Machine Learning-based Autotuning with TAU and Active Harmony

Nicholas Chaimov
University of Oregon

Paradyn Week 2013
April 29, 2013

Outline

- Brief introduction to TAU
- Motivation
- Relevant TAU Tools:
  - TAUdb
  - PerfExplorer
- Using TAU in an autotuning workflow
- Machine Learning with PerfExplorer
- Future Work

Motivation

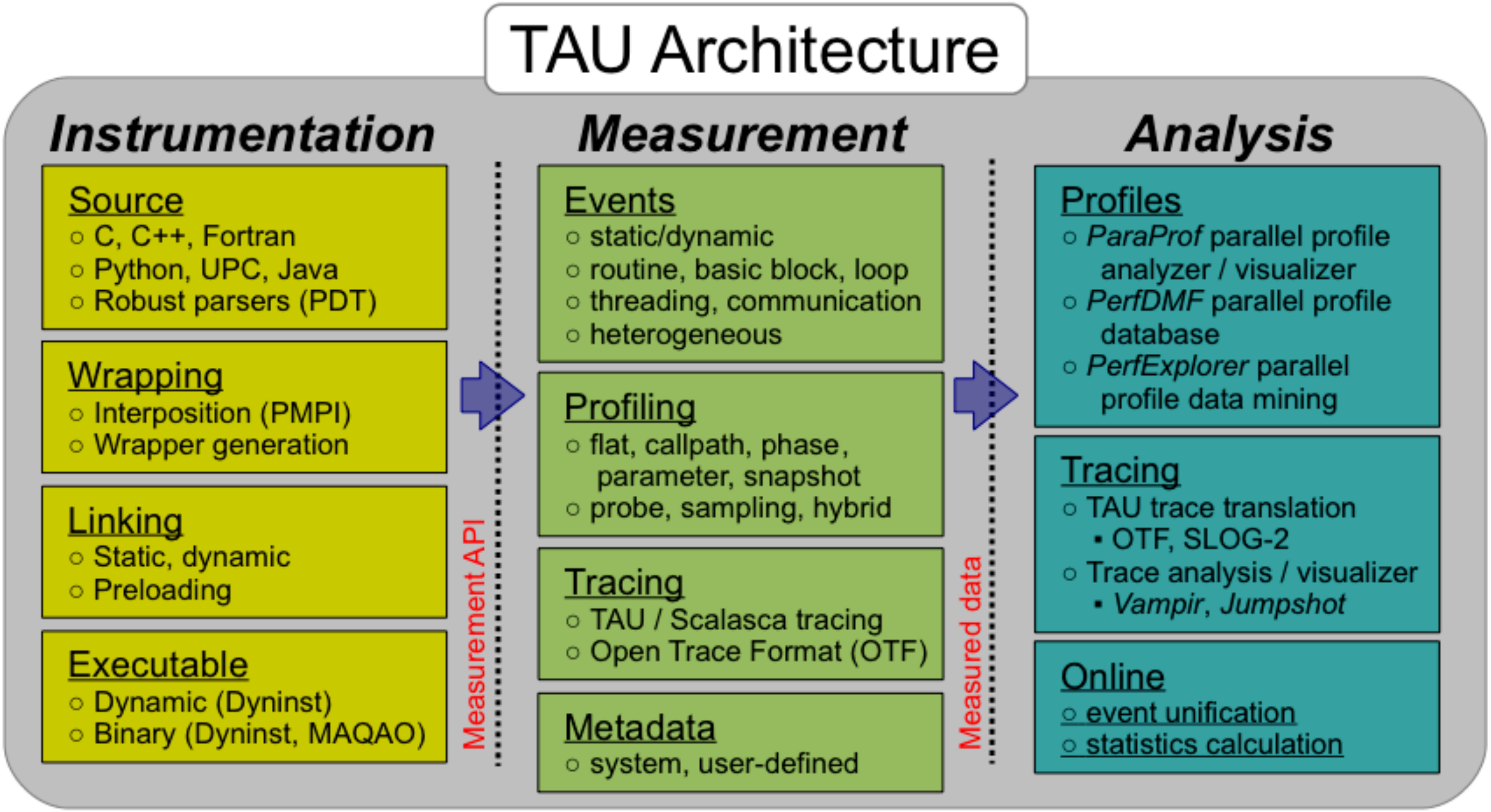
- Goals:
  - Generate code that adapts to changes in the execution environment and input datasets.
  - Avoid spending large amounts of time performing search to autotune code.
- Method: learn from past performance data in order to
  - Automatically generate code to select a variant at runtime based upon execution environment and input dataset properties.
  - Learn classifiers to select search parameters (such as initial configuration) to speed the search process.

TAU Performance System® (*http://tau.uoregon.edu*)

- Tuning and Analysis Utilities (20+ year project)
- Performance problem solving framework for HPC
  - Integrated, scalable, flexible, portable
  - Target all parallel programming / execution paradigms
- Integrated performance toolkit
  - Multi-level performance instrumentation
  - Flexible and configurable performance measurement
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
- Broad use in complex software, systems, applications

TAU Organization

- Parallel performance framework and toolkit
  - Supports all HPC platforms, compilers, runtime system
  - Provides portable instrumentation, measurement, analysis



TAU Components

- Instrumentation
  - Fortran, C, C++, UPC, Chapel, Python, Java
  - Source, compiler, library wrapping, binary rewriting
  - Automatic instrumentation
- Measurement
  - MPI, OpenSHMEM, ARMCI, PGAS
  - Pthreads, OpenMP, other thread models
  - GPU, CUDA, OpenCL, OpenACC
  - Performance data (timing, counters) and metadata
  - Parallel profiling and tracing
- Analysis
  - Performance database technology (TAUdb, formerly PerfDMF)
  - Parallel profile analysis (ParaProf)
  - Performance data mining / machine learning (PerfExplorer)

TAU Instrumentation Mechanisms

- **Source code**
  - Manual (TAU API, TAU component API)
  - Automatic (robust)
    - C, C++, F77/90/95 (Program Database Toolkit (***PDT***))
    - OpenMP (directive rewriting (*Opari*), *POMP2* spec)
- **Object code**
  - Compiler-based instrumentation (-optCompInst)
  - Pre-instrumented libraries (e.g., MPI using *PMPI*)
  - Statically-linked and dynamically-linked (tau_wrap)
- **Executable code**
  - Binary re-writing and dynamic instrumentation (*DyninstAPI, U. Wisconsin, U. Maryland*)
  - Virtual machine instrumentation (e.g., Java using *JVMPI*)
  - Interpreter based instrumentation (Python)
  - Kernel based instrumentation (KTAU)

Instrumentation: Re-writing Binaries

- Support for both static and dynamic executables
- Specify the list of routines to instrument/exclude from instrumentation
- Specify the TAU measurement library to be injected
- Simplify the usage of TAU:
  - To instrument:

    ```
    % tau_run a.out -o a.inst
    ```
  - To perform measurements, execute the application:

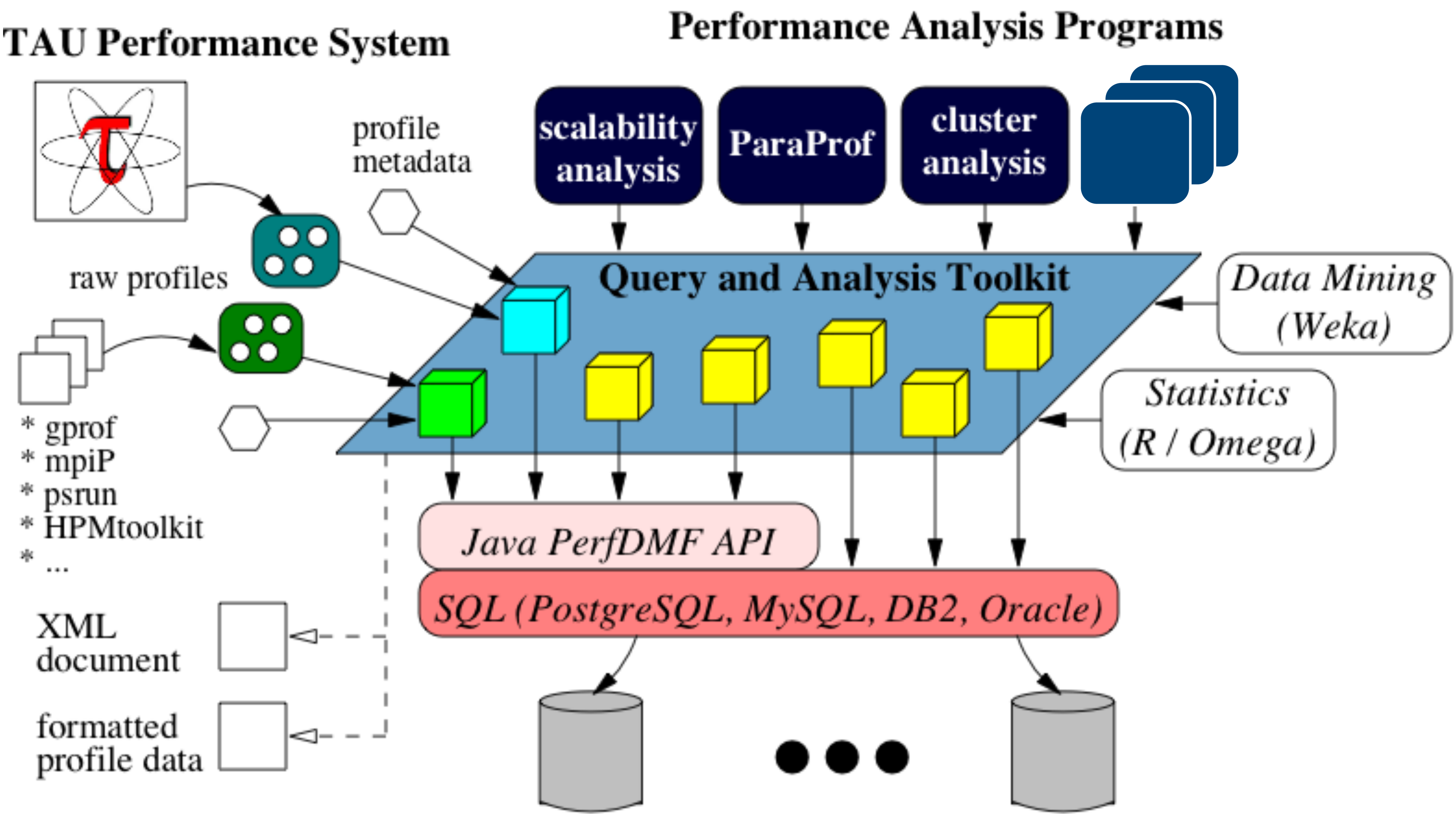
    ```
    % mpirun -np 8 ./a.inst
    ```
  - To analyze the data:

    ```
    % paraprof
    ```

DyninstAPI 8.1 support in TAU

- TAU v2.22.2 supports DyninstAPI v8.1
- Improved support for static rewriting
- Integration for static binaries in progress
- Support for loop level instrumentation
- Selective instrumentation at the routine and loop level

TAUdb: Framework for Managing Performance Data



TAU Performance Database – TAUdb

- Started in 2004 (Huck et al., ICPP 2005)
  - Performance Data Management Framework (PerfDMF)
- Database schema and Java API
  - Profile parsing
  - Database queries
  - Conversion utilities (parallel profiles from other tools)
- Provides DB support for TAU profile analysis tools
  - ParaProf, PerfExplorer, EclipsePTP
- Used as regression testing database for TAU
- Used as performance regression database
- Ported to several DBMS
  - PostgreSQL, MySQL, H2, Derby, Oracle, DB2

TAUdb Database Schema

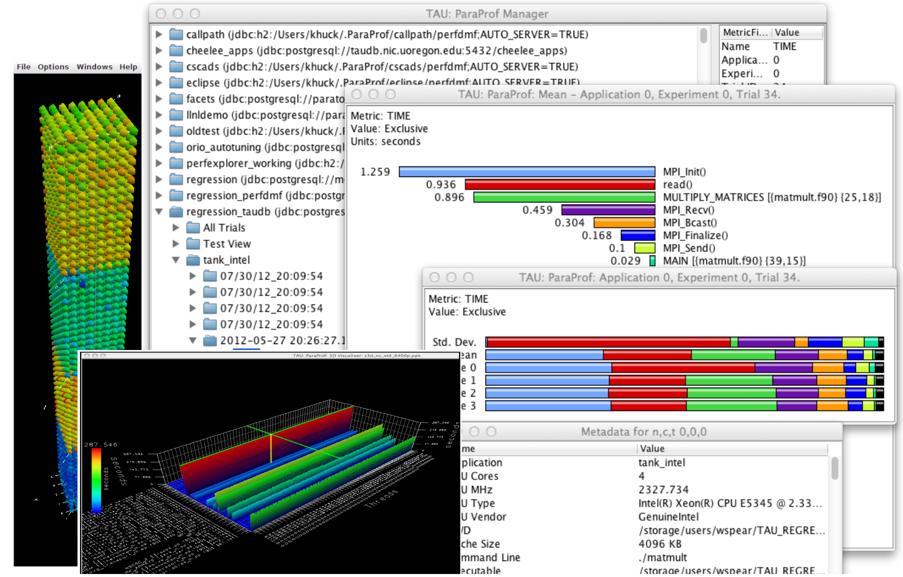
- Parallel performance profiles
- Timer and counter measurements with 5 dimensions
  - Physical location: process / thread
  - Static code location: function / loop / block / line
  - Dynamic location: current callpath and context (parameters)
  - Time context: iteration / snapshot / phase
  - Metric: time, HW counters, derived values
- Measurement metadata
  - Properties of the experiment
  - Anything from *name:value* pairs to nested, structured data
  - Single value for whole experiment or full context (tuple of thread, timer, iteration, timestamp)

