Integrating Knowledge, Automation, and Persistence with PerfExplorer and PerfDMF

Kevin A. Huck

khuck@cs.uoregon.edu

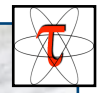
http://tau.uoregon.edu

Department of Computer and Information Science

Performance Research Laboratory

University of Oregon

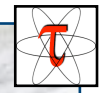
Outline



- Parallel performance data mining
- PerfExplorer as application
- PerfExplorer as framework
  - Data management framework
  - Data mining framework
  - Automation
  - Knowledge capture
  - Expert system
- Recent Work as Framework
  - OpenUH Compiler Feedback

Emphasis on modularity and programmability

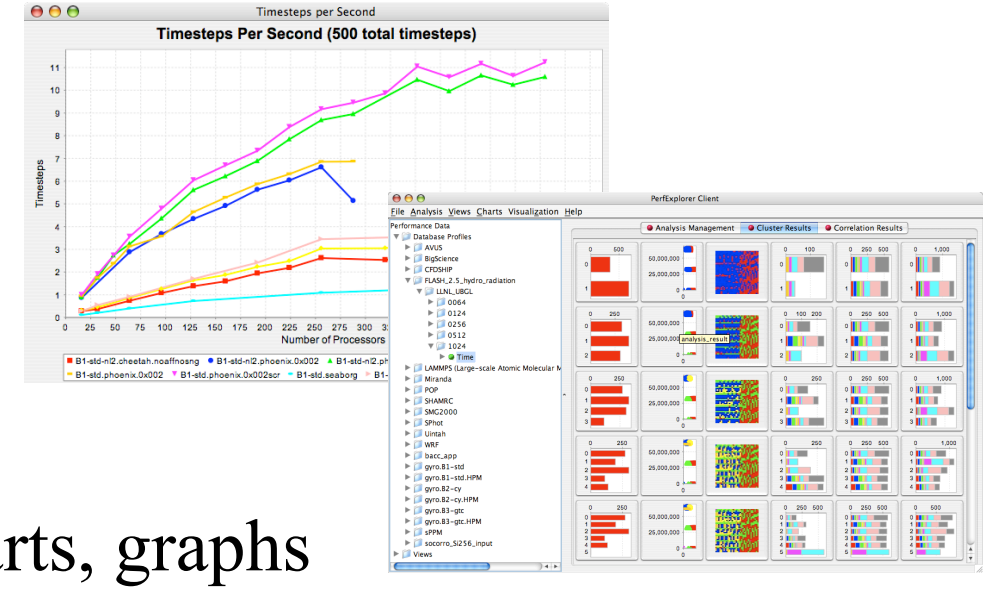
Motivation for Performance Data Mining

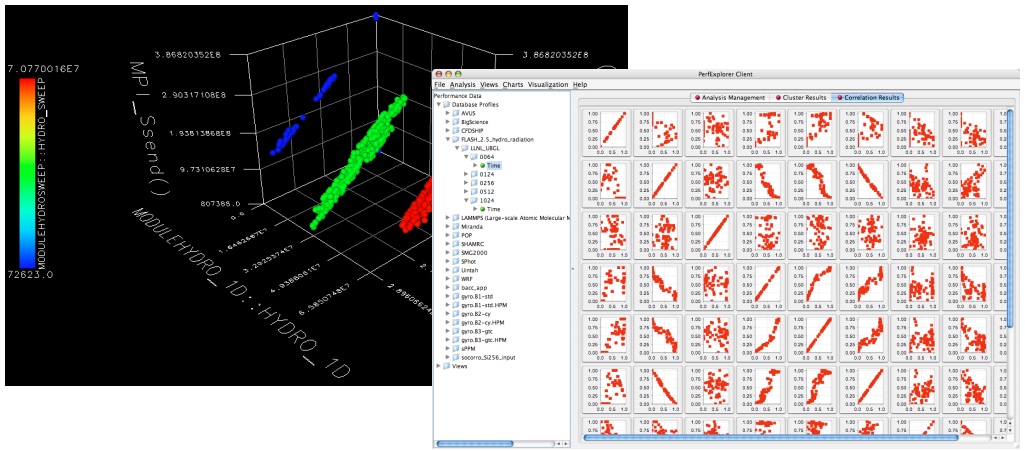


- Overwhelming complexity in parallel profile analysis
  - Events
  - Scale
- Unmanaged collections of performance data
  - Directories of profiles
- No easy way to perform parametric studies
- Terascale, petascale performance profiles difficult to:
  - Manage
  - Visualize
  - Analyze
- Needed a way to explore the data in new ways

