

Performance Profiling of Parallel Codes

Abhinav Sarje

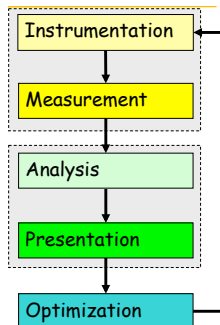
asarje@lbl.gov

Performance and Algorithms Group
CRD, LBNL

Performance Science and Engineering

A sophisticated and well developed field.

- ① Performance profiling and benchmarking.
- ② Performance modeling.
- ③ Performance tuning.



Outline and Goals

- Basics of performance evaluation.
- Tools for performance evaluation of parallel codes.
- Working with TAU.
 - Basic options for generating profiles.
 - Analyzing the performance profiles.
 - Advanced options and integration with PAPI.
 - Tracing.
- Performance evaluation on accelerators.

Building a Parallel Profiling Tool

- Language independence.
- Avoid excessive instrumentation.
- Binary analysis capability.
- Collection of multiple metrics.
- Hierarchical analysis.
- Hierarchical aggregation.
- Scalable.

Various Profiling Tools

- **gprof**
- HPCToolkit
- TAU
- PAPI
- Nvidia Visual Profiler
- Intel Vtune

TAU: Tuning and Analysis Utilities

- <http://tau.uoregon.edu>
- Developed at University of Oregon.
- Open Source.
- Comprehensive performance profiling and tracing.
- Integrated tool with instrumentation, measurement and visualization.
- Supports most HPC systems.
- Simple to integrate into applications.

DISCLAIMER: Some figures have been taken from Sameer Shende's slides.

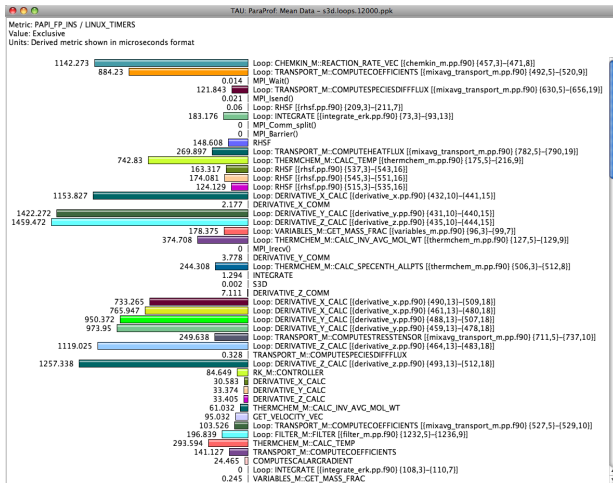
TAU: Using it right away ...

```
$ mpirun -np 4 tauex ./app
```

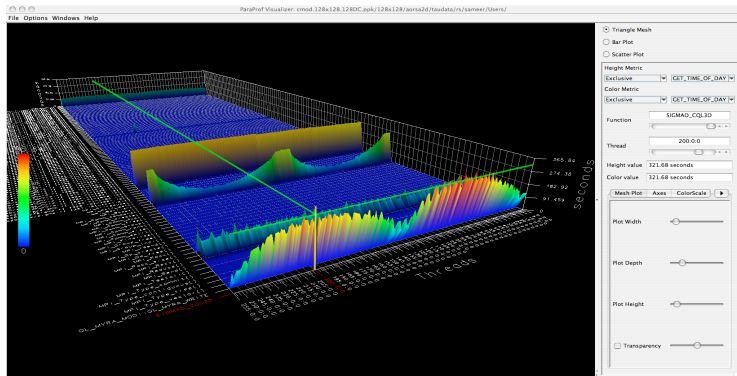
- This generates a basic profile of the application.
- Easy to identify bottleneck routines.

TAU: Visualizing Performance Data

- Paraprof (packaged with TAU.)



