The TAU Performance System

Allen D. Malony
malony@cs.uoregon.edu

Department of Computer and Information Science
Computational Science Institute
University of Oregon
Overview

- Motivation
- Tuning and Analysis Utilities (TAU)
  - Instrumentation
  - Measurement
  - Analysis
  - Performance mapping
- Example
  - PETSc
- Work in progress
- Conclusions
Performance Needs? Performance Technology

- **Performance observability requirements**
  - Multiple levels of software and hardware
  - Different types and detail of performance data
  - Alternative performance problem solving methods
  - Multiple targets of software and system application

- **Performance technology requirements**
  - Broad scope of performance observation
  - Flexible and configurable mechanisms
  - Technology integration and extension
  - Cross-platform portability
  - Open, layered, and modular framework architecture
Complexity Challenges for Performance Tools

- Computing system environment complexity
  - Observation integration and optimization
  - Access, accuracy, and granularity constraints
  - Diverse/specialized observation capabilities/technology
  - Restricted modes limit performance problem solving

- Sophisticated software development environments
  - Programming paradigms and performance models
  - Performance data mapping to software abstractions
  - Uniformity of performance abstraction across platforms
  - Rich observation capabilities and flexible configuration
  - Common performance problem solving methods
General Problems (Performance Technology)

How do we create robust and ubiquitous performance technology for the analysis and tuning of parallel and distributed software and systems in the presence of (evolving) complexity challenges?

How do we apply performance technology effectively for the variety and diversity of performance problems that arise in the context of complex parallel and distributed computer systems?
Computation Model for Performance Technology

- How to address dual performance technology goals?
  - Robust capabilities + widely available methodologies
  - Contend with problems of system diversity
  - Flexible tool composition/configuration/integration

- Approaches
  - Restrict computation types / performance problems
    - limited performance technology coverage
  - Base technology on abstract computation model
    - general architecture and software execution features
    - map features/methods to existing complex system types
    - develop capabilities that can adapt and be optimized
General Complex System Computation Model

- **Node**: physically distinct shared memory machine
- Message passing *node interconnection network*
- **Context**: distinct virtual memory space within node
- **Thread**: execution threads (user/system) in context
TAU Performance System Framework

- **Tuning and Analysis Utilities**
- **Performance system framework** for scalable parallel and distributed high-performance computing
- **Targets a general complex system computation model**
  - nodes / contexts / threads
  - Multi-level: system / software / parallelism
  - Measurement and analysis abstraction
- **Integrated toolkit** for performance instrumentation, measurement, analysis, and visualization
  - Portable **performance profiling/tracing facility**
  - Open software approach
- **University of Oregon, LANL, FZJ Germany**
Definitions – Instrumentation

☞ Instrumentation

☞ Insertion of extra code (hooks) into program

☞ Source instrumentation

☞ done by compiler, source-to-source translator, or manually
  + portable
  + links back to program code
  – re-compile is necessary for (change in) instrumentation
  – requires source to be available
  – hard to use in standard way for mix-language programs
  – source-to-source translators hard to develop (e.g., C++, F90)

☞ Object code instrumentation

☞ “re-writing” the executable to insert hooks
Definitions – Instrumentation (continued)

- **Dynamic** code instrumentation
  - a debugger-like instrumentation approach
  - executable code instrumentation on running program
  - **DynInst** and **DPCL** are examples
  +/– opposite compared to source instrumentation
- **Pre-instrumented** library
  - typically used for MPI and PVM program analysis
  - supported by link-time **library interposition**
  + easy to use since only re-linking is necessary
  – can only record information about library entities
Flexible instrumentation mechanisms at multiple levels

- Source code
  - Manual
  - automatic
    - Program Database Toolkit (PDT)
    - OpenMP directive rewriting (Opari)
- Object code
  - pre-instrumented libraries (e.g., MPI using PMPI)
  - statically linked and dynamically linked
- Executable code
  - dynamic instrumentation (pre-execution) (DynInstAPI)
  - Java virtual machine instrumentation using (JVMPI)
**TAU Instrumentation Approach**

- Targets common measurement interface
  - **TAU API**
- Object-based design and implementation
  - Macro-based, using constructor/destructor techniques
  - Program units: function, classes, templates, blocks
  - Uniquely identify functions and templates
    - name and type signature (name registration)
    - static object creates performance entry
    - dynamic object receives static object pointer
    - runtime type identification for template instantiations
- C and Fortran instrumentation variants
- Instrumentation and measurement optimization
Program Database Toolkit (PDT)

