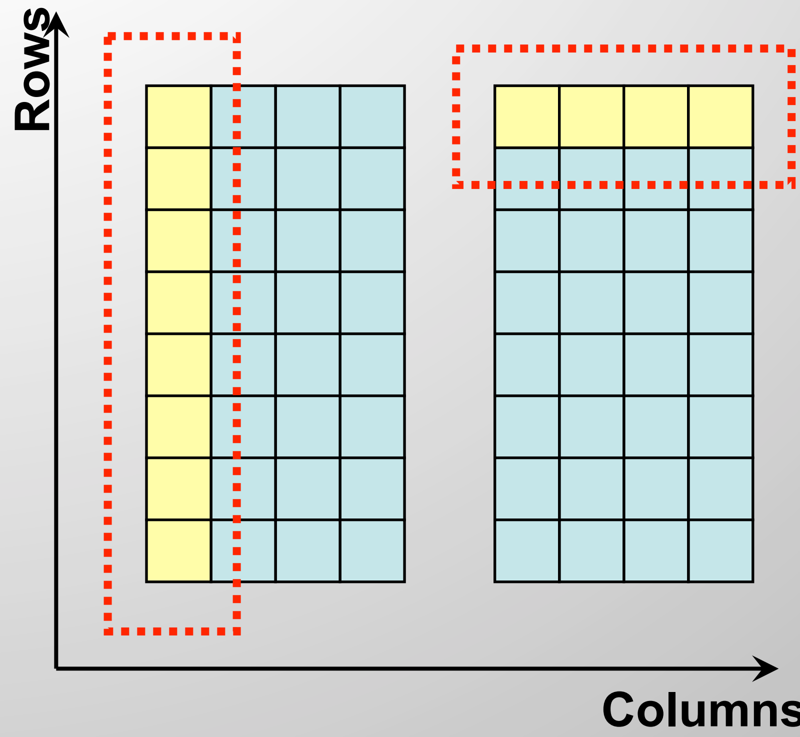
Application
PMPI Tool 1
PMPI Tool 2
MPI Library

Quick Tool Prototyping with P^NMPI

- **PMPI interception of MPI calls**
  - Used by many MPI tools
  - Limited to a single tool
- **P^NMPI virtualized PMPI**
  - Multiple tools concurrently
  - Dynamic loading of tools
  - Configuration through text file
  - Tools are independent
  - Tools can collaborate
- **Transparently adding context**
  - Select tool based on MPI context
  - Transparently isolate tool instances



Example: Optimizing an FPMD Code



- **Data structure: dense matrix**
  - Row and column communicators
  - Additional global operations
  - Standard profiles aggregate data
- **Need to profile separately**
  - Potentially different operations
  - May lead to separate optimization
  - BUT: don't want to rewrite profiler
- **Switch module to split communication (111 lines of code)**
  - Create three independent tool stacks
  - Apply unmodified profiler (mpiP) in each stack
  - Transparent to profiler, application & MPI library

Example: Defining Switch Modules in P^NMPI

Configuration file:

Default Stack

```
module commsize-switch
argument sizes 8 4
argument stacks column row
module mpiP
```