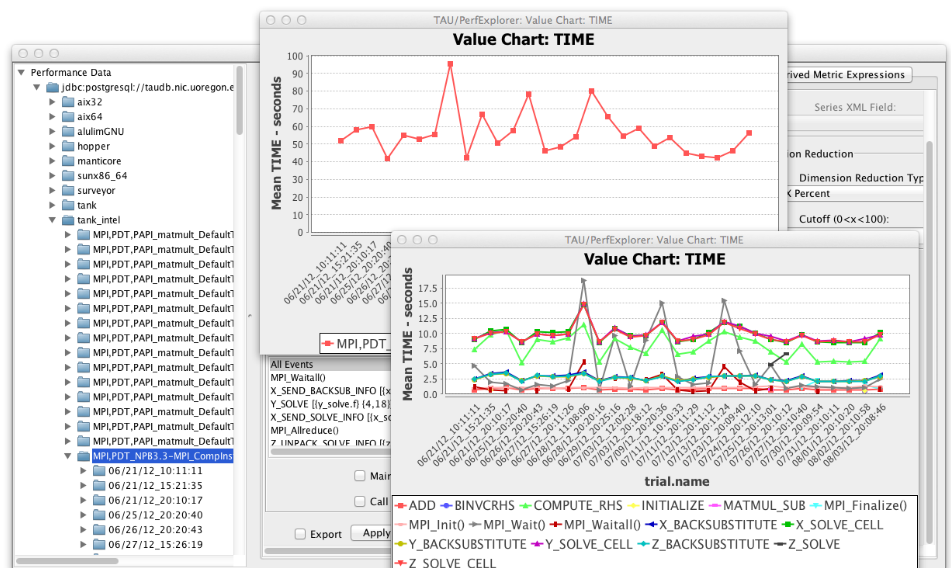
TAUdb Programming APIs

- Java
  - Original API
  - Basis for in-house analysis tool support
  - Command line tools for batch loading into the database
  - Parses 15+ profile formats
    - TAU, gprof, Cube, HPCT, mpiP, DynaProf, PerfSuite, …
  - Supports Java embedded databases (H2, Derby)
- C programming interface under development
  - PostgreSQL support first, others as requested
  - Query Prototype developed
  - Plan full-featured API: Query, Insert, & Update
  - Evaluating SQLite support

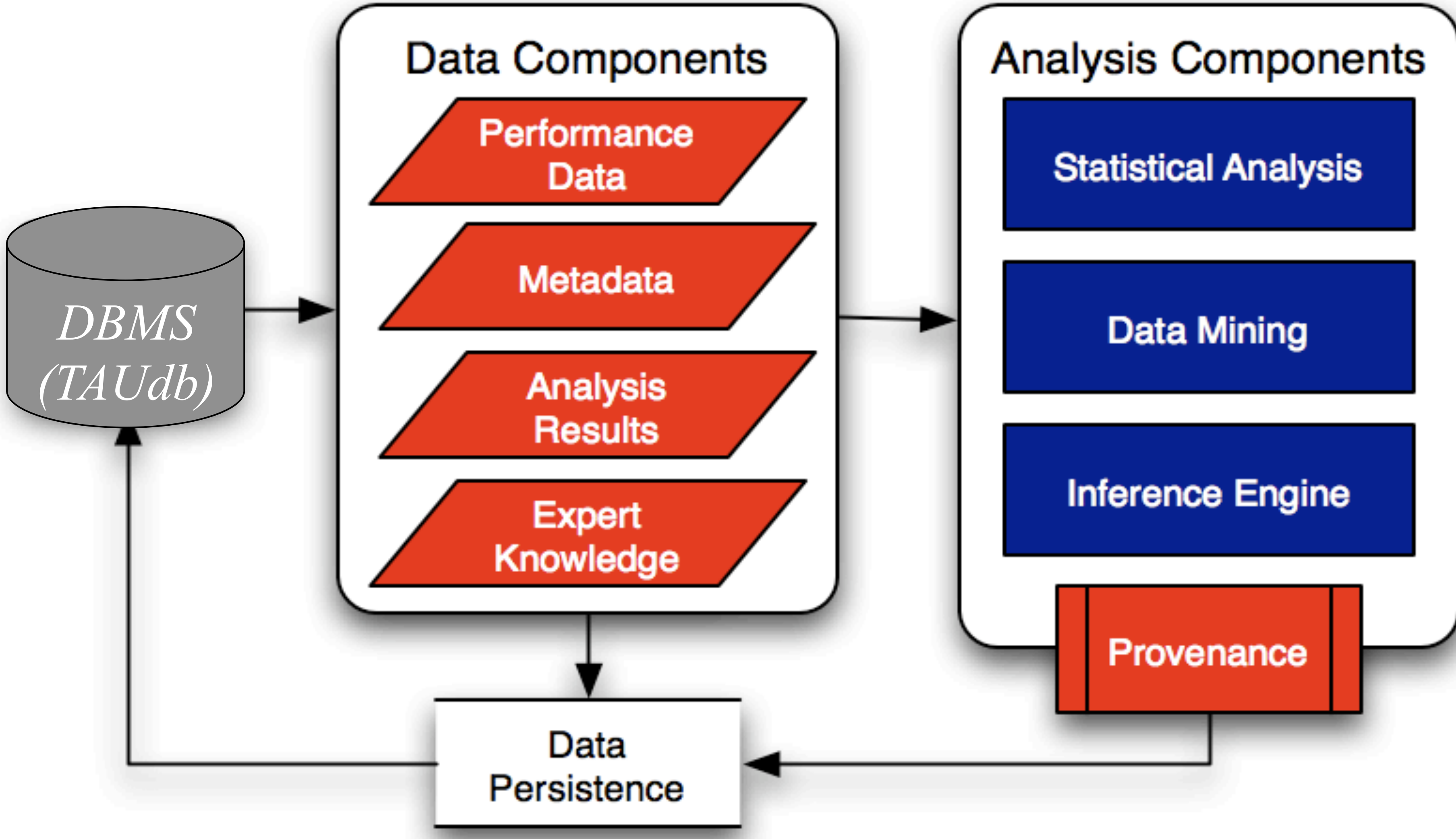
TAUdb Tool Support

- ParaProf
  - Parallel profile viewer / analyzer
  - 2, 3+D visualizations
  - Single experiment analysis
- PerfExplorer
  - Data mining framework
    - Clustering, correlation
  - Multi-experiment analysis
  - Scripting engine
  - Expert system