- Program code analysis framework
  - develop source-based tools
- High-level interface to source code information
- Integrated toolkit for source code parsing, database creation, and database query
  - Commercial grade front end parsers
  - Portable IL analyzer, database format, and access API
  - Open software approach for tool development
- Multiple source languages
- Automated performance instrumentation tools
  - TAU instrumentor
**PDT Architecture and Tools**

Application / Library

- **C / C++ parser**
- **Fortran 77/90 parser**

Intermediate Language (IL)

- **C / C++ IL analyzer**
- **Fortran 77/90 IL analyzer**

Program Database Files

- **DUCTAPE**
- **PDBhtml**
- **SILOON**
- **CHASM**
- **TAU_instr**

- Program documentation
- Application component glue
- C++ / F90 interoperability
- Automatic source instrumentation

The TAU Performance System

DOE ACTS Workshop, September 2002
PDT Components

☞ Language front end
  ☞ Edison Design Group (EDG): C, C++, Java
  ☞ Mutek Solutions Ltd.: F77, F90
  ☞ Creates an intermediate-language (IL) tree

☞ IL Analyzer
  ☞ Processes the intermediate language (IL) tree
  ☞ Creates “program database” (PDB) formatted file

☞ DUCTAPE (Bernd Mohr, FZJ/ZAM, Germany)
  ☞ C++ program Database Utilities and Conversion Tools Application Environment
  ☞ Processes and merges PDB files
  ☞ C++ library to access the PDB for PDT applications
Definitions – Profiling

Profiling

- Recording of summary information during execution
  - execution time, # calls, hardware statistics, …
- Reflects performance behavior of program entities
  - functions, loops, basic blocks
  - user-defined “semantic” entities
- Very good for low-cost performance assessment
- Helps to expose performance bottlenecks and hotspots
- Implemented through
  - sampling: periodic OS interrupts or hardware counter traps
  - instrumentation: direct insertion of measurement code
Definitions – Tracing

Tracing

- Recording of information about significant points (events) during program execution
  - entering/exiting code regions (function, loop, block, …)
  - thread/process interactions (e.g., send/receive messages)
- Save information in event record
  - timestamp
  - CPU identifier, thread identifier
  - Event type and event-specific information
- Event trace is a time-sequenced stream of event records
- Can be used to reconstruct dynamic program behavior
- Typically requires code instrumentation
TAU Measurement

Performance information
  - Performance events
  - High-resolution timer library (real-time / virtual clocks)
  - General software counter library (user-defined events)
  - Hardware performance counters
    - PCL (Performance Counter Library) (ZAM, Germany)
    - PAPI (Performance API) (UTK, Ptools Consortium)
    - consistent, portable API

Organization
  - Node, context, thread levels
  - Profile groups for collective events (runtime selective)
  - Performance data mapping between software levels
TAU Measurement Options

- Parallel profiling
  - Function-level, block-level, statement-level
  - Supports user-defined events
  - TAU parallel profile database
  - Hardware counts values
  - Multiple counters (new)
  - Callpath profiling (new)

- Tracing
  - All profile-level events
  - Inter-process communication events
  - Timestamp synchronization

- Configurable measurement library (user controlled)
**TAU Measurement System Configuration**

```plaintext
configure [OPTIONS]

- `{c++=<CC>, -cc=<cc>}` Specify C++ and C compilers
- `{pthread, -sproc , -smarts}` Use pthread, SGI sproc, smarts threads
- `openmp` Use OpenMP threads
- `opari=<dir>` Specify location of Opari OpenMP tool
- `{papi , -pcl=<dir>}` Specify location of PAPI or PCL
- `pdt=<dir>` Specify location of PDT
- `{mpiinc=<d>, mpilib=<d>}` Specify MPI library instrumentation
- `-TRACE` Generate TAU event traces
- `-PROFILE` Generate TAU profiles
- `-PROFILECALLPATH` Generate Callpath profiles (1-level)
- `-MULTIPLECOUNTERS` Use more than one hardware counter
- `-CPUTIME` Use usertime+system time
- `-PAPIWALLCLOCK` Use PAPI to access wallclock time
- `-PAPIVIRTUAL` Use PAPI for virtual (user) time …
```
TAU Measurement API