Target Stack 1

```
stack row
module mpiP1
```

Target Stack 2

```
stack column
module mpiP2
```

Switch Module

Arguments controlling switch module

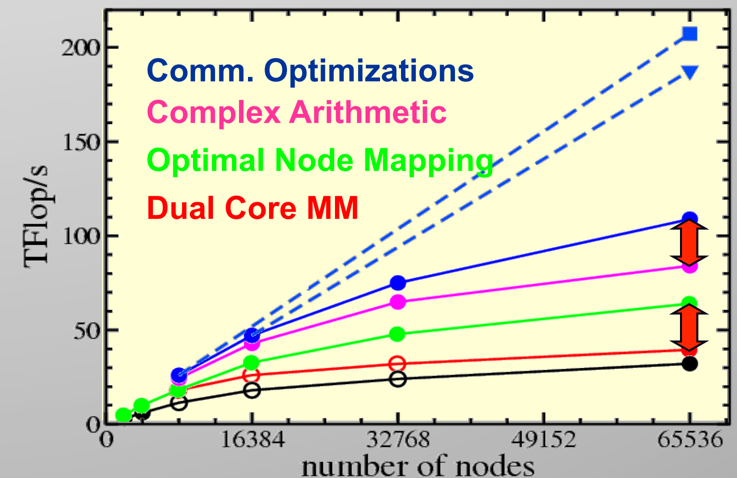
Multiple profiling instances

Example: Communicator Profiling for an FPMD Code

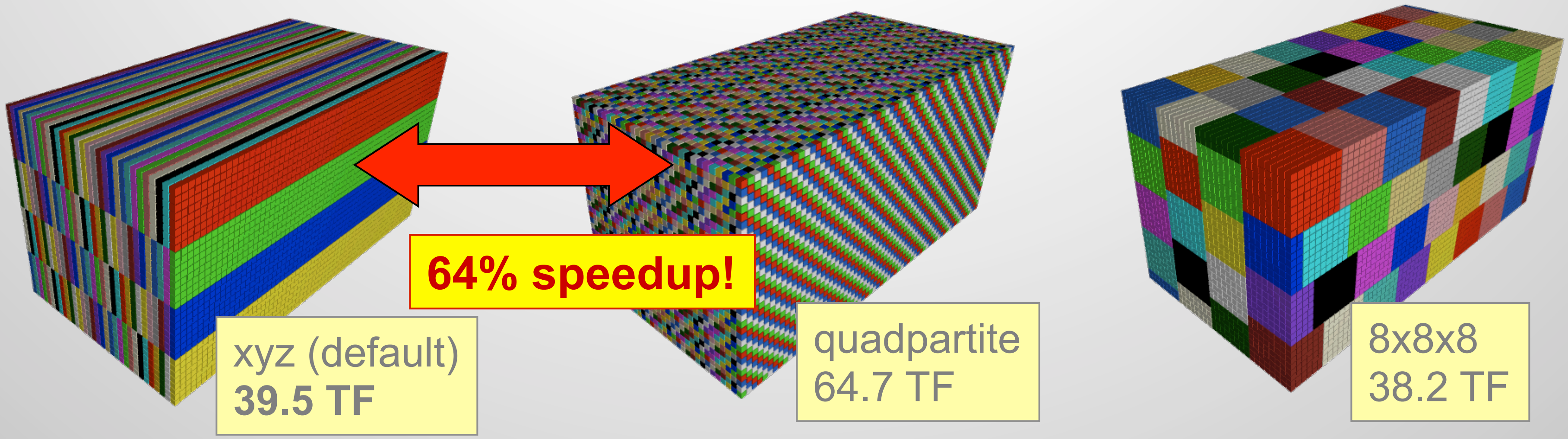
Operation	Sum	Global	Row	Column
Send	317245	31014	202972	83259
Allreduce	319028	269876	49152	0
Alltoallv	471488	471488	0	0
Recv	379265	93034	202972	83259
Bcast	401042	11168	331698	58176

- **Information helpful for …**
  - Understanding behavior
  - Locating optimization targets
  - Optimizing of collectives
  - Identifying better node mapping