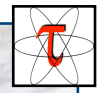
PerfExplorer As Application

- Java GUI Application
- Data browsing
  - Custom views
- Metadata browsing
  - XML tree table
- Parametric studies, charts, graphs
- 3-D visualization of performance profiles
- Data mining
  - Clustering
  - Correlation
  - Dimension reduction



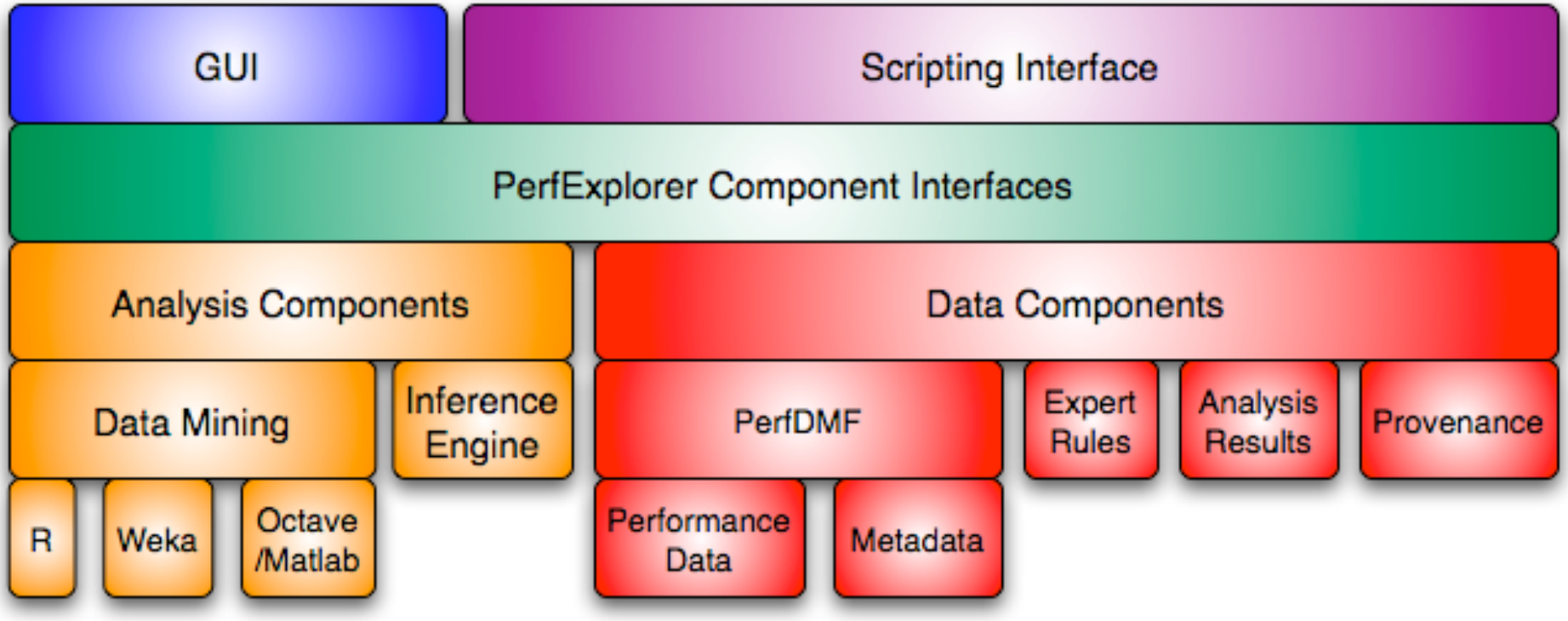


PerfExplorer As Framework

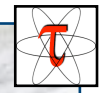


- Transition from “tool” to “toolbox”
- Automation
  - Repeatable analysis
  - Chain operations together as workflows
- Persistence
  - Option to save intermediate / derived results
  - Reuse cached results
- Provenance
  - Information on how intermediate / final results created
- Knowledge Capture / Reuse
  - Parallel performance profile expert system