- Initialization and runtime configuration
  - TAU_PROFILE_INIT(argc, argv);
  - TAU_PROFILE_SET_NODE(myNode);
  - TAU_PROFILE_SET_CONTEXT(myContext);
  - TAU_PROFILE_EXIT(message);

- Function and class methods
  - TAU_PROFILE(name, type, group);

- Template
  - TAU_TYPE_STRING(variable, type);
  - TAU_PROFILE(name, type, group);
  - CT(variable);

- User-defined timing
  - TAU_PROFILE_TIMER(timer, name, type, group);
  - TAU_PROFILE_START(timer);
  - TAU_PROFILE_STOP(timer)
TAU Measurement API (continued)

- **User-defined events**
  - TAU_REGISTER_EVENT(variable, event_name);
  - TAU_EVENT(variable, value);
  - TAU_PROFILE_STMT(statement);

- **Mapping**
  - TAU_MAPPING(statement, key);
  - TAU_MAPPING_OBJECT(funcIdVar);
  - TAU_MAPPING_LINK(funcIdVar, key);
  - TAU_MAPPING_PROFILE(funcIdVar);
  - TAU_MAPPING_PROFILE_TIMER(timer, funcIdVar);
  - TAU_MAPPING_PROFILE_START(timer);
  - TAU_MAPPING_PROFILE_STOP(timer);

- **Reporting**
  - TAU_REPORT_STATISTICS();
  - TAU_REPORT_THREAD_STATISTICS();
TAU Analysis

Profile analysis

- Pprof
  - parallel profiler with text-based display
- Racy
  - graphical interface to pprof (Tcl/Tk)
- jRacy
  - Java implementation of Racy

Trace analysis and visualization

- Trace merging and clock adjustment (if necessary)
- Trace format conversion (ALOG, SDDF, Vampir, Paraver)
- Vampir (Pallas) trace visualization
Pprof Command

- pprof [-c|-b|-m|-t|-e|-i] [-r] [-s] [-n num] [-f file] [-l] [nodes]
  -c Sort according to number of calls
  -b Sort according to number of subroutines called
  -m Sort according to msecs (exclusive time total)
  -t Sort according to total msecs (inclusive time total)
  -e Sort according to exclusive time per call
  -i Sort according to inclusive time per call
  -v Sort according to standard deviation (exclusive usec)
  -r Reverse sorting order
  -s Print only summary profile information
  -n num Print only first number of functions
  -f file Specify full path and filename without node ids
  -l nodes List all functions and exit (prints only info about all contexts/threads of given node numbers)
Pprof Output (NAS Parallel Benchmark – LU)

- Intel Quad
- PIII Xeon
- F90 +
- MPICH
- Profile
  - Node
  - Context
  - Thread
- Events
  - code
  - MPI