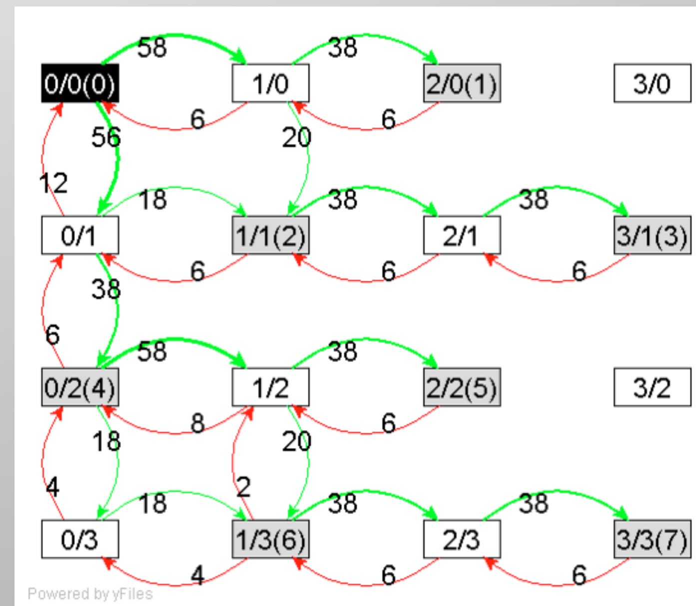
AMD Opteron/Infiniband Cluster



Example: Improved Node Mappings in an FPMD Code

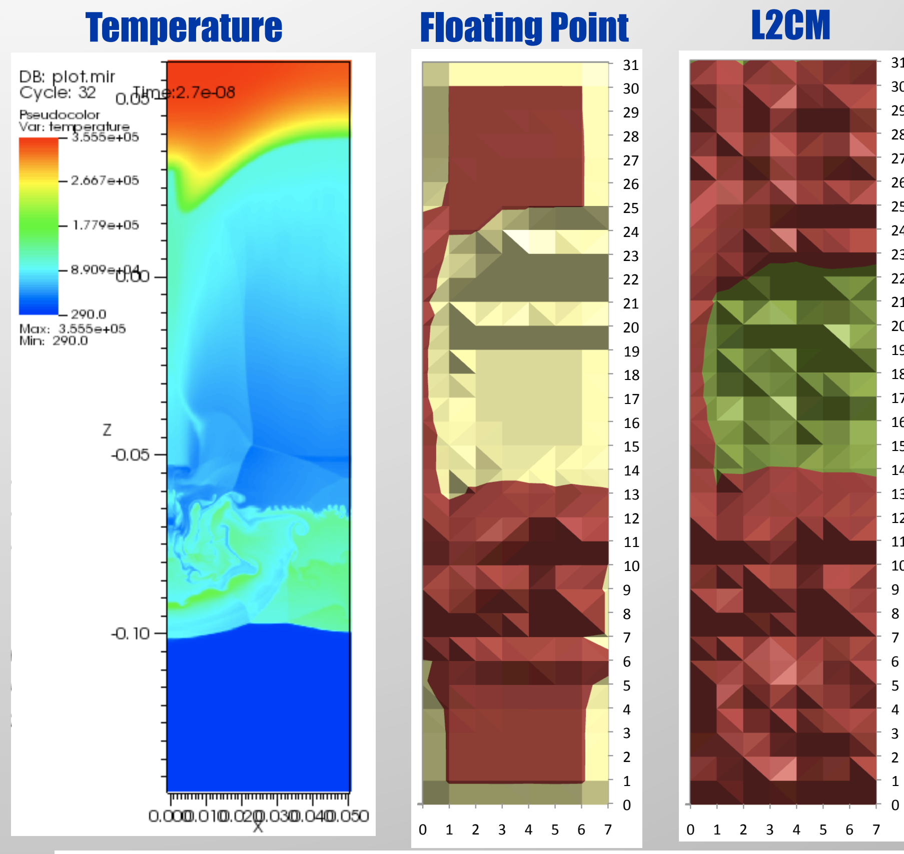


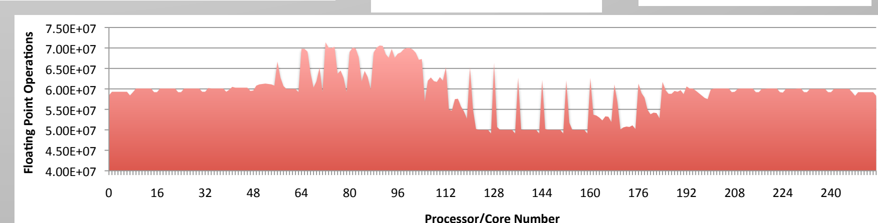
- **Communicator specific profiling was critical for optimization**
  - Identified different row/column behavior
  - Isolated critical MPI operations
  - Enabled specialized traffic modeling
- **Key factor: collective bandwidth**
  - Important to keep more links active



A Case for More Intuitive Analysis Tools

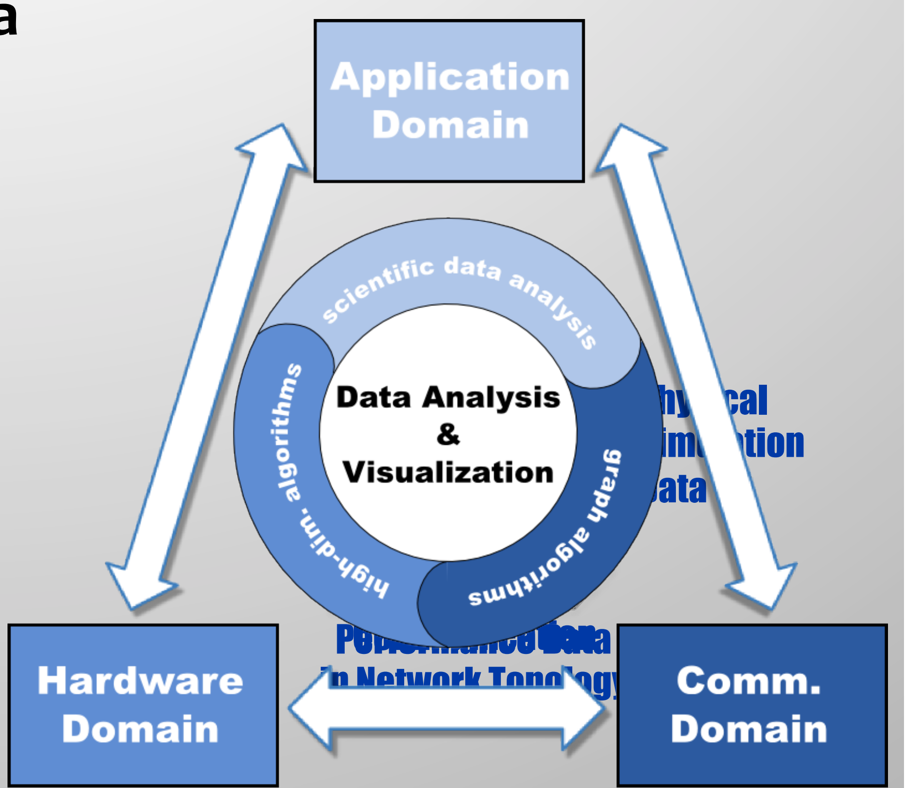
- **Example: 256 core run of a CFD application**
  - Floating point operations
- **Application developers think in the app domain**
- **Simple step:**
  - Map floating point ops onto the application domain
  - Similar L2 cache misses
- **Clear correlations**
  - Explains performance
  - Helps establish a baseline