PerfExplorer v2: Architecture

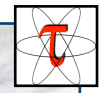


Performance Data Management



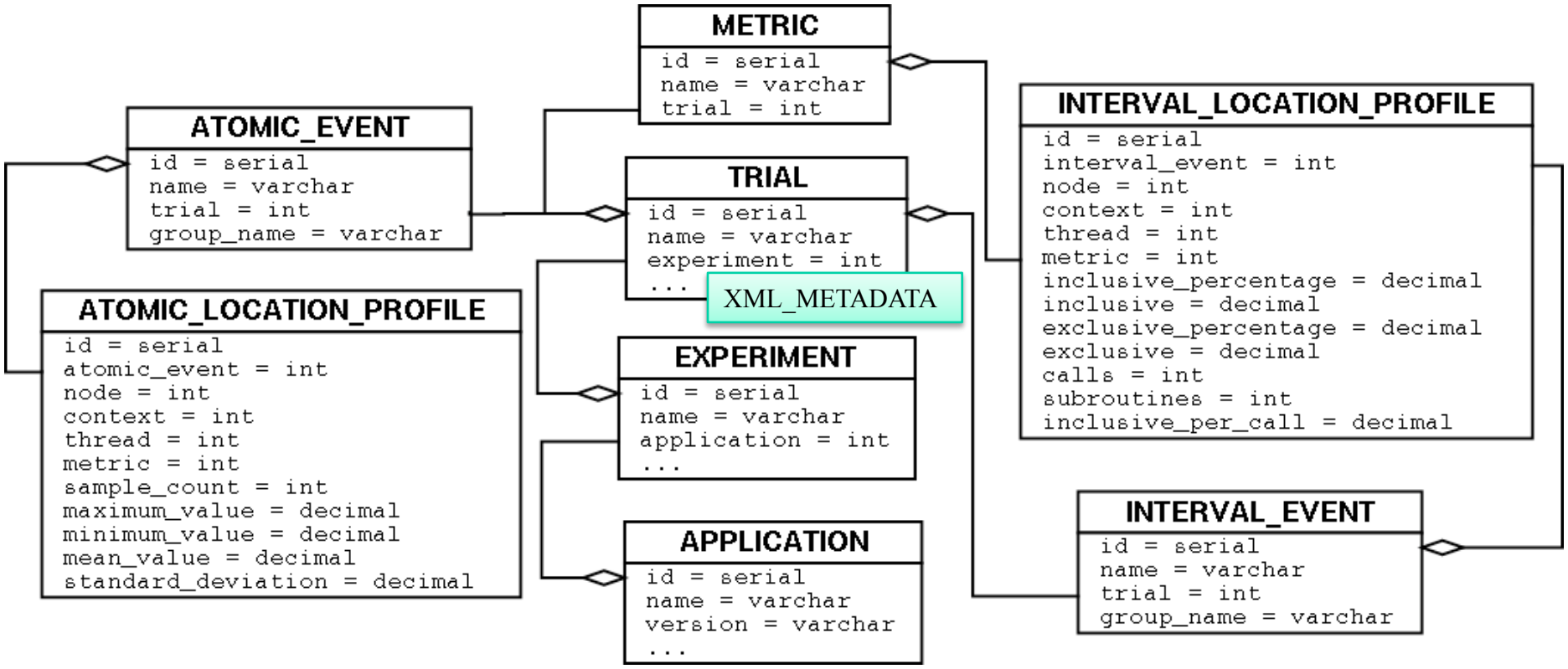
- Need for robust processing and storage of multiple profile performance data sets
- Avoid developing independent data management solutions
  - Waste of resources
  - Incompatibility among analysis tools
- Goals
  - Foster multi-experiment performance evaluation
  - Develop a common, reusable foundation of performance data storage, access and sharing
  - A core module in an analysis system, and/or as a central repository of performance data

PerfDMF Approach



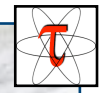
- Performance Data Management Framework
- Extensible toolkit to promote integration and reuse across available performance tools
  - Supported profile formats:
    TAU, CUBE, Dynaprof, HPC Toolkit (Rice), HPC Toolkit (IBM), gprof, mpiP, psrun (PerfSuite), Open|SpeedShop, OMPP, GPTL, IPM, PERI-XML, Paraver
  - PostgreSQL, MySQL, Oracle, DB2, Derby/Cloudscape
  - Profile query and analysis API
- Data can be imported from / exported to PERI-DB
  - PERI SciDAC project (UTK, NERSC, UO, PSU, TAMU)

PerfDMF Schema



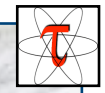
All supported data formats map to this schema