TAU: Visualizing Performance Data



TAU: System Workings

① Instrumentation

- Source code instrumentation
- External library wrapping
- Re-generate application binary

② Measurement

- Direct instrumentation: Interval events
- Indirect instrumentation: Sampling

③ Analysis

- Visualization and analysis in paraprof and perexplorer
- Visualization in external tools, like Vampir and Jumpshot

TAU: Instrumentation Options

- Source code instrumentation
 - Automatic instrumentation with static source code analysis.
 - Manual instrumentation.
- Library level instrumentation
 - Wrapping external libraries.
- Binary code instrumentation
 - Runtime instrumentation.

TAU: Instrumentation Options

- Instrumentation levels:
 - Source code.
 - Object code.
 - Library code.
 - Executable code.
 - Runtime system.
 - Operating system.
- Different levels provide different information.

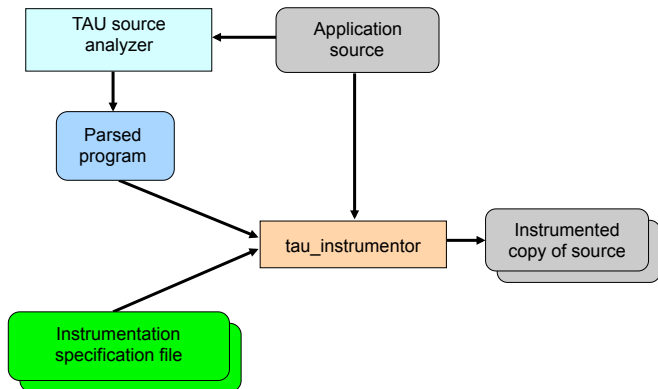
TAU: Instrumentation Techniques

- Static: instrument code prior to execution.
- Dynamic: instrument executable code at runtime.
- Manual and automatic.

TAU: Events

- **Interval events.** Have begin and end events.
E.g. exclusive and inclusive duration
- **Atomic events.** Capture state with data values.
E.g. cache miss reading from hardware counter

TAU: Automatic Source Instrumentation



- Compile time options with **TAU_OPTIONS**.
- Runtime environment variables, e.g. **TAU_TRACE**, **TAU_METRICS**.

TAU: Using

- Source code instrumentation:

```
$ export TAU_MAKEFILE=/path/to/tau/lib/Makefile.tau-mpi-pdt
$ export TAU_OPTIONS='-optVerbose'
$ taucxx app.cpp
$ mpirun -np 4 ./app
```

- Analyze generated performance data:

```
$ paraprof          or,
$ pprof
```


TAU: Makefiles

```
$ ls -l /path/to/tau/lib/Makefile.*  
Makefile.tau-mpi-pdt  
Makefile.tau-mpi-pdt-openmp  
Makefile.tau-mpi-pdt-pthread  
Makefile.tau-papi-mpi-pdt  
...
```

TAU: Selective Instrumentation

```
$ cat select.tau
BEGIN_INSTRUMENT_SECTION
loops routine="#"
END_INSTRUMENT_SECTION
BEGIN_EXCLUDE_LIST
foo
bar
END_EXCLUDE_LIST
$ export TAU_OPTIONS='-optTauSelectFile=select.tau
-optVerbose'
```

TAU: Hardware Counters with PAPI

- `http://icl.cs.utk.edu/papi`
- Developed at University of Tennessee, Knoxville.
- Provides interface to hardware counters found in most processors.
- Information such as cache behaviors, branching, memory patterns, stalls, floating point efficiency, number of instructions, etc.

PAPI: Available Counters

```
$ papi_avail
```

```
...
```

Name	Code	Avail	Deriv	Description (Note)
PAPI_L1_DCM	0x80000000	No	No	Level 1 data cache misses
PAPI_L1_ICM	0x80000001	Yes	No	Level 1 instruction cache misses
PAPI_L2_DCM	0x80000002	Yes	Yes	Level 2 data cache misses

```
...
```

PAPI: Choosing Counters

```
$ papi_event_chooser PAPI_FP_OPS
```

```
Event Chooser: Available events which can be added with given events.
```

```
...
```

Name	Code	Deriv	Description (Note)
PAPI_L1_ICM	0x80000001	No	Level 1 instruction cache misses
PAPI_L2_DCM	0x80000002	Yes	Level 2 data cache misses
PAPI_L2_ICM	0x80000003	No	Level 2 instruction cache misses