Considering Multiple Domains of Performance

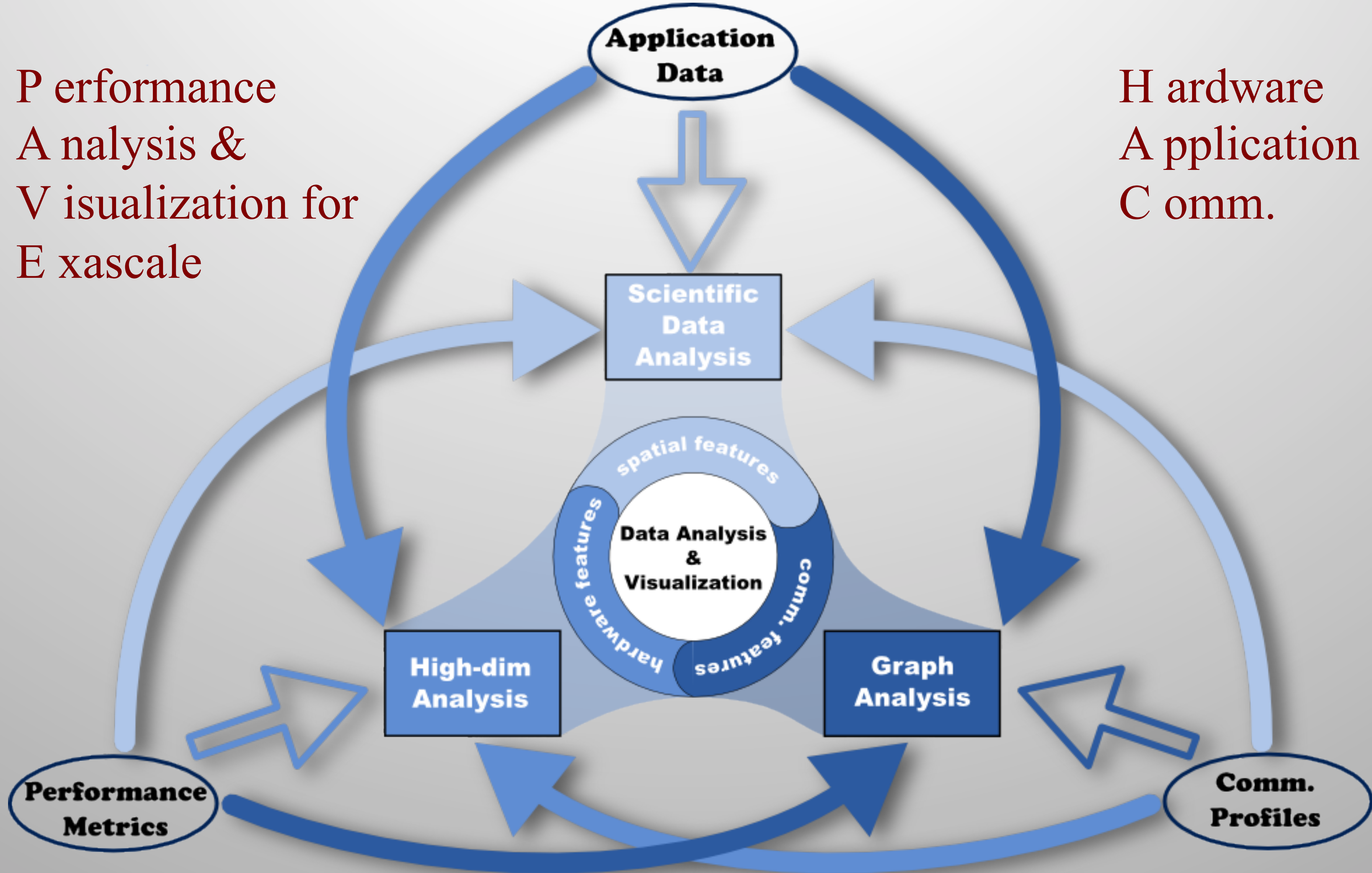
- **Tools must consider several domains / perspectives on data**
  - Application domain
  - Hardware domain
  - Communication domain
- **Visualize in the domains**
- **Inter domain mappings**
  - Enable new perspectives
  - Analysis across domains
  - Use data analysis techniques
- **Forming a bridge between Performance Analysis and Data Analysis/Visualization**



The PAVE/HAC Model

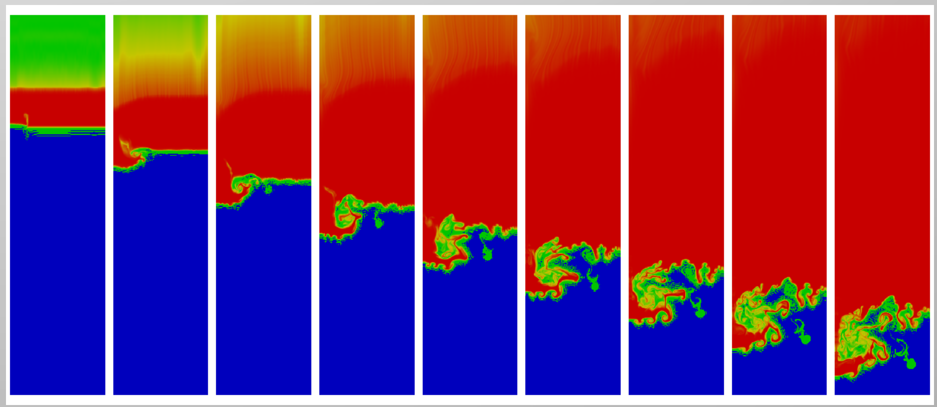
P erformance
A nalysis &
V isualization for
E xascale

H ardware
A pplication
C omm.