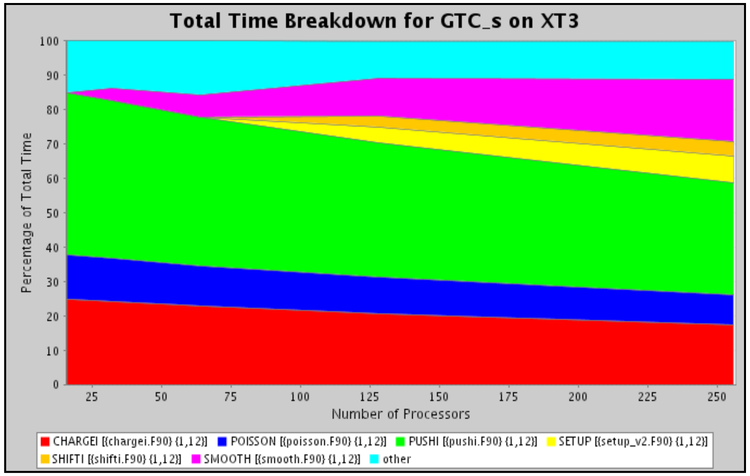


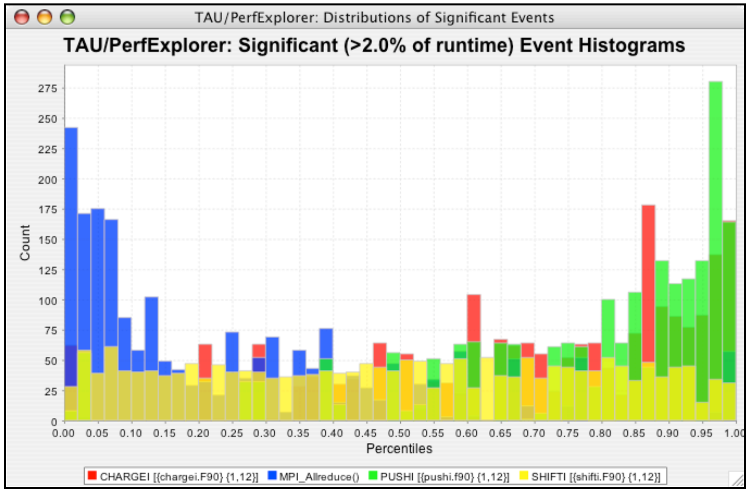
PerfExplorer



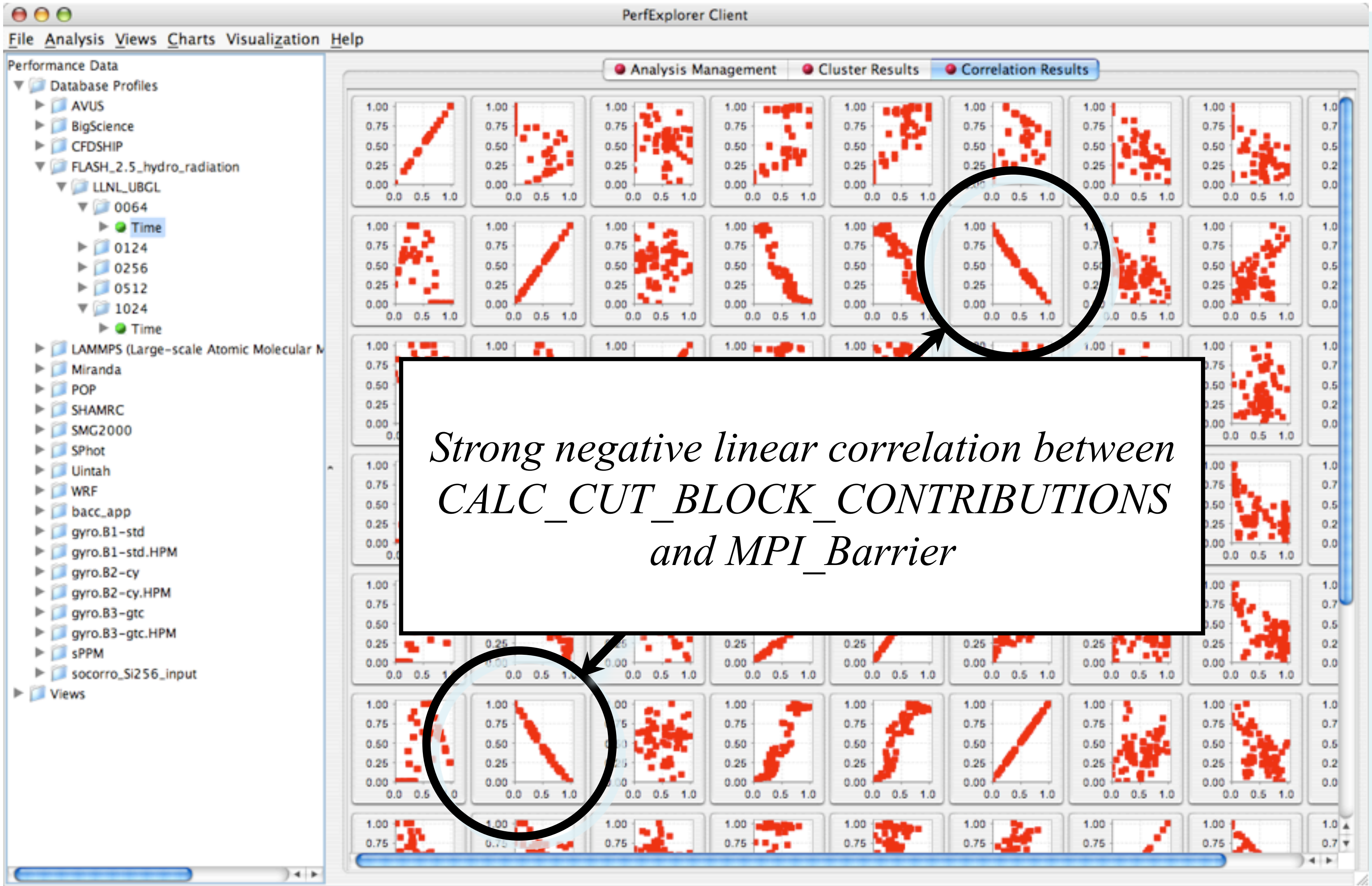
PerfExplorer – Relative Comparisons

- Total execution time
- Timesteps per second
- Relative efficiency
- Relative efficiency per event
- Relative speedup
- Relative speedup per event
- Group fraction of total
- Runtime breakdown
- Correlate events with total runtime
- Relative efficiency per phase
- Relative speedup per phase
- Distribution visualizations