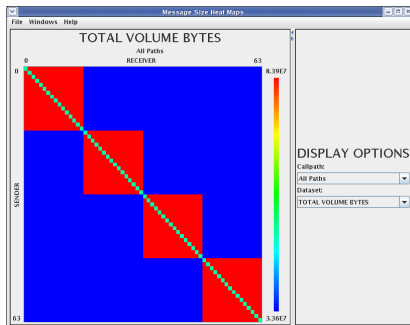
```
...
```

TAU: Profiling with PAPI

```
$ export TAU_MAKEFILE=/path/to/tau/lib/Makefile.tau-papi-mpi-pdt
$ export TAU_METRICS=TIME:PAPI_FP_INS:PAPI_L1_DCM
$ mpiexec -np 4 ./app
$ paraprof
```

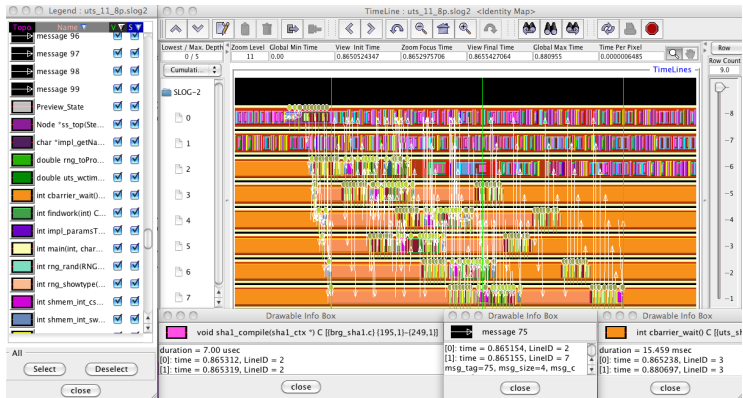
TAU: Profiling MPI Communication

```
$ export TAU_COMM_MATRIX=1  
$ mpiexec -np 4 ./app  
$ paraprof
```



TAU: Tracing

```
$ export TAU_TRACE=1
$ mpiexec -np 4 ./app
$ tau_treemerge.pl
$ tau2slog2 tau.trc tau.edf -o app.slog2
$ jumpshot
```



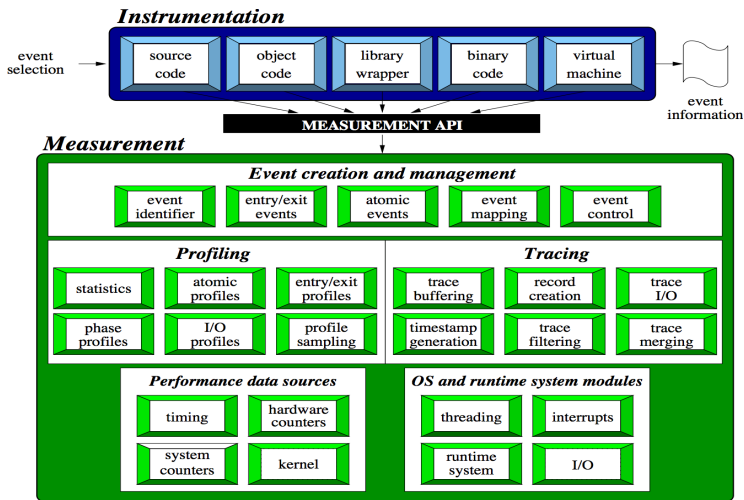
TAU: Manual Instrumentation

```
#include <tau.h>

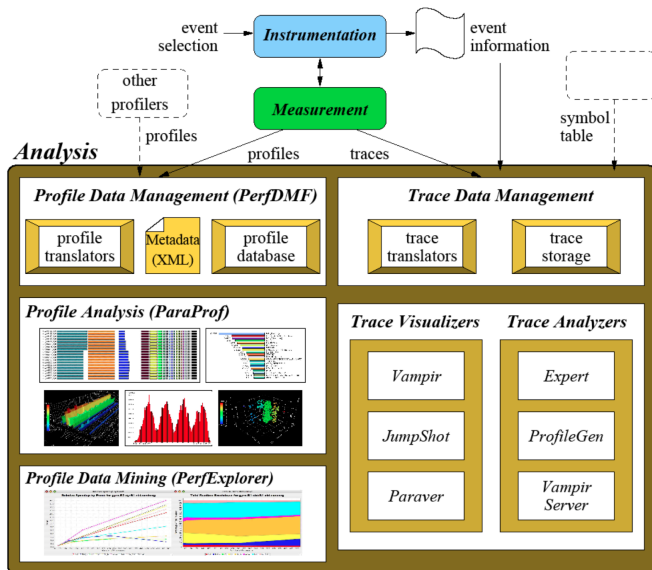
int foo(int x) {
    TAU_START("foo");
    /* do something */
    TAU_STOP("foo");
}

int main(int argc, char **argv) {
    TAU_INIT(&argc, &argv);
    TAU_START("main");
    TAU_PROFILE_SET_NODE(rank);
    ...
    TAU_STOP("main");
}
```