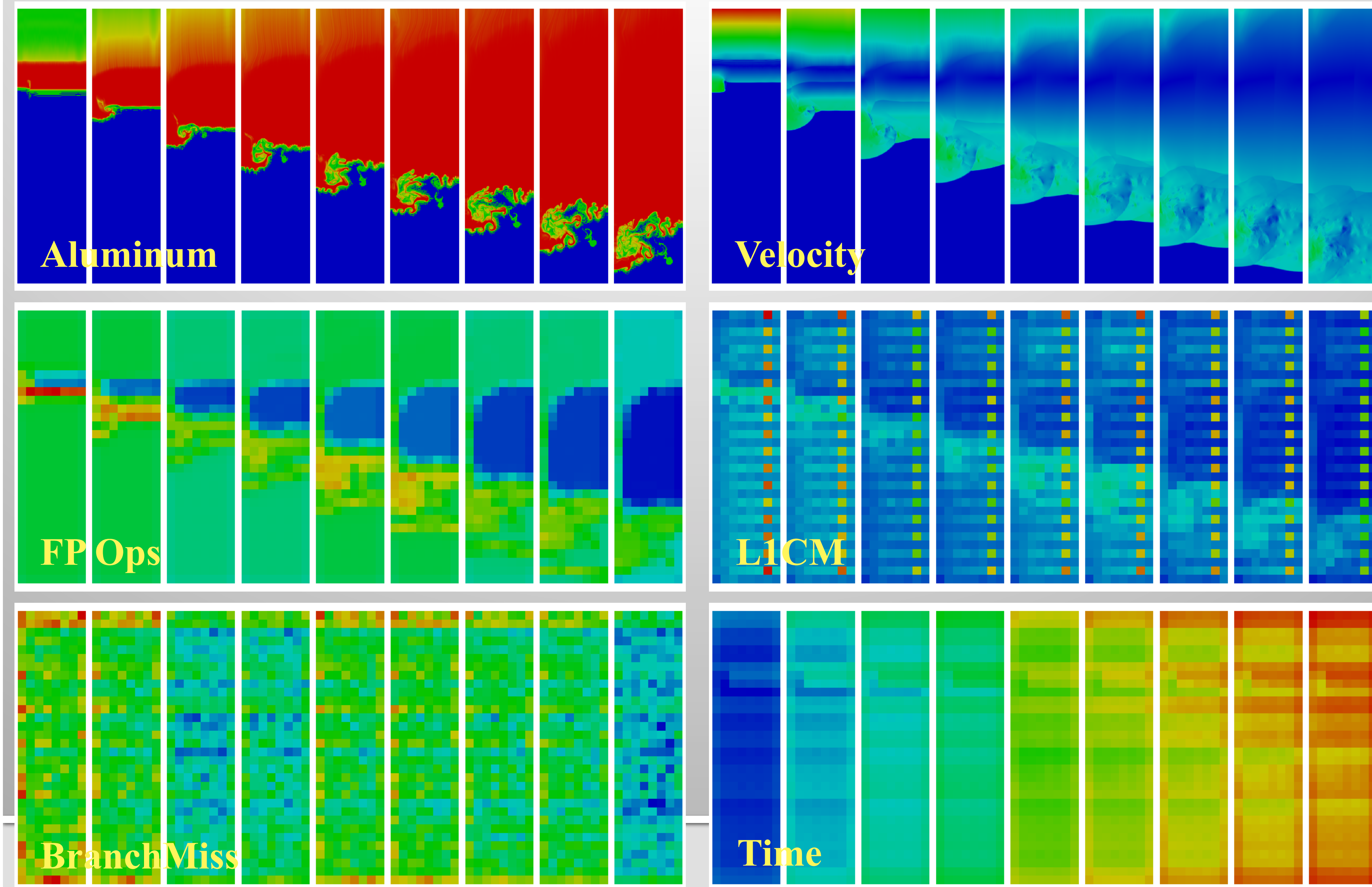


Mapping Measurements into the Application Domain

- **Performance measurements acquired in the hardware domain**
  - Map into the application's physical space
  - Visualize data in the application domain that is familiar to the developer
  - Ability to use existing and proven visualization techniques
- **Requirements**
  - Applications need to expose process ID -> grid mapping
  - In some cases we extract this automatically
- **Case study**
  - CFD application
  - Shock wave caused by Aluminum jet
  - 2D version, 32x8 CPUs
  - 9 time steps