<table>
<thead>
<tr>
<th>Node</th>
<th>Context</th>
<th>Thread</th>
<th>Time</th>
<th>Exclusive Usec</th>
<th>Inclusive Usec</th>
<th>Call</th>
<th>Subrs</th>
<th>Inclusive Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3:11.293</td>
<td>1</td>
<td>15</td>
<td>applu</td>
</tr>
<tr>
<td>99.6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3,667</td>
<td>3:10.463</td>
<td>3</td>
<td>37517</td>
<td>63487925</td>
</tr>
<tr>
<td>67.1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>491</td>
<td>2:08.326</td>
<td>37200</td>
<td>37200</td>
<td>exchange_1</td>
</tr>
<tr>
<td>44.5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6,461</td>
<td>1:25.159</td>
<td>9300</td>
<td>18600</td>
<td>9157</td>
</tr>
<tr>
<td>41.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1:18.436</td>
<td>1:18.436</td>
<td>9300</td>
<td>18600</td>
<td>4173</td>
</tr>
<tr>
<td>29.5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6,778</td>
<td>56,407</td>
<td>9300</td>
<td>18600</td>
<td>6085</td>
</tr>
<tr>
<td>28.2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>50,142</td>
<td>50,142</td>
<td>9300</td>
<td>18600</td>
<td>2611</td>
</tr>
<tr>
<td>15.2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>24,481</td>
<td>31,031</td>
<td>9300</td>
<td>18600</td>
<td>103036</td>
</tr>
<tr>
<td>13.9</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>7,501</td>
<td>7,501</td>
<td>9300</td>
<td></td>
<td>107jacl</td>
</tr>
<tr>
<td>3.4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>438</td>
<td>6,594</td>
<td>9300</td>
<td></td>
<td>10918</td>
</tr>
<tr>
<td>3.4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4,500</td>
<td>6,590</td>
<td>9300</td>
<td></td>
<td>703</td>
</tr>
<tr>
<td>2.6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4,989</td>
<td>4,989</td>
<td>9300</td>
<td></td>
<td>8206</td>
</tr>
<tr>
<td>0.2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.44</td>
<td>400</td>
<td>9300</td>
<td></td>
<td>400081</td>
</tr>
<tr>
<td>0.2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>398</td>
<td>399</td>
<td>9300</td>
<td></td>
<td>399634</td>
</tr>
<tr>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>140</td>
<td>247</td>
<td>9300</td>
<td></td>
<td>247086</td>
</tr>
<tr>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>131</td>
<td>131</td>
<td>9300</td>
<td></td>
<td>103168</td>
</tr>
<tr>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>89</td>
<td>103</td>
<td>9300</td>
<td></td>
<td>96458</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>95</td>
<td>95</td>
<td>9300</td>
<td></td>
<td>10603</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>26</td>
<td>44</td>
<td>9300</td>
<td></td>
<td>44878</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>24</td>
<td>24</td>
<td>9300</td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>15</td>
<td>15</td>
<td>9300</td>
<td></td>
<td>15630</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>12</td>
<td>9300</td>
<td></td>
<td>12335</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>9300</td>
<td></td>
<td>2833</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>9300</td>
<td></td>
<td>491</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9300</td>
<td></td>
<td>3874</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.116</td>
<td>0.837</td>
<td>9300</td>
<td></td>
<td>1007</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.512</td>
<td>0.512</td>
<td>9300</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.121</td>
<td>0.353</td>
<td>9300</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.194</td>
<td>0.191</td>
<td>9300</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.103</td>
<td>0.103</td>
<td>9300</td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

---

The TAU Performance System

DOE ACTS Workshop, September 2002
**jRacy (NAS Parallel Benchmark – LU)**

- **n**: node
- **c**: context
- **t**: thread

**Global profiles**

- Routine profile across all nodes

**Individual profile**

- **MPL_Recv0**
- **MPL_Send0**
- **MPL_Type_commit()**
- **MPL_Type_contiguous()**
- **MPL_Type_struct()**
- **MPL_Wait0**
- **MPL_Wtime0**
- **applu**
- **bcast_inputs**
- **bits**
- **buts**
- **erhs**
- **error**

---

*The TAU Performance System*
TAU + PAPI (NAS Parallel Benchmark – LU)

- Floating point operations
- Replaces execution time
- Only requires re-linking to different TAU library
TAU + Vampir (NAS Parallel Benchmark – LU)

Timeline display

Callgraph display

Parallelism display

Communications display

The TAU Performance System

DOE ACTS Workshop, September 2002
TAU Performance System Status

- Computing platforms
  - IBM SP / Power4, SGI Origin 2K/3K, Intel Teraflop, Cray T3E / SV-1 (X-1 planned), Compaq SC, HP, Sun, Hitachi SR8000, NEX SX-5 (SX-6 underway), Intel (x86, IA-64) and Alpha Linux cluster, Apple, Windows

- Programming languages
  - C, C++, Fortran 77, F90, HPF, Java, OpenMP, Python

- Communication libraries
  - MPI, PVM, Nexus, Tulip, ACLMPL, MPIJava

- Thread libraries
  - pthreads, Java, Windows, Tulip, SMARTS, OpenMP
TAU Performance System Status (continued)

- **Compilers**
  - KAI, PGI, GNU, Fujitsu, Sun, Microsoft, SGI, Cray, IBM, Compaq

- **Application libraries**
  - Blitz++, A++/P++, ACLVIS, PAWS, SAMRAI, Overture

- **Application frameworks**
  - POOMA, POOMA-2, MC++, Conejo, Uintah, VTF, UPS

- **Projects**
  - Aurora / SCALEA: ACPC, University of Vienna

- **TAU full distribution (Version 2.1x, web download)**
  - Measurement library and profile analysis tools
  - Automatic software installation and examples
  - TAU User’s Guide
PDT Status