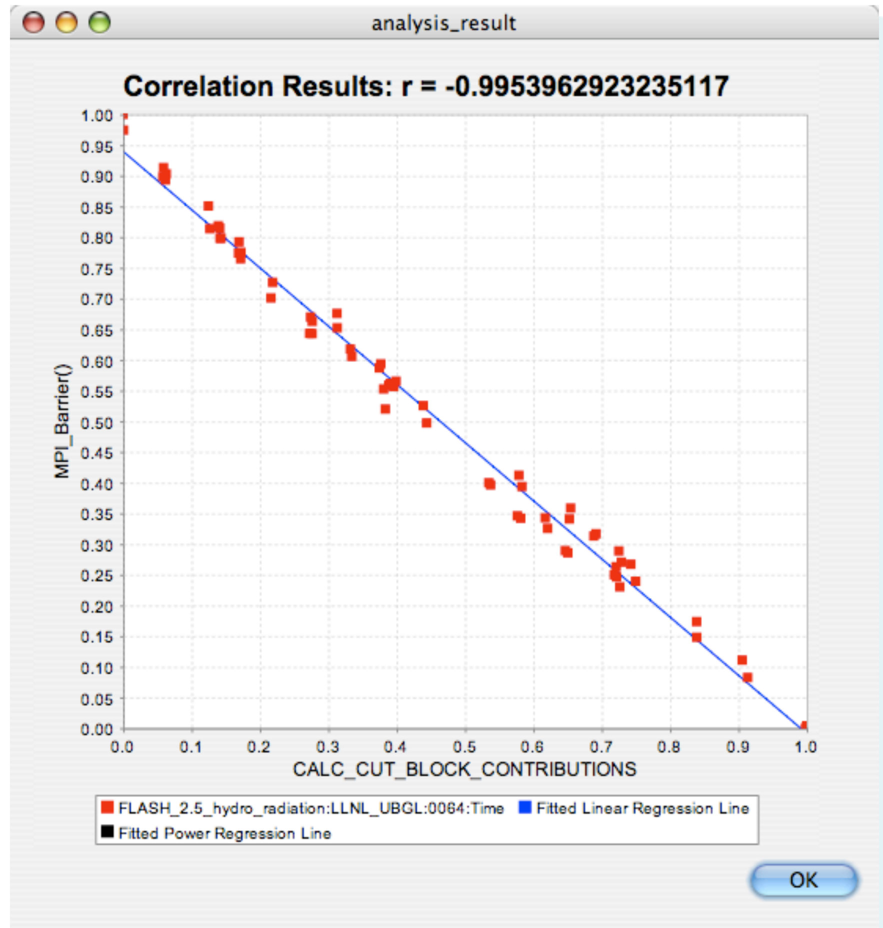


PerfExplorer – Correlation Analysis

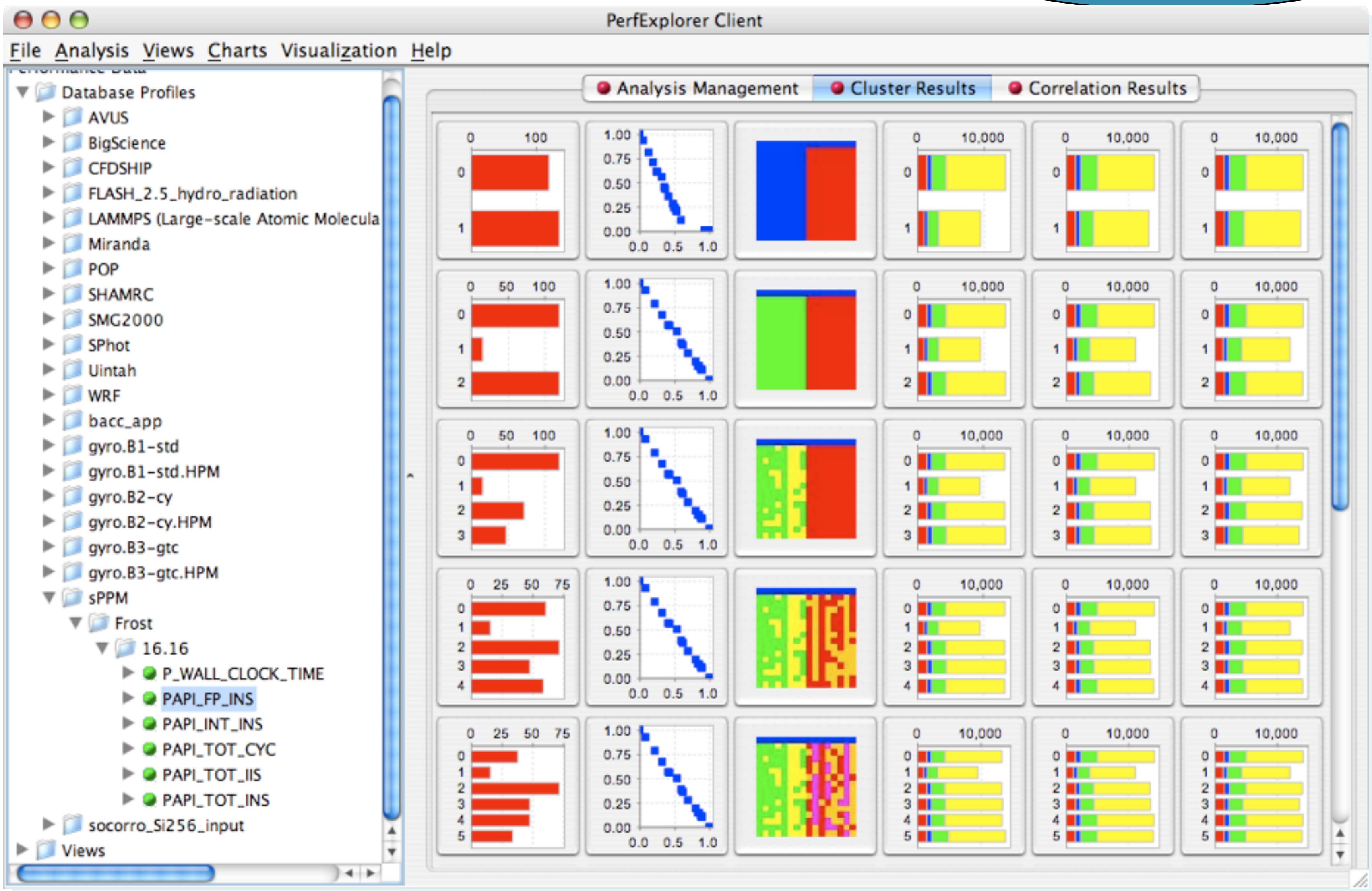


PerfExplorer – Correlation Analysis

- -0.995 indicates strong, negative relationship. As CALC_CUT_BLOCK_CONTRIBUTIONS() increases in execution time, MPI_Barrier() decreases