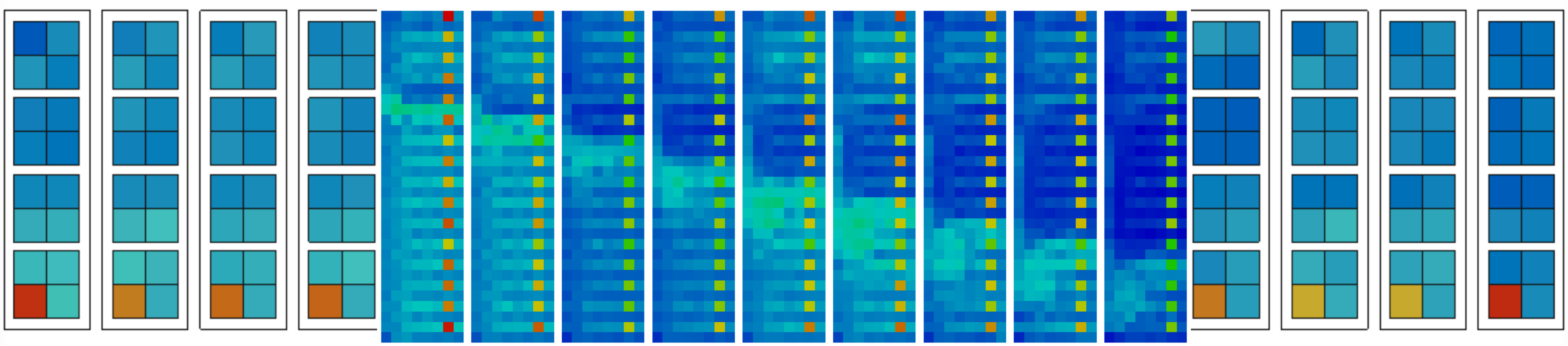


CFD Code, 9 Time Steps, 32x8 Processor Grid



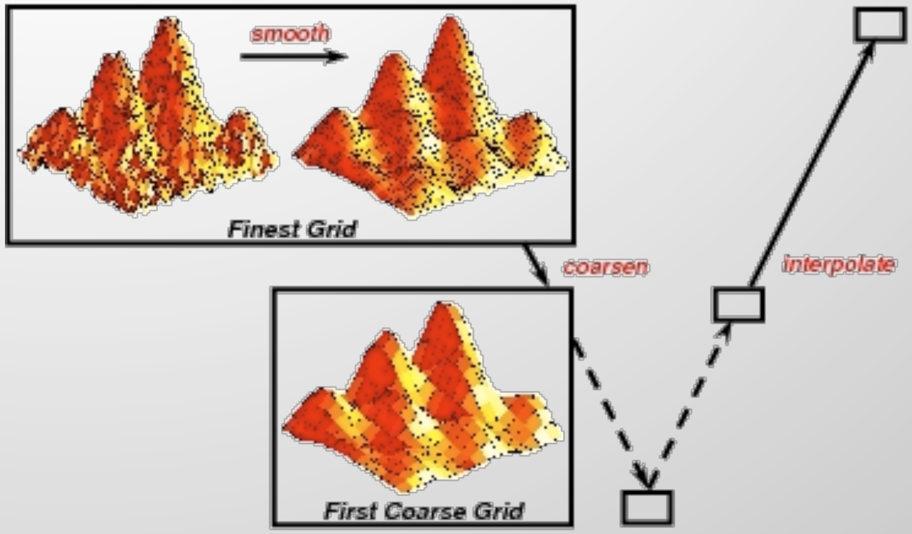
Observations

- **Clear correlations for FP Ops and L1 misses**
- **Branch misses and time show boundary effects**
- **Time steps get more expensive over time**
- **Secondary, independent effect for L1 misses**
  - Single core per socket creates more cache misses
  - Most likely caused by MPI activity
  - Shows that we need different perspectives to disambiguate causes



Case Study: Algebraic Multigrid Solver (AMG)

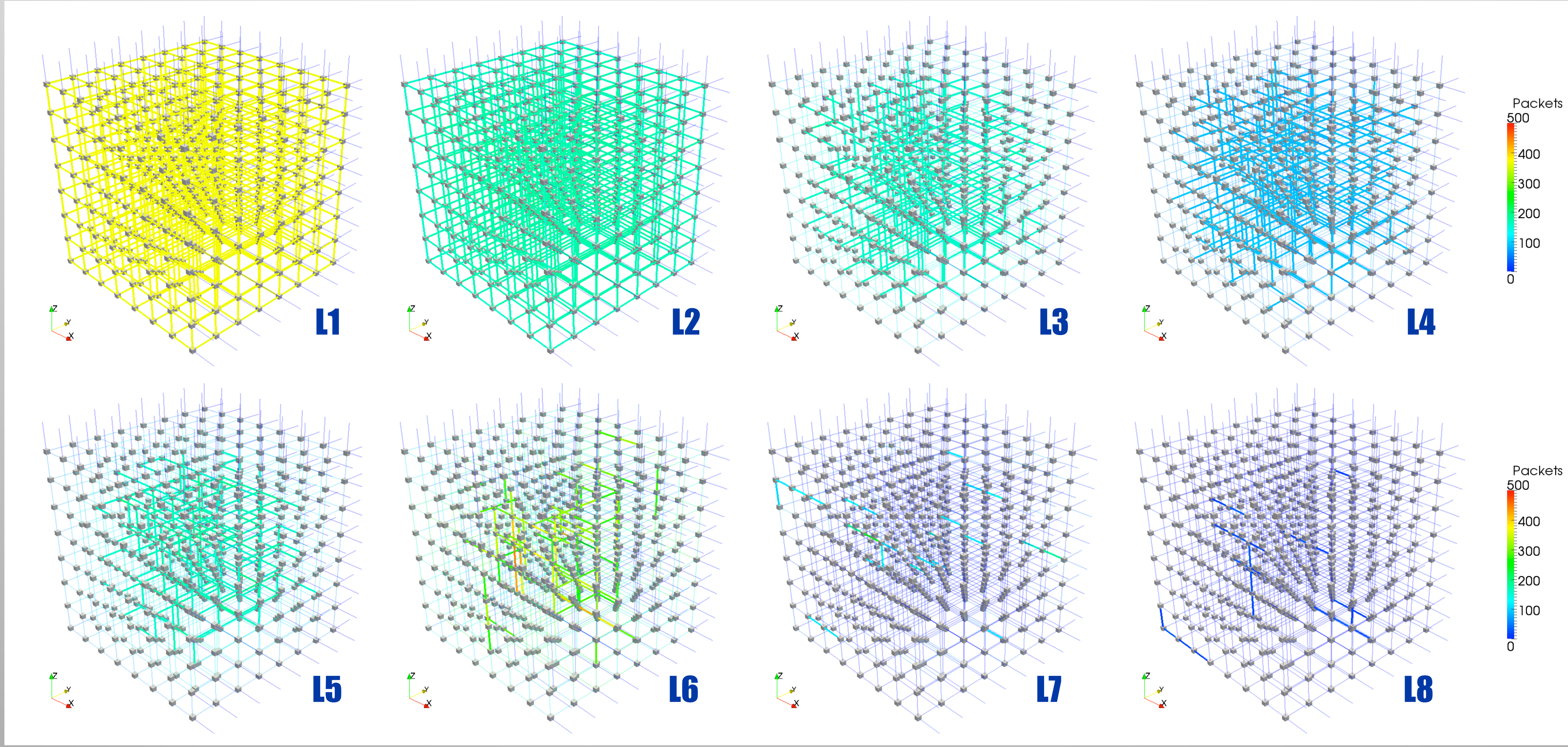
- **Essential component for many applications**
- **Series of V Cycles**
  - Coarsening
  - Direct solve
  - Interpolation