TAU: Architecture

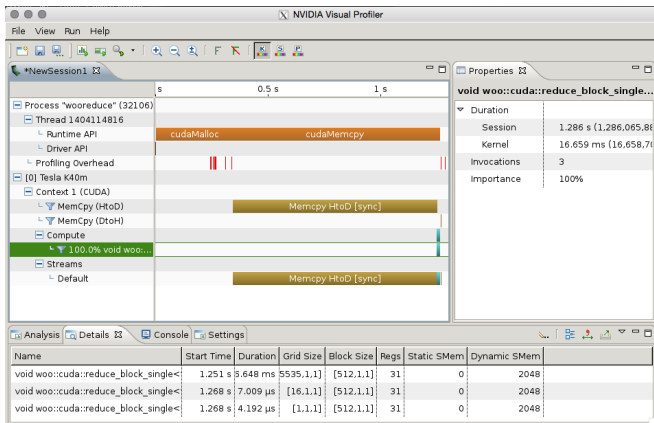


TAU: Architecture



Profiling CUDA Codes

- TAU supports CUDA profiling.
- Other tools include Nvidia's **nvprof** and **nvvp**.



Profiling with Intel Xeon Phi

- Intel VTune.

The screenshot shows the Intel VTune Amplifier XE 2013 interface. The main window is titled 'Choose Analysis Type' and displays a tree view of analysis types on the left. The selected analysis type is 'General Exploration - Knights Corner Platform'. The right pane shows configuration options for this analysis type, including checkboxes for 'Analyze general cache usage', 'Analyze vectorization usage', 'Analyze TLB misses', 'Analyze additional L2 cache events', and 'Analyze user tasks'. A text field for 'List of Intel Xeon Phi coprocessor cards' is set to '0'. Below these options is a 'Details' section with a table of events configured for the CPU: Intel(R) Xeon(R) E5 processor.

Event Name	Sample After	Event Description
CPU_CLK_UNHALTED	10000000	
DATA_READ_MISS_OR_WRITE_MISS	1000000	
DATA_READ_OR_WRITE	1000000	
EXEC_STAGE_CYCLES	10000000	
INSTRUCTIONS_EXECUTED	10000000	
L1_DATA_HIT_INFLIGHT_PF1	50000	
L2_DATA_READ_MISS_CACHE_FILL	100000	

Additional configuration options include 'Analyze memory bandwidth' (unchecked), 'Event mode' (set to 'All'), and 'Analyze user tasks' (unchecked). The interface also features 'Start' and 'Start Paused' buttons, a 'Project Properties' link, and a 'Command Line...' field at the bottom right.

Further ...

- **“Performance Tuning of Scientific Applications”**, D.H. Bailey, R.F. Lucas, S.W. Williams, CRC Press 2011.
- **<http://www.cs.uoregon.edu/research/tau/docs.php>**
- **<http://icl.cs.utk.edu/papi>**
- **“Nvidia CUDA Compute Visual Profiler Guide”**, version 7.0, 2015.