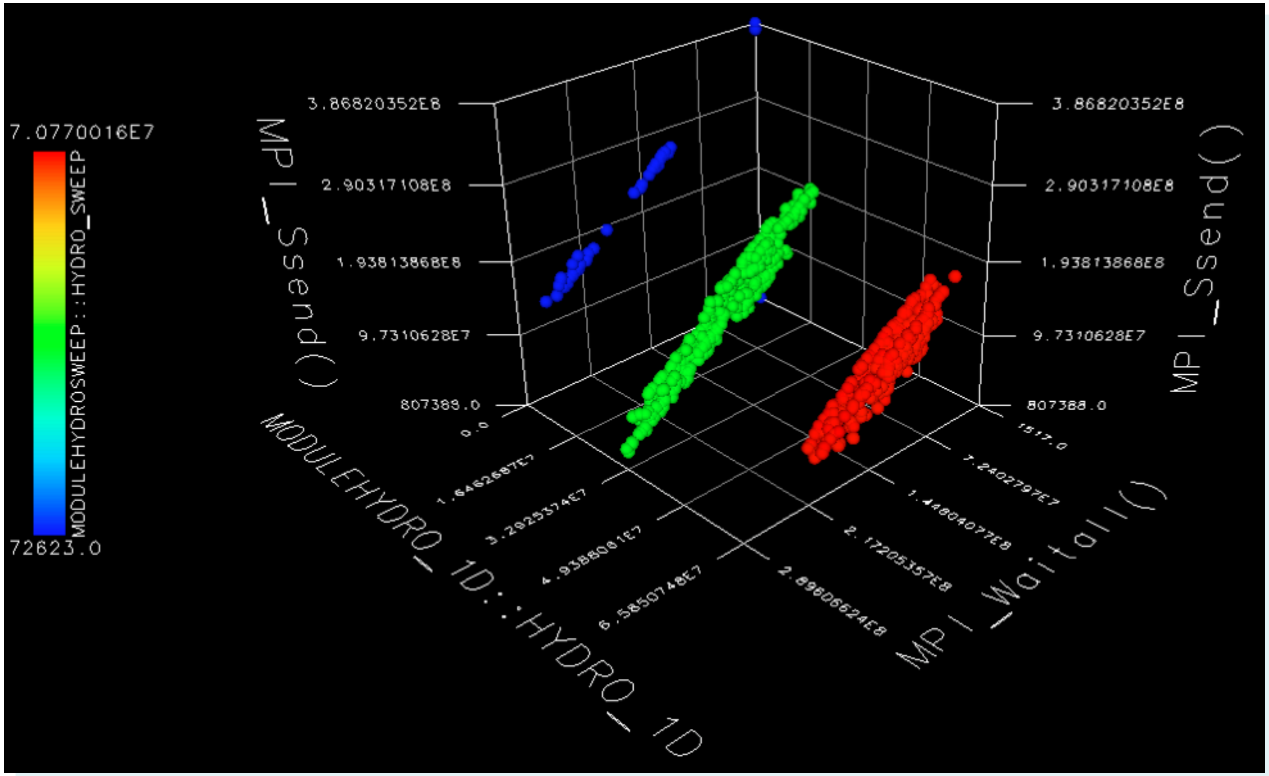


PerfExplorer – Cluster Analysis

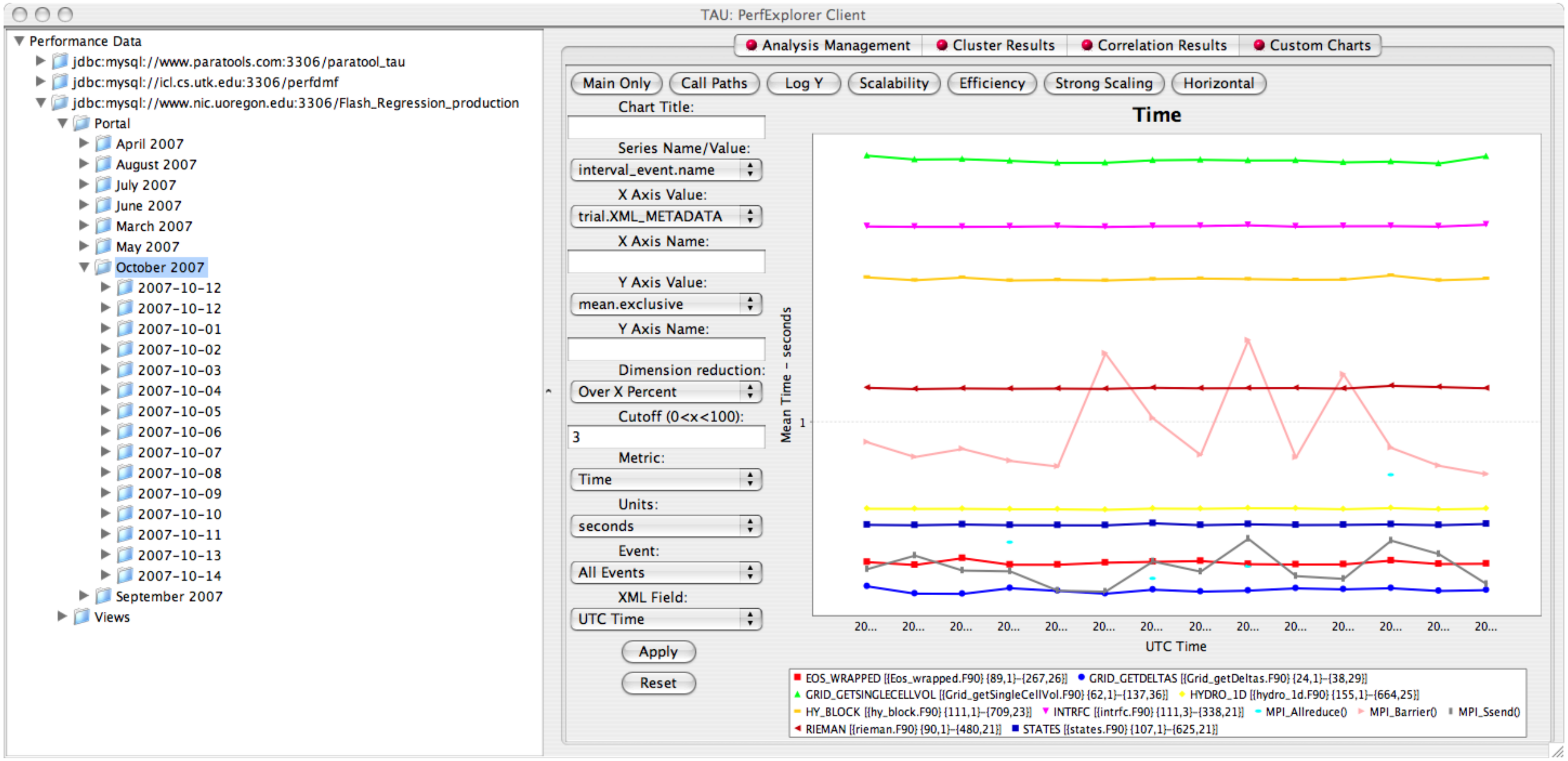


PerfExplorer – Cluster Analysis

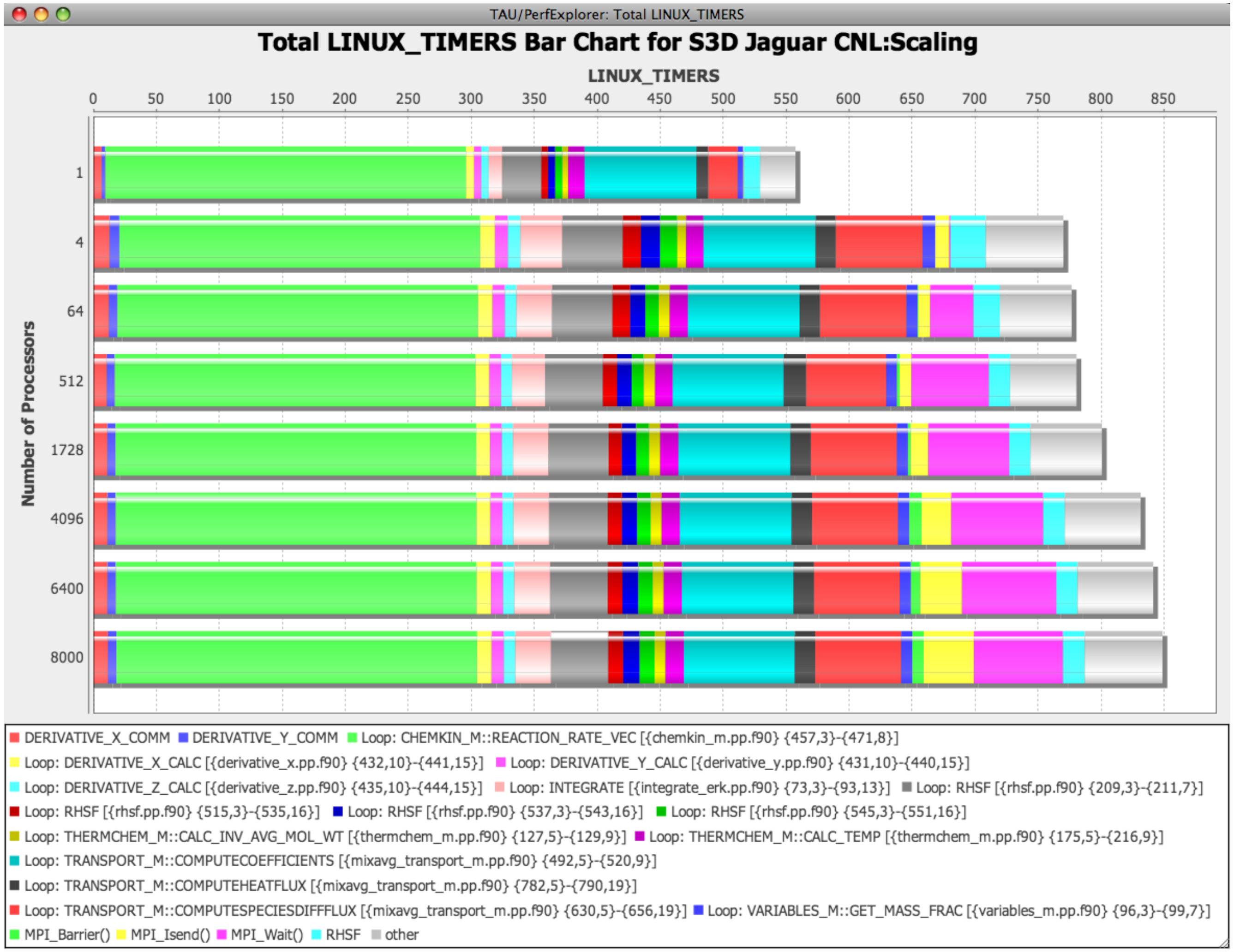
- Four significant events automatically selected
- Clusters and correlations are visible