Metadata Collection



- Integration of XML metadata for each profile
- Ways to incorporate metadata
  - Measured hardware/system information
    - CPU speed, memory in GB, MPI node IDs, …
  - Application instrumentation (application-specific)
    - Application parameters, input data, domain decomposition
  - PerfDMF data management tools can incorporate an XML file of additional metadata
    - Compiler flags, submission scripts, input files, …

XML Metadata File Example



```
<?xml version="1.0" encoding="UTF-8"?>
<tau:metadata xmlns:tau="http://www.cs.uoregon.edu/research/tau">
  <tau:CommonProfileAttributes>
    <tau:attribute>
      <tau:name>build:module.F90</tau:name>
      <tau:value>qk-pgf90 -I.  -fastsse -c module.inst.F90     -I/spin/home/khuck/src/tau2
        /include -I/opt/xt-mpt/default/mpich2-64/P2/include -o module.o</tau:value>
    </tau:attribute>
    <tau:attribute>
      <tau:name>buildenv:XNLSPATH</tau:name>
      <tau:value>/usr/X11R6/lib/X11/nls</tau:value>
    </tau:attribute>
    <tau:attribute>
      <tau:name>job:#PBS -l</tau:name>
      <tau:value>walltime=0:45:00,size=512</tau:value>
    </tau:attribute>
    <tau:attribute>
      <tau:name>input:mstep</tau:name>
      <tau:value>100</tau:value>
    </tau:attribute>
  </tau:CommonProfileAttributes>
</tau:metadata>
```