- Program Database Toolkit (Version 2.1, web download)
  - EDG C++ front end (Version 2.45.2)
  - Mutek Fortran 90 front end (Version 2.4.1)
  - C++ and Fortran 90 IL Analyzer
  - DUCTAPE library
  - Standard C++ system header files (KCC Version 4.0f)

- PDT-constructed tools
  - TAU instrumentor (C/C++/F90)
  - Program analysis support for SILOON and CHASM

- Platforms
  - SGI, IBM, Compaq, SUN, HP, Linux (IA32/IA64), Apple, Windows, Cray T3E, Hitachi
**Semantic Performance Mapping**

- Associate performance measurements with high-level semantic abstractions
- Need mapping support in the performance measurement system to assign data correctly
Semantic Entities/Attributes/Associations (SEAA)

- New dynamic mapping scheme (S. Shende, Ph.D. thesis)
  - Contrast with ParaMap (Miller and Irvin)
  - Entities defined at any level of abstraction
  - Attribute entity with semantic information
  - Entity-to-entity associations
- Two association types (implemented in TAU API)
  - Embedded – extends associated object to store performance measurement entity
  - External – creates an external look-up table using address of object as key to locate performance measurement entity
Hypothetical Mapping Example

Particles distributed on surfaces of a cube

```c
Particle* P[MAX]; /* Array of particles */

int GenerateParticles() {
    /* distribute particles over all faces of the cube */
    for (int face=0, last=0; face < 6; face++) {
        /* particles on this face */
        int particles_on_this_face = num(face);
        for (int i=last; i < particles_on_this_face; i++) {
            /* particle properties are a function of face */
            P[i] = ... f(face);
            ...
        }
        last+= particles_on_this_face;
    }
}
```
Hypothetical Mapping Example (continued)

```c
int ProcessParticle(Particle *p) {
    /* perform some computation on p */
}
int main() {
    GenerateParticles();
    /* create a list of particles */
    for (int i = 0; i < N; i++)
        /* iterates over the list */
        ProcessParticle(P[i]);
}
```

FAQ:
- How much time is spent processing face \( i \) particles?
- What is the distribution of performance among faces?
No Performance Mapping versus Mapping

מנון התוכן:

- מיון הביצועיםument מדווח על הביצועים במבחן תוכני
- לא 제공 תמיכה עבור מיפוי

כוננות במעון

- כלים צעד צעד במעון
- ניתן לצפות במעון עם תמיכה בתוכני הצוות של תורת הניסיון

TAU (לא מתוחכם)

TAU (עם מתוחכם)

The TAU Performance System

DOE ACTS Workshop, September 2002


Strategies for Empirical Performance Evaluation

- Empirical performance evaluation as a series of performance experiments
  - Experiment trials describing instrumentation and measurement requirements
  - Where/When/How axes of empirical performance space
    - where are performance measurements made in program
    - when is performance instrumentation done
    - how are performance measurement/instrumentation chosen

- Strategies for achieving flexibility and portability goals
  - Limited performance methods restrict evaluation scope
  - Non-portable methods force use of different techniques
  - Integration and combination of strategies
**PETSc (ANL)**

- Portable, Extensible Toolkit for Scientific Computation
- Scalable (parallel) PDE framework
  - Suite of data structures and routines
  - Solution of scientific applications modeled by PDEs
- Parallel implementation
  - MPI used for inter-process communication
- TAU instrumentation
  - PDT for C/C++ source instrumentation
  - MPI wrapper library layer instrumentation
- Example
  - Solves a set of linear equations (Ax=b) in parallel (SLES)
# PETSc Linear Equation Solver Profile

![PETSc Performance Profile](image.png)

