Software Profiling with TAU

Prasad Maddumage
mhemantha@fsu.edu

Research Computing Center
Florida State University
Software Profiling

• Dynamic program analysis using various measures related to code execution
  – CPU/memory utilization, frequency of function calls, I/O, MPI library usage, hardware counters, etc.

• Profilers *instrument* source or binary to obtain such measures during runtime
  – Instrumenting is inserting probes and replacing or wrapping function calls (eg: MPI calls, I/O) with modified calls of a source code

• Analyzing the results will help programmers/scientists to improve code performance
Profile vs Trace

- **Profile**: statistical summary of all metrics measured
  - Shows how much total time/resources each call utilized

- **Trace**: timeline of runtime events took place
  - Shows when each event happened and where
Why use TAU?

- Tuning and Analysis Utilities (20+ year project)
  - Actively developed by Univ. of Oregon, ANL, LANL, Julich

- Comprehensive performance profiling and tracing
  - Integrated, scalable, flexible, portable
  - Targets all parallel programming/execution paradigms

- Integrated performance toolkit
  - Instrumentation, measurement, analysis, visualization
  - Performance data management and data mining
  - Open source

- Easy to integrate in application frameworks

- Well documented
How does TAU work?

**Instrumentation**
- source code
- object code
- library wrapper
- binary code
- virtual machine

**Measurement**

**Event creation and management**
- event identifier
- entry/exit events
- atomic events
- event mapping
- event control

**Profiling**
- statistics
- atomic profiles
- entry/exit profiles
- profile I/O
- sampling profiles
- mapping (callpath)

**Tracing**
- trace buffering
- record creation
- trace I/O
- timestamp generation
- trace filtering

**Performance data sources**
- timing
- hardware counters
- system counters
- kernel

**OS and runtime system modules**
- threading
- interrupts
- runtime system
- I/O
How does TAU work?

- **Instrumentation**
  - Source code instrumentation using pre-processors and compiler scripts
  - Wrapping external libraries (I/O, MPI, Memory, CUDA, OpenCL, pthread)
  - Rewriting the binary executable

- **Measurement**
  - Direct: interval events, Indirect: collect samples to profile statement execution
  - Per-thread storage of performance data
  - Throttling and runtime control of low-level events
How does TAU work?

• Analysis
  – TAU creates one profile file per node in a single location
  – Profile file names look like,
    ```
    profile.0.0.0, profile.1.0.0, ...
    ```
  – 2D and 3D visualization of profile data using pprof and paraprof
  – Trace conversion & display in external visualizers such as Jumpshot
TAU Event Types

- **Interval**: start-stop events (eg: function call)
- **Atomic**: trigger at a single point with data (eg: memory allocation)
  - Measures total, samples, min/max/mean/std. deviation statistics
- **Context**: atomic events with executing context
  - Measures total, samples, min/max/mean/std. deviation statistics
### TAU event types

#### profile.0.0.0

<table>
<thead>
<tr>
<th>%Time Exclusive</th>
<th>Inclusive</th>
<th>#Call</th>
<th>#Subrs</th>
<th>Inclusive</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>msec</td>
<td>total msec</td>
<td></td>
<td></td>
<td>usec/call</td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>1:18.355</td>
<td>1:18.561</td>
<td>1</td>
<td>1818</td>
<td>.TAU application</td>
</tr>
<tr>
<td>0.3</td>
<td>202</td>
<td>202</td>
<td>1814</td>
<td>0</td>
<td>read()</td>
</tr>
<tr>
<td>0.0</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>open()</td>
</tr>
<tr>
<td>0.0</td>
<td>0.004</td>
<td>0.004</td>
<td>2</td>
<td>0</td>
<td>lseek()</td>
</tr>
</tbody>
</table>