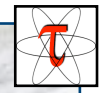
Build data

Environment data

Job data

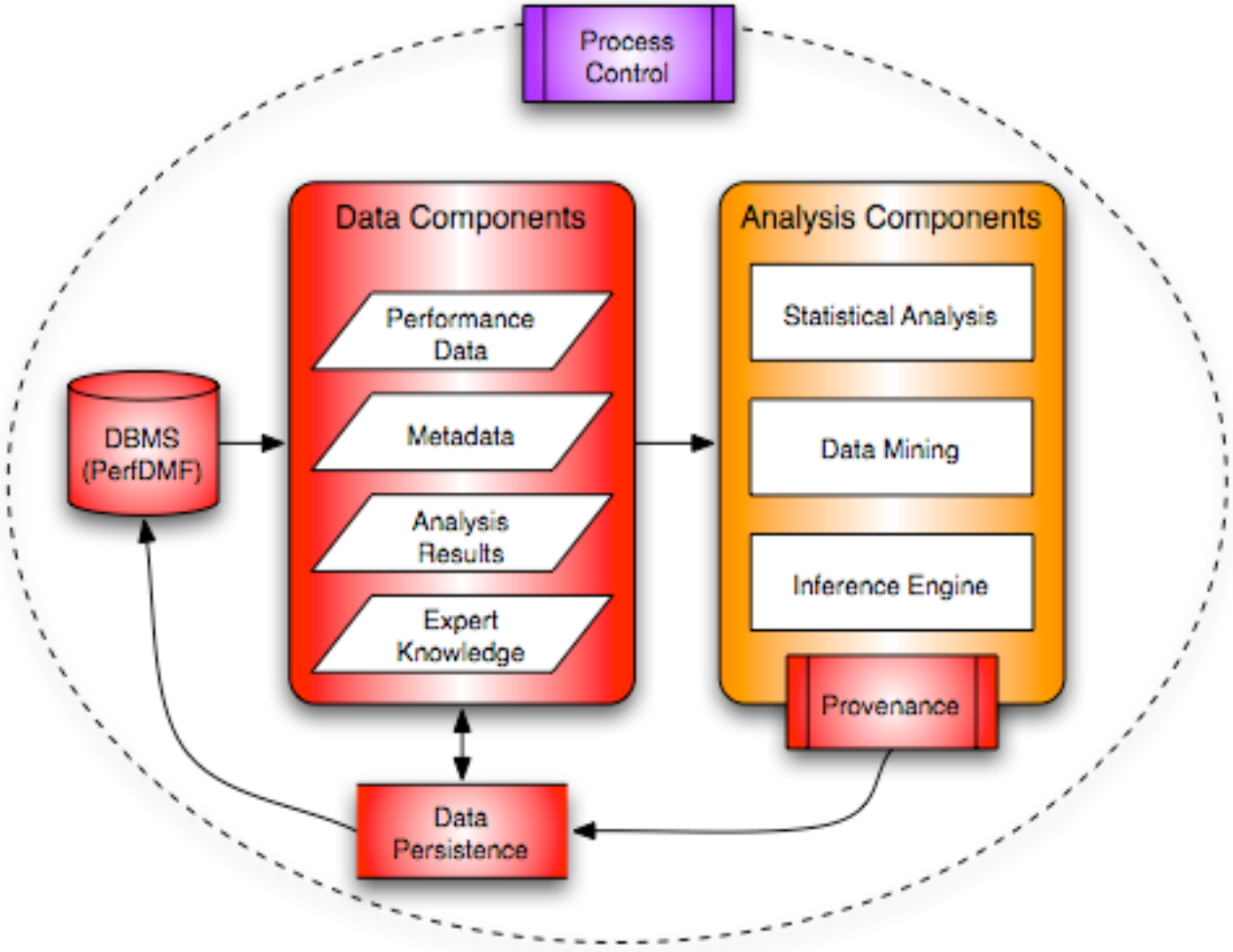
Input file data

PerfExplorer v2 – Requirements and Features

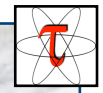


- Component-based analysis process
  - Analysis operations implemented as modules
  - Linked together in analysis process and workflow
- Scripting
  - Provides process/workflow development and automation
- Metadata input, management, and access
- Inference engine
  - Reasoning about causes of performance phenomena
  - Analysis knowledge captured in expert rules
- Persistence of intermediate results
- Provenance
  - Provides historical record of analysis results

PerfExplorer Component Interaction



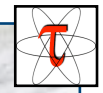
Example Analysis Components



Basic Statistics
Copy
Correlation
Correlation with metadata
Derive metrics
Difference
Extract callpath events
Extract non-callpath events
Extract events
Extract metrics
Extract phases
Extract rank
k-means
Merge trials
PCA
Scalability
Top X events
Top X percent events
ANOVA
Linear regression
Ratios
Non-linear regression*
Backward elimination*
Correlation elimination*

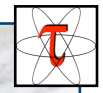
** future development*

Analysis Processes and Workflow Automation



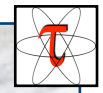
- Create analysis processes from analysis components
  - Repeatable analysis for common analysis workflows
  - Extensible solution with easy accessiblity and reuse
  - Run analysis workflows automatically
- Need programming system for process development
  - Provide full access to PerfDMF and PerfExplorer
  - Chose Jython (Python interpreter written in Java)
    - http://www.jython.org/
    - other languages possible
    - Java objects can execute Python scripts
    - Python scripts can execute Java code

PerfExplorer Script Example



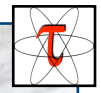
```
# create a rulebase for processing
ruleHarness = RuleHarness.useGlobalRules("openuh/OpenUHRules.drl")

# load a trial
Utilities.setSession("openuh”)
trial = TrialMeanResult(Utilities.getTrial("Fluid Dynamic", "rib 45", "1_8"))

# calculate the derived metric
stalls = "BACK_END_BUBBLE_ALL"
cycles = "CPU_CYCLES"
operator = DeriveMetricOperation(trial, stalls, cycles,
    DeriveMetricOperation.DIVIDE)
derived = operator.processData().get(0)

# compare values to average for application
for event in derived.getEvents():
    MeanEventFact.compareEventToMain(derived, mainEvent, derived, event)

# process the rules
ruleHarness.processRules()
```

Knowledge Inferencing



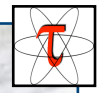
- Knowledge capture / reuse
  - Parallel performance profile expert system
  - Performance expertise captured as inference rules
  - Rules used to help explain performance symptoms
- Need rule development system and execution
  - Chose JBoss Rules (Java-based rules engine)
    - only supports forward-chaining
  - Facts are asserted
  - Rules are processed to ascertain all true assertions
  - Rules fire
    - Java code is executed
    - new facts can be asserted, which can cause new rules to fire

PerfExplorer Rule Example



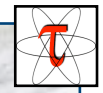
```
rule "High Remote Memory Accesses"
    when
        // there is a low fraction of local memory references
        f : MeanEventFact (
            m : metric == "((L3_MISSES-DATA_EAR_CACHE_LAT128)/L3_MISSES)",
            b : betterWorse == MeanEventFact.LOWER,
            s: severity > 0.02,
            e : eventName,
            a : mainValue,
            v : eventValue,
            factType == "Compared to Main" )
    then
        System.out.println("The event " + e + " has a lower than average fraction of
     local memory references.");
        System.out.println("\tAverage fraction: " + a + ", Event fraction: " + v);
        System.out.println("\tPercentage of total runtime: " + f.getPercentage(s));
end
```