<table>
<thead>
<tr>
<th>%time</th>
<th>msec</th>
<th>total msec</th>
<th>#call</th>
<th>#subs</th>
<th>usec/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.3</td>
<td>7.794</td>
<td>7.794</td>
<td>407</td>
<td>0</td>
<td>19152</td>
<td>VecMAXPY_Seq(int, const PetscScalar *, Vec, V)</td>
</tr>
<tr>
<td>20.6</td>
<td>7.534</td>
<td>7.534</td>
<td>393</td>
<td>0</td>
<td>19172</td>
<td>VecMDot_Seq(int, Vec, const Vec *, PetscScalar)</td>
</tr>
<tr>
<td>18.8</td>
<td>6.886</td>
<td>6.908</td>
<td>407</td>
<td>1528</td>
<td>16973</td>
<td>MatSolve_SeqAIJ_NaturalOrdering(Mat, Vec, Vec)</td>
</tr>
<tr>
<td>12.5</td>
<td>4.548</td>
<td>4.599</td>
<td>407</td>
<td>1528</td>
<td>11302</td>
<td>MatMult_SeqAIJ(Mat, Vec, Vec) C</td>
</tr>
<tr>
<td>7.9</td>
<td>2.877</td>
<td>2.877</td>
<td>1353.75</td>
<td>0</td>
<td>2126</td>
<td>MPI_Recv()</td>
</tr>
<tr>
<td>4.9</td>
<td>1.282</td>
<td>1.801</td>
<td>49800</td>
<td>49800</td>
<td>36 int MatSetValues(Mat, int, int *, int, int *, PetscScalar)</td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>785</td>
<td>36,651</td>
<td>1</td>
<td>49832</td>
<td>36651451 int main(int, char **) C</td>
<td></td>
</tr>
<tr>
<td>1.7</td>
<td>618</td>
<td>627</td>
<td>407</td>
<td>1528</td>
<td>1543</td>
<td>MatMultAdd_SeqAIJ_Inode(Mat, Vec, Vec, Vec) C</td>
</tr>
<tr>
<td>1.4</td>
<td>519</td>
<td>519</td>
<td>49800</td>
<td>0</td>
<td>10 int MatSetValues_MPIAIJ(Mat, int, int *, int, int *, PetscScalar)</td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td>337</td>
<td>337</td>
<td>1</td>
<td>35</td>
<td>337700</td>
<td>MPI_Init()</td>
</tr>
<tr>
<td>1.1</td>
<td>328</td>
<td>394</td>
<td>3142</td>
<td>15205</td>
<td>126 int PetscOptionsFindPair_Private(const char *, const char *, ...)</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>233</td>
<td>240</td>
<td>182</td>
<td>649</td>
<td>1320</td>
<td>PetscFListGetPathAndFunction(const char *, char *, ...)</td>
</tr>
<tr>
<td>1.3</td>
<td>219</td>
<td>463</td>
<td>153</td>
<td>1110</td>
<td>3032</td>
<td>PetscFListAdd(PetscFList *, const char *, const char *, ...)</td>
</tr>
<tr>
<td>0.6</td>
<td>215</td>
<td>215</td>
<td>1526.25</td>
<td>0</td>
<td>141 int PetscStackCopy(PetscStack *, PetscStack *) C</td>
<td></td>
</tr>
</tbody>
</table>

---

The TAU Performance System

DOE ACTS Workshop, September 2002
PETSc Linear Equation Solver Profile
PETSc Trace Summary Profile
PETSc Performance Trace
Trace visualization

- TAU will generate event-traces with PAPI performance data. Vampir (v3.0) will support visualization of this data

Runtime performance monitoring and analysis

- Online performance data access
  - incremental profile sampling
- Performance analysis and visualization in SCIRun

Performance Database Framework

- XML parallel profile representation
  - TAU profile translation
- PostgresSQL performance database

Statement-level automatic performance instrumentation
Concluding Remarks

- Complex software and parallel computing systems pose challenging performance analysis problems that require robust methodologies and tools.
- To build more sophisticated performance tools, existing proven performance technology must be utilized.
- Performance tools must be integrated with software and systems models and technology.
  - Performance engineered software
  - Function consistently and coherently in software and system environments.
- PAPI and TAU performance systems offer robust performance technology that can be broadly integrated.
Acknowledgements

- Department of Energy (DOE)
  - MICS office
    - DOE 2000 ACTS contract
    - “Performance Technology for Tera-class Parallel Computer Systems: Evolution of the TAU Performance System”
  - University of Utah DOE ASCI Level 1 sub-contract
  - DOE ASCI Level 3 (LANL, LLNL)
- DARPA
- NSF National Young Investigator (NYI) award
- Research Centre Juelich
  - John von Neumann Institute for Computing
  - Dr. Bernd Mohr
- Los Alamos National Laboratory
Information

- TAU (http://www.acl.lanl.gov/tau)
- PDT (http://www.acl.lanl.gov/pdtoolkit)
- PAPI (http://icl.cs.utk.edu/projects/papi/)
- OPARI (http://www.fz-juelich.de/zam/kojak/)