PerfExplorer – Performance Regression



Usage Scenarios: Evaluate Scalability



PerfExplorer Scripting Interface

- Control PerfExplorer analyses with Python scripts.
  - Perform built-in PerfExplorer analyses.
  - Call machine learning routines in Weka.
  - Export data to R for analysis.

```
Utilities.setSession("peri_s3d")
trial = Utilities.getTrial("S3D", "hybrid-study", "hybrid")
result = TrialResult(trial)

reducer = TopXEvents(result1, 10)
reduced = reducer.processData().get(0)

for metric in reduced.getMetrics():
        k = 2
        while k<= 10:
                kmeans = KMeansOperation(reduced, metric,
                          AbstractResult.EXCLUSIVE, k)
                kmeans.processData()
```

Using TAU in an Autotuning Workflow

- Active Harmony proposes variant.
- Instrument code variant with TAU
  - Captures time measurements and hardware performance counters
    - Interfaces for PAPI, CUPTI, etc.
  - Captures metadata describing execution environment
    - OS name, version, release, native architecture, CPU vendor, ID, clock speed, cache sizes, # cores, memory size, etc. plus user-defined metadata
- Save performance profiles into TAUdb
  - Profiles tagged with provenance metadata describing which parameters produced this data.
- Repeat autotuning across machines/architectures and/or datasets.
- Analyze stored profiles with PerfExplorer.

Multi-Parameter Profiling

- Added multi-parameter-based profiling in TAU to support specialization
  - User can select which parameters are of interest using a selective instrumentation file
- Consider a matrix multiply function
  - We can generate profiles based on the dimensions of the matrices encountered during execution:

e.g., for `void matmult(float **c, float **a, float **b, int L, int M, int N)`, parameterize using L, M, N

Using Parameterized Profiling in TAU

BEGIN_INCLUDE_LIST matmult

BEGIN_INSTRUMENT_SECTION

loops file="foo.c" routine="matrix#"

param file="foo.c" routine="matmult" param="L" param="M" param="N"

END_INSTRUMENT_SECTION

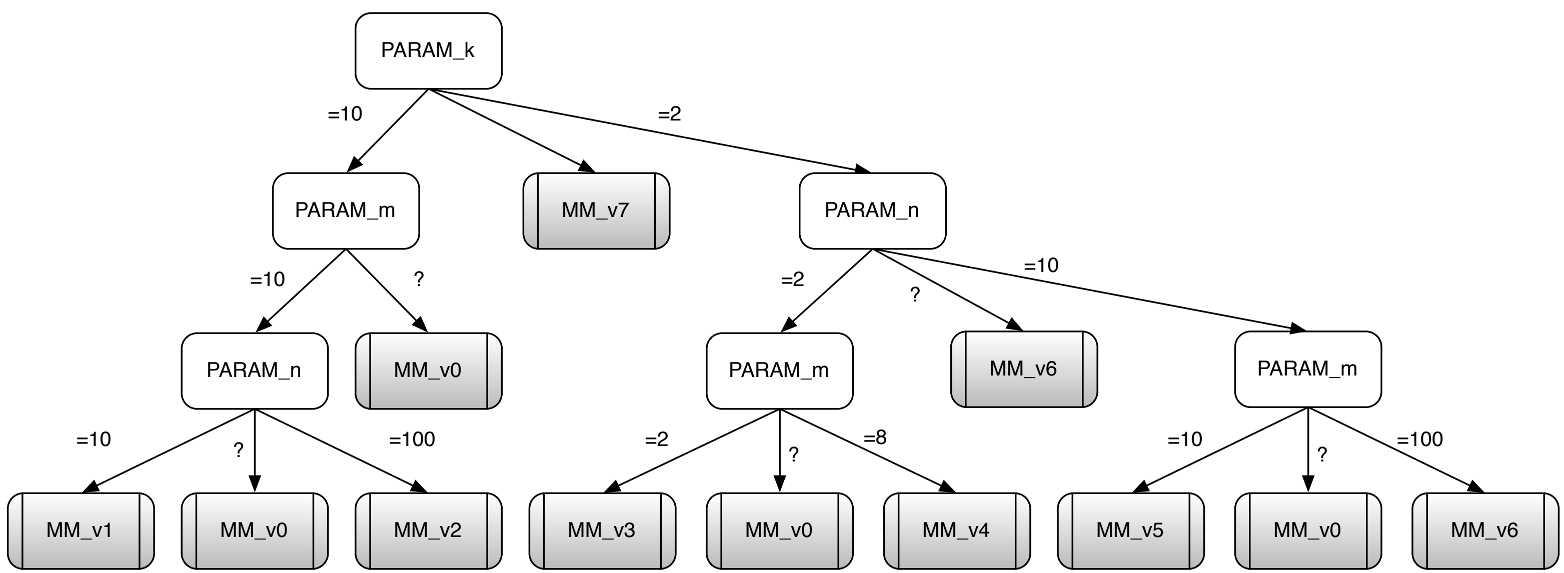
```
int matmult(float **, float **, float **, int, int, int)
<L=100, M=8, N=8> C
```

```
int matmult(float **, float **, float **, int, int, int)
<L=10, M=100, N=8> C
```

```
int matmult(float **, float **, float **, int, int, int)
<L=10, M=8, N=8> C
```