Analysis Examples



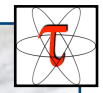
- Correlating performance differences with metadata
- Intelligent scaling analysis (weak, strong)
- L1, L2 hit ratio hotspots
- Memory bound region detection
- Load balance analysis
- Examining stall sources (Itanium HW counters)
- Memory stall source breakdown (Itanium HW counters)
- Phase analysis (i.e. per-iteration analysis)
- Context event analysis (i.e. message sizes)
- Power modeling for optimization (using HW counters)

Example: OpenUH Compiler Feedback



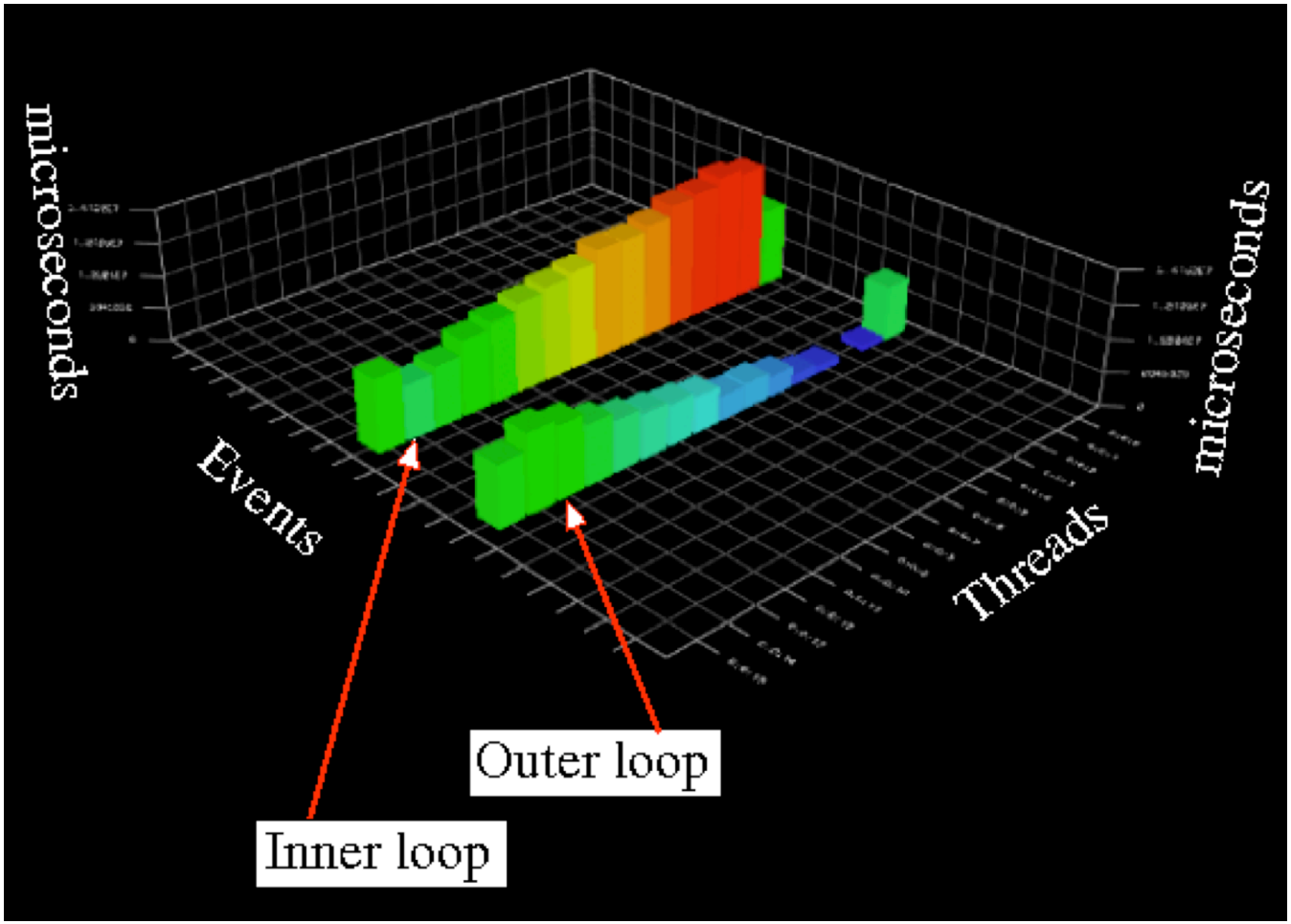
- OpenUH: branch of the open-source Open64 compiler suite for C, C++, Fortran95, OpenMP 2.5
- Wanted to give feedback to the compiler for:
  - Evaluating performance of OpenMP compile and runtime decision making
  - Validating cost model computation in the loop nest optimizer (LNO)
  - Provide runtime-based feedback to the compiler to improve LNO
- Example: Multiple Sequence Alignment
  - OpenMP parallelism with default pragma behavior
  - Switching to dynamic scheduling improves performance

