Prototype Visualization Tools for Multi-Experiment Multiprocessor Performance Analysis of DoD Applications

Patricia Teller, Roberto Araiza, Jaime Nava, and Alan Taylor
University of Texas-El Paso
{pteller,raraiza,jenava,ataylor}@utep.edu

Shirley Moore and David Cronk
University of Tennessee-Knoxville
{shirley,cronk}@cs.utk.edu
Outline

• Motivation – Why do we need performance analysis tools?
• Performance data management
• Performance data analysis via new, platform-independent, easy-to-use, PCAT visualization tools
• Findings
• Conclusions and future work
Outline

• Motivation – Why do we need performance analysis tools?
  • Performance data management
  • Performance data analysis via new, platform-independent, easy-to-use, PCAT visualization tools
• Findings
• Conclusions and future work
Motivation

• Gap between peak and achieved performance in high-end systems
• Tools are needed to
  – Manage massive data sets
  – Identify and analyze relevant information
• So developers can tune performance by
  – Identifying bottlenecks and hotspots
  – Determining causes of bottlenecks
Outline

• Motivation – Why do we need performance analysis tools?
• **Performance data management**
  • Performance data analysis via new, platform-independent, easy-to-use, PCAT visualization tools
• Findings
• Conclusions and future work
Performance Data Management

• Users need a system to
  – Organize data
  – Preserve metadata
  – Facilitate retrieval and analysis
• PETCE and HPCMP defined schemas for all performance data levels
• TAU PerfDMF was extended to support these schemas
TAU PerfDMF

• The underlying relational DB can be
  – MySQL
  – PostgresSQL
  – Added Oracle support
• Also supports data files from
  – gprof
  – hpmcount
  – mpiP
• Serves as focal point for tool interoperability and collaboration on performance tuning
TAU Performance Data Management Framework (PerfDMF)

Possible User Questions

• How does performance vary with different compilers?
• Is poor performance correlated with certain OS features?
• Has a recent change caused unanticipated performance?
• How does performance vary with MPI variants?
• Why is one application version faster than another?
• What is the reason for the observed scaling behavior?
• Did two runs exhibit similar performance?
• How are performance data related to application events?
• Which machines will run my code the fastest and why?
• Which benchmarks predict my code performance best?
Automatic collection of performance data

- Selection of best version
- Refinement of existing optimization strategies
- Creation of new optimization strategies
- Scalability analysis

Operations to compare, integrate, and summarize different experiments

PerfDMF

Unified performance data model

PCAT Tools
CUBE
ParaProf
SvPablo
Vampir

CONE Profile
Dynaprof Profile
EXPERT Profile
SvPablo Profile
TAU Profile

Event Trace

Automatic transformation of event traces into high-level profile

Multiple code versions

Visualization & Analysis

Feedback

Visualization & Analysis

Feedback

Multiple code versions

Automatic collection of performance data

Automatic transformation of event traces into high-level profile

Event Trace

Automatic collection of performance data

Visualization & Analysis

Feedback

Multiple code versions

Automatic collection of performance data

Automatic transformation of event traces into high-level profile

Event Trace

Automatic collection of performance data

Visualization & Analysis

Feedback

Multiple code versions

Automatic collection of performance data

Automatic transformation of event traces into high-level profile

Event Trace
Automatic collection of performance data

Visualization & Analysis

- Selection of best version
- Refinement of existing optimization strategies
- Creation of new optimization strategies
- Scalability analysis

PCAT Tools
CUBE
ParaProf
SvPablo
Vampir

Operations to compare, integrate, and summarize different experiments

PerfDMF

Unified performance data model

CONE Profile
Dynaprof Profile
EXPERT Profile
SvPablo Profile
TAU Profile

Event Trace

Automatic transformation of event traces into high-level profile

Multiple code versions

Automatic collection of performance data
Outline

• Motivation – Why do we need performance analysis tools?
• Performance data management
• Performance data analysis via new, platform-independent, easy-to-use, PCAT visualization tools
  – Example views
• Findings
• Conclusions and future work
PCAT Visualization Tools - 1

- Platform-independent (written in Java)
- Provide multiple views of performance data and derived metrics
- Views controlled by GUIs
- Used to analyze performance data collected for two DoD codes, SHAMRC and LAMMPS
PCAT Visualization Tools - 2

• **Database Query Tool**
  – Downloads data from TAU PerfDMF or other database using the same schema

• **Color Tree Viewer**
  – Enables user to locate “hot code regions” in terms of selected metrics

• **2-D Visualizer**
  – Provides quick comparison of different runs of a program on different numbers of processors
PCAT Visualization Tools - 3

• Comparator
  – Compares two different runs of a program
  – Enables user to rapidly spot performance differences in terms of a user-selected metric

• 3-D Visualizer
  – Permits analysis of one or more metrics in terms of function(s) and processors
  – Provides user with a means to view a large data set in a multiprocessor context
PCAT Database Query Tool
Selecting SHAMRC Performance Data