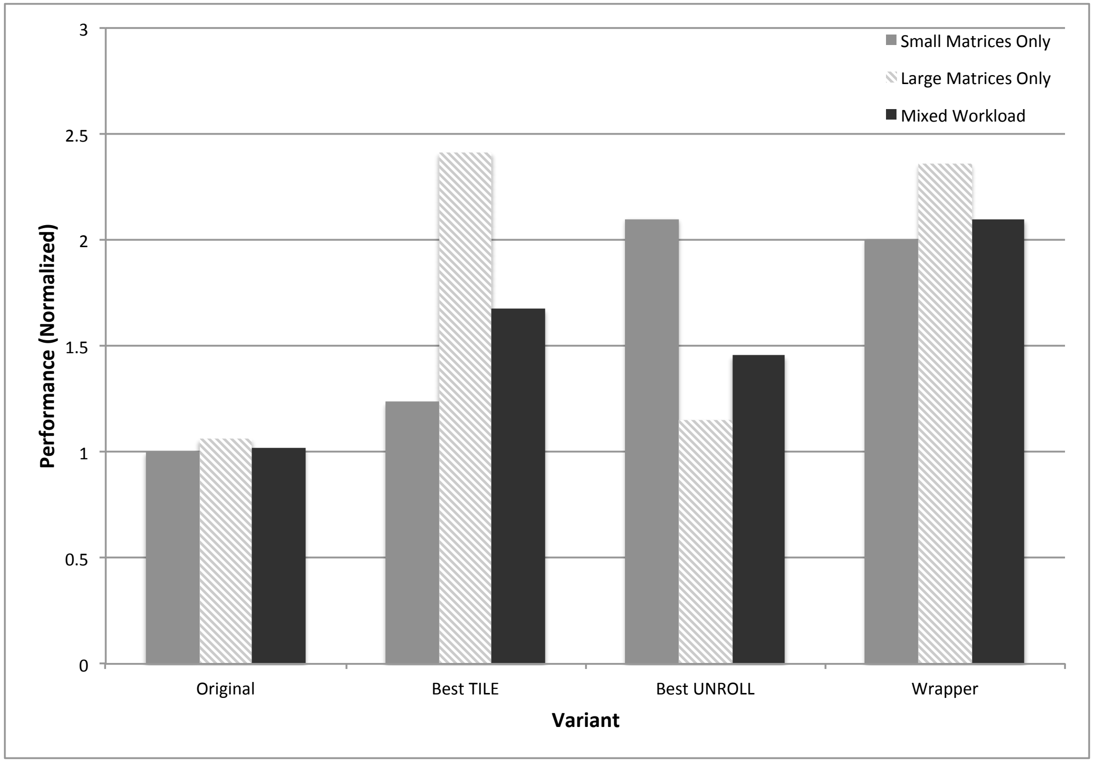
Specialization using Decision-Tree Learning

- For a matrix multiply kernel:
  - Given a dataset containing matrices of different sizes
  - and for which some matrix sizes are more common than others
  - automatically generate function to select specialized variants at runtime based on matrix dimensions



Specialization using Decision-Tree Learning

- For a matrix multiply kernel:
  - Given a dataset containing matrices of different sizes
  - and for which some matrices are small enough to fit in the cache, while others do not
  - automatically generate function to select specialized variants at runtime based on matrix dimensions

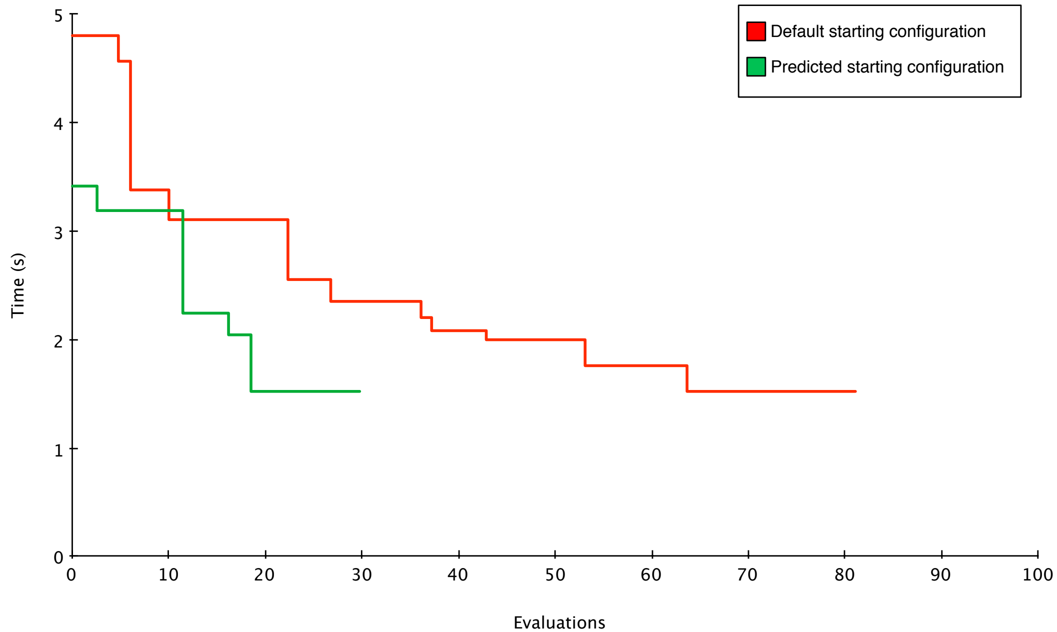


Initial Configuration Selection

- Speed autotuning search process by learning classifier to select an initial configuration.
- When starting out autotuning a new code:
  - Use default initial configuration
  - Capture performance data into TAUdb
- Once sufficient data is collected:
  - Generate classifier
- On subsequent autotuning runs:
  - Use classifier to propose an initial configuration for search

Initial Configuration Selection Example

- Matrix multiplication kernel in C
- CUDA code generated using CUDA-CHiLL
- Tuned on several different NVIDIA GPUs.
  - S1070, C2050, C2070, GTX480
- Learn on data from three GPUs, test on remaining one.
- Results in reduction in evaluations required to converge.



Ongoing Work

- Guided Search
  - We choose an initial configuration largely because this was easy to implement — Active Harmony already provided the functionality to specify this.
  - With the Active Harmony plugin interface, we could provide input beyond the first step of the search.
    - e.g, at each step, incorporate newly acquired data into the classifier and select a new proposal.

Ongoing Work

- Real applications!
  - So far we have only used kernels in isolation.
  - Currently working on tuning OpenCL derived field generation routines in VisIt visualization tool.
  - Cross-architecture: x86, NVIDIA GPU, AMD GPU, Intel Xeon Phi

Acknowledgments – U. Oregon

- ***Prof. Allen D. Malony, Professor CIS, and Director NeuroInformatics Center***
- ***Dr. Sameer Shende, Director, Performance Research Lab***
- ***Dr. Kevin Huck***
- ***Wyatt Spear***
- ***Scott Biersdorff***
- ***David Ozog***
- ***David Poliakoff***
- ***Dr. Robert Yelle***
- ***Dr. John Linford, ParaTools, Inc.***

Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science
  - ASC/NNSA
- Department of Defense (DoD)
  - HPC Modernization Office (HPCMO)
- NSF Software Development for Cyberinfrastructure (SDCI)
- Research Centre Juelich
- Argonne National Laboratory
- Technical University Dresden
- ParaTools, Inc.
- NVIDIA