OpenMP loop, before

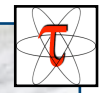


```
#pragma omp for
for(md=first;md<=last;md++) {
  for(nd=md+1;nd<=last;nd++) {
     /*  … do stuff …*/
  }
}
```

Multiple Sequence Alignment, 16 threads

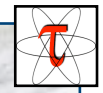


Script to detect conditions for load imbalance



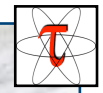
```
ruleHarness = RuleHarness.useGlobalRules("openuh/OpenUHRules.drl")
Utilities.setSession("openuh")
trial = TrialResult(Utilities.getTrial("static", "size.400", "16.threads"))
extractor = ExtractNonCallpathEventOperation(trial)
extracted = extractor.processData().get(0)

statMaker = BasicStatisticsOperation(extracted, False)
stats = statMaker.processData()
stddev = stats.get(BasicStatisticsOperation.STDDEV)
means = stats.get(BasicStatisticsOperation.MEAN)

ratioMaker = RatioOperation(stddev, means)
ratios = ratioMaker.processData().get(0)
for event in ratios.getEvents():
    MeanEventFact.evaluateLoadBalance(means, ratios, event, metric)

extractor = ExtractCallpathEventOperation(trial)
extracted = extractor.processData().get(0)
for event in extracted.getEvents():
    fact = FactWrapper("Callpath name/value", event, None)
    RuleHarness.assertObject(fact)
RuleHarness.getInstance().processRules()
```