PCAT Color Tree Viewer

SHAMRC: Loop-level Time Profile – 1 Processor – Initial View

PCAT Color Tree Viewer

SHAMRC: Loop-level Time Profile – 1 Processor – More Detail

PCAT Color Tree Viewer
SHAMRC: Loop-level Time Profile – 1 Processor – More Detail for H2
PCAT Color Tree Viewer
SHAMRC: Time Profile – 8 Processors
PCAT 2-D Visualizer

SHAMRC: Time vs. Number of Processors

![Plot for TIME across different configurations of 'SHARC'](image)

**Configurations:**
- Line 1: [TAU profiling - share.par - v2 - 1 proc; 1 processors] [TAU profiling - share.par - v2 - 2 proc; 2 processors] [TAU profiling - share.par - v2 - 4 processors]

**Metric Type:**
- Inclusive
- Exclusive

Show Counts for:
- Line 1
PCAT 2-D Visualizer
SHAMRC: Resource Stalls vs. Number of Processors

![Plot for PAPI_RES_STL across different configurations of SHARC](image)

**Configurations:**
- Line 1: [TAU profiling - shared par - 1 proc: 1 processor] [TAU profiling - shared par - 2 proc: 2 processors] [TAU profiling - shared par - 4 proc: 4 processors]

PCAT Comparator
SHAMRC: L3 Data Cache Misses – 2 vs. 4 Processors
### PCAT Comparator

**SHAMRC: Total Instructions – Two Program Versions**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Version 1</th>
<th>Version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVBE2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVBY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVEX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHARC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AB1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AB2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3CPY1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUTPUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA_BARRIER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FBC2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPBC1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TTBC2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHRCIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3CPY2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TAU profiling** - sharc.par - v2 - lproc: 1 processor

**TAU profiling** - sharc.par - l proc: 1 processor
PCAT Comparator
SHAMRC: Floating-Point Stalls – Two Program Versions

TAU profiling - sharc.par - v2 - lproc: 1 processor

PCAT 3-D Visualizer
SHAMRC: 2 Processors / 8 Metrics / 9 Functions
PCAT 3-D Visualizer
SHAMRC: 1 Processor (0) / 8 Metrics / 9 Functions
PCAT 3-D Visualizer

SHAMRC: Projection of Processor 0
PCAT 3-D Visualizer
SHAMRC: Projection of Processor 0
Findings - SHAMRC

• Without source code – using performance data and derived metrics – identified potential targets for performance improvement
• Compared two different versions of SHAMRC, a serial version (S) and the first parallel version (P)
  – P and S are the same code except that P has MPI communication calls added
  – S was significantly faster than P
  – Using the comparator, identified significant differences w.r.t. MOVEZ and H1, especially in critical metrics such as
    • floating-point stalls,
    • number of clocks with no instructions completed, and
    • number of clocks with no instructions issued
  – Hypothesis: Change in the memory access pattern
    • delays in getting data from memory
    • a larger number of floating-point and pipeline stalls and, accordingly,
    • increased execution time
• Compared first parallel version (P) with an improved parallel version (P’, which uses an improved Cartesian communicator in MPI)
  – P’ executes faster than P
  – This faster execution time is coupled with fewer floating-point and pipeline stalls (“no instructions completed” and “no instructions issued”)
  – Hypothesis: Memory access pattern was fixed
Findings - LAMMPS

• Without source code – using performance data and derived metrics – identified potential targets for performance improvement
• Using the color viewer, determined that
  – Program execution time = 1.683 E 10 time units (tus)
  – Verlet function execution time = 1.161 E 10 tus
  – Verlet:iterate function (within Verlet)
    • Execution time = 1.160 E 10 tus
    • Calls 16 other functions, many of which involve communication
      – Comm::exchange execution time = 4.685 E 9 tus
      – Neighbor::build execution time = 1.311 E 9 tus
• Using the 3-D viewer, Comm::exchange and Neighbor::build were analyzed further; they generate large numbers of
  – floating-point stalls,
  – branch mispredictions, and
  – resource stalls,
  – which are associated with MPI communication functions, such as MPI_Wait, MPI_Sendrecv, MPI_Send, and MPI_Irecv
Outline

• Motivation – Why do we need performance analysis tools?
• Automated data collection
• Performance data management
• Performance data analysis via new, platform-independent, easy-to-use, PCAT visualization tools
• Findings
• Conclusions and future work
Conclusions

• Performance database technology available for users to upload and store performance data
• Platform-independent, easy-to-use, visualization tools automatically download performance data from database and via GUIs provide multiple views for different data analysis purposes
• Used tools to analyze data from SHAMRC and LAMPPS and without source code, using only performance data and derived metrics, identified potential targets for performance improvement
• For more information, contact pteller@utep.edu or petce@cs.utk.edu
Future Work

• Integrate TAU, PCAT visualization tools, and statistical analysis tools into Eclipse framework