#### USER EVENTS: profile.0.0.0

<table>
<thead>
<tr>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1812</td>
<td>8192</td>
<td>2174</td>
<td>8186</td>
<td>179.6</td>
<td>Bytes Read</td>
</tr>
<tr>
<td>1812</td>
<td>8192</td>
<td>2174</td>
<td>8186</td>
<td>179.6</td>
<td>Bytes Read : .TAU application =&gt; read()</td>
</tr>
<tr>
<td>906</td>
<td>8192</td>
<td>2174</td>
<td>8185</td>
<td>199.8</td>
<td>Bytes Read &lt;file=data1.dat&gt; : .TAU application =&gt; read()</td>
</tr>
<tr>
<td>906</td>
<td>8192</td>
<td>3467</td>
<td>8187</td>
<td>156.9</td>
<td>Bytes Read &lt;file=data2.dat&gt; : .TAU application =&gt; read()</td>
</tr>
<tr>
<td>1812</td>
<td>1170</td>
<td>0.113</td>
<td>913.9</td>
<td>124.7</td>
<td>Read Bandwidth (MB/s)</td>
</tr>
</tbody>
</table>

- **Interval events**
- **Atomic event**
- **Context events**
Exclusive vs Inclusive time

```c
int foo()
{
    int a;
    a = a + 1;

    bar();

    a = a + 1;
    return a;
}
```

exclusive duration

inclusive duration
TAU at RCC

- Currently available on HPC for all six compilers
  - GNU, Intel, PGI - OpenMPI and MVAPICH2
- To use serial version
  - `module load tau-serial`
- To use parallel version
  - `module load tau`
- Documentation: https://rcc.fsu.edu/software/tau
TAU Instrumentation

- Library interposition (dynamic instrumentation)
  - No need to recompile your code
  - `mpirun -np 4 tau_exec <options> <your binary>`
  - Can profile MPI (default), memory use, I/O, ...
  - Cannot track user functions

```
$ gfortran -o gauss gauss.f90
$ module load tau-serial
$ tau_exec -T serial -io ./gauss
$ pprof -s
Reading Profile files in profile.0.0.0.*
FUNCTION SUMMARY (total):
```

<table>
<thead>
<tr>
<th>%Time Exclusive</th>
<th>Msec</th>
<th>Inclusive</th>
<th>#Call</th>
<th>#Subrs</th>
<th>Inclusive usec/call</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>1:16.607</td>
<td>1:16.627</td>
<td>1</td>
<td>1818</td>
<td>76627147</td>
</tr>
<tr>
<td>0.0</td>
<td>19</td>
<td>19</td>
<td>1814</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>0.0</td>
<td>0.026</td>
<td>0.026</td>
<td>2</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>0.0</td>
<td>0.001</td>
<td>0.001</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Still buggy!
TAU Instrumentation

- **Scripted Compilation**
  - Use `tau_f90.sh`, `tau_cc.sh`, and `tau_cxx.sh` to instrument and compile Fortran, C, and C++ programs
  - **Compiler Based Instrumentation**
    - Use the compiler itself for instrumenting
    - Provides more detailed profiles than dynamic approach
    - Cannot profile user functions
    - Needs recompilation of the code

```
$ tau_cc.sh -tau_options=--optCompInst samplecprogram.c
```
TAU Instrumentation

- Source based instrumentation
  - Uses PDT (Program Database Toolkit) to fully instrument the source code
  - Able to generate complete profiles by measuring low level events (loops, hardware counters, etc.)
  - Needs recompilation of the code (Simply switch `CC` or `FC` with `tau_cc.sh` or `tau_f90.sh`)

```bash
$ module load gnu-openmpi
$ module load tau
$ tau_f90.sh -o mat_mul_par mat_mul_par.f90
$ msub mat_mul_par.sh
$ pprof
```
pprof

Reading Profile files in profile.*

NODE 0; CONTEXT 0; THREAD 0:

<table>
<thead>
<tr>
<th>%Time</th>
<th>Exclusive</th>
<th>Inclusive</th>
<th>#Call</th>
<th>#Subrs</th>
<th>Inclusive</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>0.139</td>
<td>55,725</td>
<td>1</td>
<td>1</td>
<td>55725516</td>
<td>.TAU application</td>
</tr>
<tr>
<td>100.0</td>
<td>26,947</td>
<td>55,725</td>
<td>1</td>
<td>7</td>
<td>55725377</td>
<td>MAT_MUL_PAR</td>
</tr>
<tr>
<td>49.5</td>
<td>27,590</td>
<td>27,590</td>
<td>1</td>
<td>0</td>
<td>27590687</td>
<td>MPI_Gather()</td>
</tr>
<tr>
<td>1.0</td>
<td>541</td>
<td>541</td>
<td>1</td>
<td>0</td>
<td>541913</td>
<td>MPI_Init()</td>
</tr>
<tr>
<td>0.9</td>
<td>488</td>
<td>488</td>
<td>1</td>
<td>0</td>
<td>488823</td>
<td>MPI_Bcast()</td>
</tr>
<tr>
<td>0.2</td>
<td>94</td>
<td>94</td>
<td>1</td>
<td>0</td>
<td>94155</td>
<td>MPI_Scatter()</td>
</tr>
<tr>
<td>0.1</td>
<td>62</td>
<td>62</td>
<td>1</td>
<td>0</td>
<td>62281</td>
<td>MPI_Finalize()</td>
</tr>
<tr>
<td>0.0</td>
<td>0.001</td>
<td>0.001</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1 MPI_Comm_size()</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0 MPI_Comm_rank()</td>
</tr>
</tbody>
</table>

USER EVENTS Profile : NODE 0, CONTEXT 0, THREAD 0

<table>
<thead>
<tr>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.4E+07</td>
<td>6.4E+07</td>
<td>6.4E+07</td>
<td>0</td>
<td>Message size for broadcast</td>
</tr>
<tr>
<td>1</td>
<td>4E+06</td>
<td>4E+06</td>
<td>4E+06</td>
<td>0</td>
<td>Message size for gather</td>
</tr>
<tr>
<td>1</td>
<td>4E+06</td>
<td>4E+06</td>
<td>4E+06</td>
<td>0</td>
<td>Message size for scatter</td>
</tr>
</tbody>
</table>
paraprof

Metric: TIME
Value: Exclusive
Units: seconds

How much time spent on each operation

Poor load balancing!

Only MPI calls are profiled
Source based Instrumentation

• There is more...
  
  – The TAU module picks a “Makefile” for you, depending on the compiler you are using
    
    • It is stored in the variable `TAU_MAKEFILE`
    
    • Eg: Default for gnu-openmpi is `Makefile.tau-papi-mpi-pdt`
    
    • Makefiles can be changed by user depending on the purpose

    
    ```
    Makefile.tau-communicators-papi-mpi-pdt
    Makefile.tau-headroom-papi-mpi-pdt
    Makefile.tau-memory-papi-mpi-pdt
    Makefile.tau-papi-mpi-pdt
    Makefile.tau-papi-mpi-pdt-trace
    Makefile.tau-phase-papi-mpi-pdt
    ```
### Using Different Makefiles

<table>
<thead>
<tr>
<th>%Time</th>
<th>Exclusive</th>
<th>Inclusive</th>
<th>#Call</th>
<th>#Subrs</th>
<th>Inclusive Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>msec</td>
<td>total msec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>0.012</td>
<td>1:06.231</td>
<td>1</td>
<td>1</td>
<td>66231389 .TAU application</td>
</tr>
<tr>
<td>100.0</td>
<td>2,021</td>
<td>1:06.231</td>
<td>1</td>
<td>1</td>
<td>66231377 GAUSS</td>
</tr>
<tr>
<td>96.9</td>
<td>19,828</td>
<td>1:04.209</td>
<td>1</td>
<td>6000</td>
<td>64209853 GAUSSJ</td>
</tr>
<tr>
<td>45.8</td>
<td>30,348</td>
<td>30,348</td>
<td>4000</td>
<td>0</td>
<td>7587 OUTERPROD</td>
</tr>
<tr>
<td>21.2</td>
<td>14,023</td>
<td>14,023</td>
<td>1000</td>
<td>0</td>
<td>14023 OUTERAND</td>
</tr>
<tr>
<td>0.0</td>
<td>9</td>
<td>9</td>
<td>1000</td>
<td>0</td>
<td>10 SWAP</td>
</tr>
</tbody>
</table>