Rules to interpret conditions of load imbalance



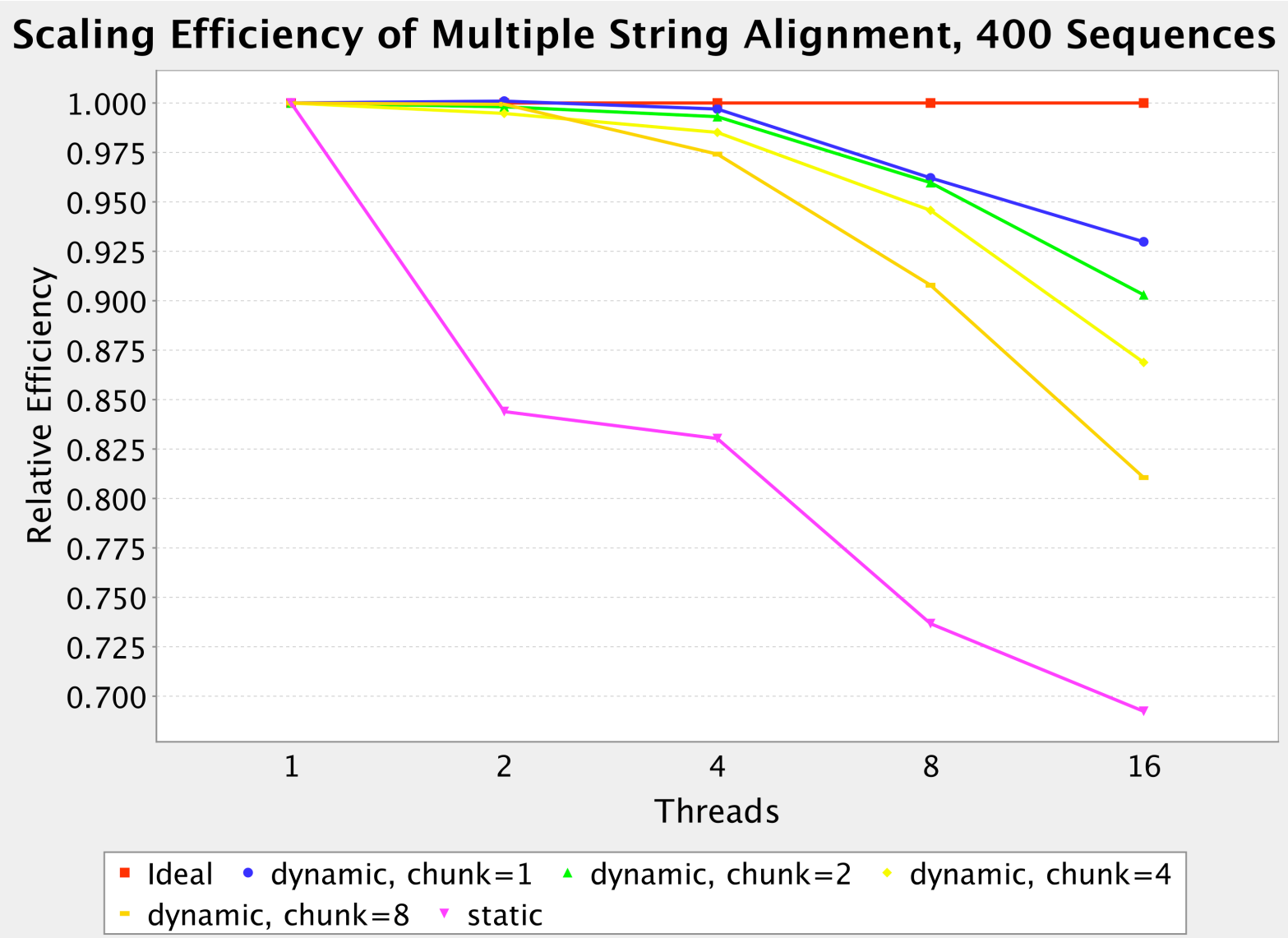
```
rule "Load Imbalance"
    when
        // there is a load imbalance for one event which is a significant event
        f : MeanEventFact (
            s : severity > 0.05, e : eventName,
            factType == "Load Imbalance" )
    then
      assert(new FactWrapper("Imbalanced Event", e, null));
end

rule "New Schedule Suggested"
    when
      f1 : FactWrapper ( factName == "Imbalanced Event", e1 : factType )
      f2 : FactWrapper ( factName == "Imbalanced Event", e2 : factType != e1 )
      f3 : FactWrapper ( factName == "Callpath name/value", e3 : factType )
      eval ( e3.equals( e1 + " => " + e2) )
    then
        System.out.println(e1 + " calls " + e2 + ", and they are both showing
     signs of load imbalance.");
        System.out.println("If these events are in an OpenMP parallel region,
     consider methods to balance the workload, including dynamic instead of
     static work assignment.\n");
end
```

OpenMP loop, before and after

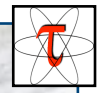


```
#pragma omp for
for(md=first;md<=last;md++) {
   for(nd=md+1;nd<=last;nd++) {
     /*  … do stuff …*/
   }
}

int chunk = 1;
#pragma omp for schedule(dynamic,chunk) nowait
for(md=first;md<=last;md++) {
   for(nd=md+1;nd<=last;nd++) {
     /*  … do stuff …*/
   }
}
```

MSAP Scaling with different chunk sizes



Conclusion, discussion points



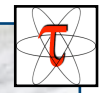
- PerfDMF can (or should) support your profile format (or trace analysis output)
- The PerfExplorer *framework* is configurable, extensible, automated, reusable, available
- Need more Jython scripts for repeatable analysis – how can we encode the processes you repeat often?
- Need more rules to build the expert system – how can we encode the knowledge that you have?
- What does PerfExplorer *not* support that you want?

Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science
    - MICS, Argonne National Lab
  - ASC/NNSA
    - University of Utah ASC/NNSA Level 1
    - ASC/NNSA, Lawrence Livermore National Lab
- Department of Defense (DoD)
  - HPC Modernization Office (HPCMO)
- NSF Software and Tools for High-End Computing
- Research Centre Juelich
- Los Alamos National Laboratory
- TU Dresden
- ParaTools, Inc.

ParaTools

Acknowledgements



- Prof. Allen D. Malony, Advisor
- Dr. Sameer Shende, Director, Performance Research Lab
- Alan Morris, Senior software engineer
- Wyatt Spear, Software engineer
- Scott Biersdorff, Software engineer
- Dr. Matt Sottile, Research Faculty
- Aroon Nataraj, Ph.D. student
- Shangkar Mayanglambam, Ph.D. student
- Brad Davidson, Systems administrator