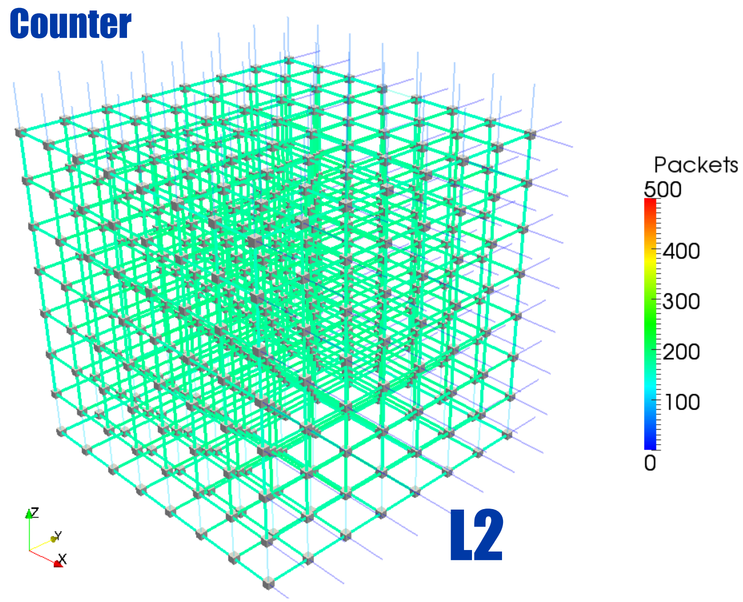
- **Communication requirements change between layers**
  - Fine layers have nearest neighbor communication
  - Coarse layers typically have more communication partners
  - Potential for link contention
- **Experimental setup**
  - AMG2006 on BG/P, 512 nodes/tasks
  - Measurements of X+/X-, Y+/Y-, Z+/Z- link activity

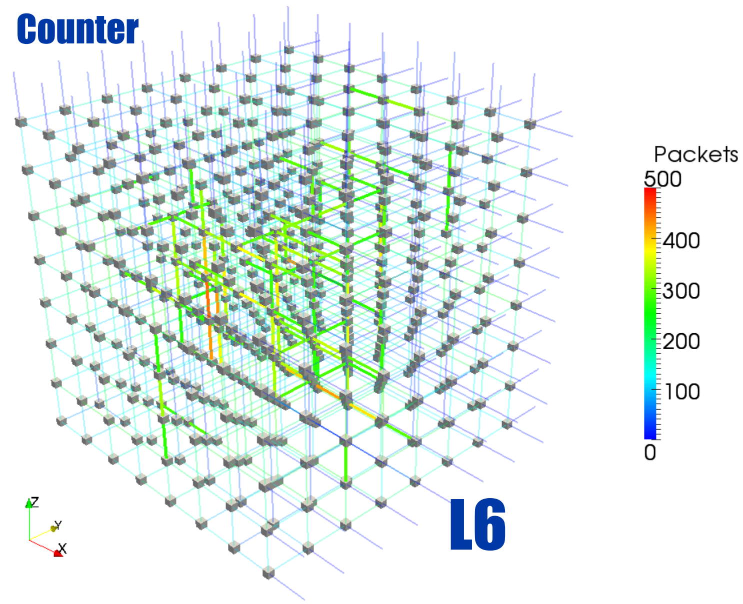
AMG on BG/L, 8x8x8 HW Torus, 8x8x8 virtual topology

- **Communication counters for all eight levels of AMG**
  - Mapped/Aggregated to the edges in a torus display



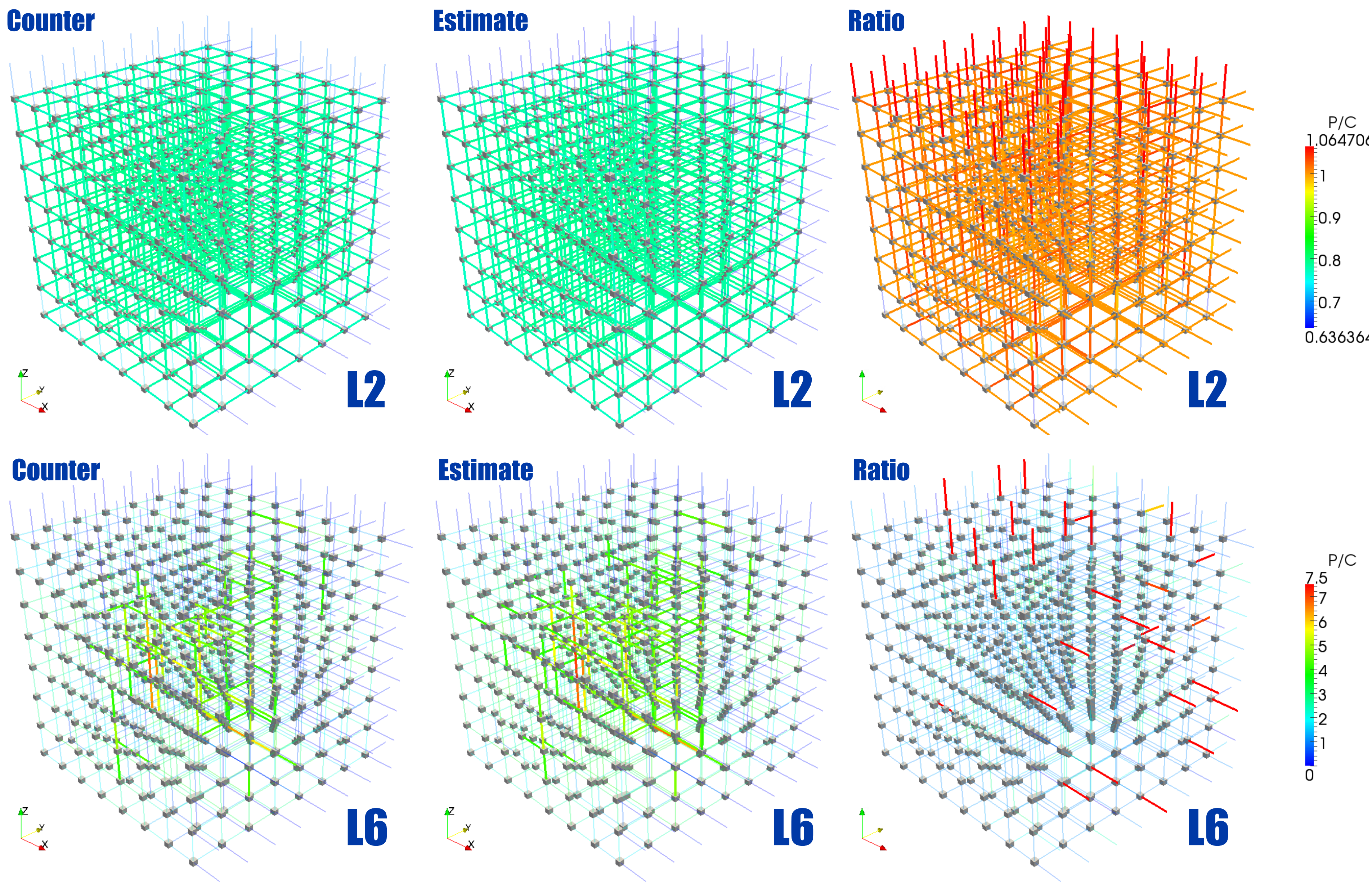
AMG on BG/L, 8x8x8 HW Torus, 8x8x8 virtual topology





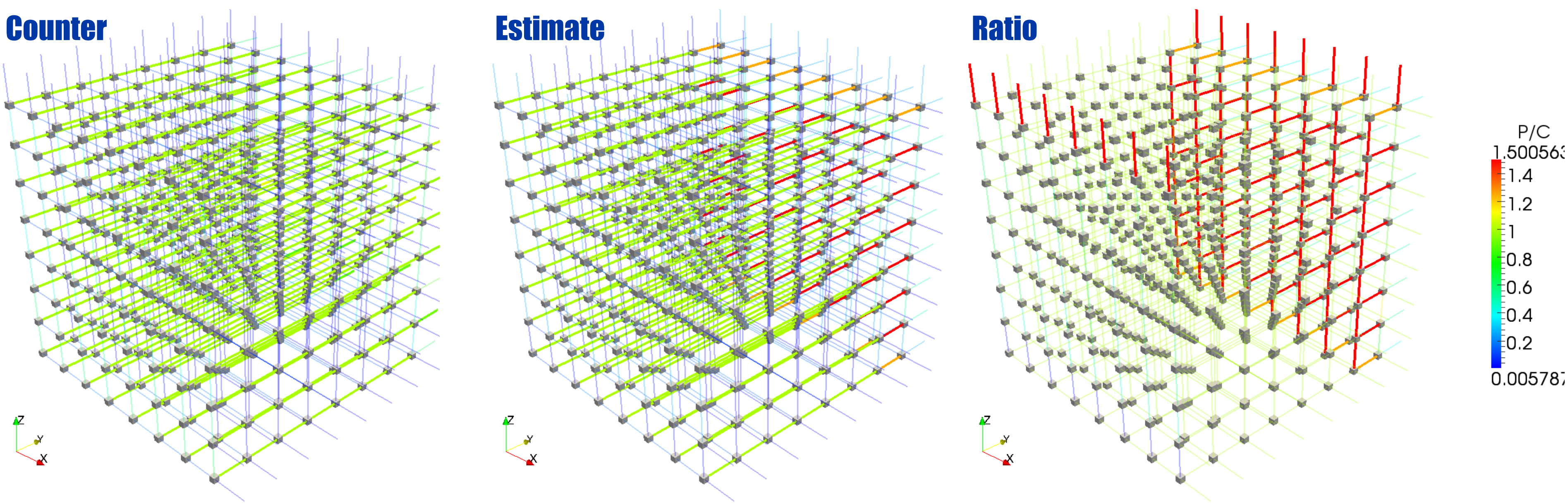
- **New visualization that shows the hardware topology (level 2 vs. 6)**
  - Finer layers are more global
  - Coarse layers have fewer partners
- **How can we interpret the data?**
  - Need connection to MPI communication
  - Need a baseline to compare to
- **Map Communication to HW domain**
  - Gather full MPI communication matrix
  - Emulate each message based on observed patters and aggregate
  - Contrast estimate with measurements
  - Ability to detect hotspots/contention

AMG on BG/L, 8x8x8 HW Torus, 8x8x8 virtual topology



Observation

- **Identify communication sparsity**
  - Communication displays provide good insight
  - Leverage data analysis and visualization techniques
- **Experiments with non optimal decompositions**
  - BG/L, 8x8x8 HW Torus, 2x4x64 virtual topology
  - Results show more potential bottlenecks, but ratio is small



Conclusions

- **Existing tools provide large range of measurements**
  - Variety of options and source attribution techniques
  - Proven in many projects and at large scale
  - BUT: Often monolithic, overly general tools
- **We need more modularity in tools**
  - Must enable quick prototyping of tools
  - Essential to explore application and machine specific problems
- **We need for more intuitive tools**
  - Project data into domains the user can understand
  - Enable data and feature correlation across domains
- **Need for more collaboration towards Exascale**
  - Tools will no longer be free or fully transparent
  - Users need to create and deploy tools in their environment
  - Dialogue between application and tool community needs to grow