**USER EVENTS: profile.-1.0.0**

<table>
<thead>
<tr>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.TAU application - Heap Memory Used (KB)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GAUSS - Heap Memory Used (KB)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GAUSSJ - Heap Memory Used (KB)</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>OUTERAND - Heap Memory Used (KB)</td>
</tr>
<tr>
<td>4000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>OUTERPROD - Heap Memory Used (KB)</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SWAP - Heap Memory Used (KB)</td>
</tr>
</tbody>
</table>
Selective Instrumentation

- Not all functions need to be profiled in large applications

```
export TAU_OPTIONS="-optTauSelectFile=select.tau"

cat select.tau
BEGIN_INSTRUMENT_SECTION
loops file="mat_mul_par.f90" routine="#"
END_INSTRUMENT_SECTION
```

Only need to profile outer loops of the given file
Selective Instrumentation

Metric: TIME
Value: Exclusive
Units: seconds

- 23.574
- 20.214
- 9.26

Loop: MAT_MUL_PAR [{mat_mul_par.f90} {71,3}--{77,8}]
  - MPI_Finalize()
  - MPI_Gather()

Loop: MAT_MUL_PAR [{mat_mul_par.f90} {88,8}--{90,13}]
  - 1.32
  - MPI_Bcast()
  - 0.532
  - MPI_Init()
  - 0.447
  - MPI_Scatter()
  - 0.037

Loop: MAT_MUL_PAR [{mat_mul_par.f90} {10,1}--{96,23}]
  - 0.02

Loop: MAT_MUL_PAR [{mat_mul_par.f90} {33,8}--{37,13}]
  - 0.002

Loop: MAT_MUL_PAR [{mat_mul_par.f90} {39,8}--{43,13}]
  - 0.002

8.3E-5 | TAU application
9.4E-7 | MPI_Comm_size()
3.1E-7 | MPI_Comm_rank()
Selective Instrumentation

BEGIN_EXCLUDE_LIST
void quicksort(int *, int, int)
# The next line excludes all functions beginning with "sort_"
# and having arguments "int *
void sort_(int *)
void interchange(int *, int *)
END_EXCLUDE_LIST

#Exclude these files from profiling
BEGIN_FILE_EXCLUDE_LIST
*.so
END_FILE_EXCLUDE_LIST

BEGIN_INSTRUMENT_SECTION
# instrument all the outer loops in this routine
loops file="loop_test.cpp" routine="multiply"
# tracks memory allocations/deallocations as well as
# potential leaks
memory file="foo.f90" routine="INIT"
# tracks the size of read, write and print statements in
# this routine
io file="foo.f90" routine="RINB"
Using Optional TAU Compiler Options

- By setting \texttt{TAU\_OPTIONS} variable or directly using TAU compiler options while compiling will change its behavior
  - \texttt{-optTrackIO} will profile I/O operations
  - \texttt{-optHeaderInst} will enable instrumentation of headers
  - For a full list, use \texttt{tau_compiler.sh -help} command
# Runtime Environment Variables

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU_PROFILE</td>
<td>1</td>
<td>Set 0 to stop profiling (eg: for tracing)</td>
</tr>
<tr>
<td>PROFILEDIR</td>
<td>.</td>
<td>Location for profile files</td>
</tr>
<tr>
<td>TAU_TRACE</td>
<td>0</td>
<td>Set 1 for tracing</td>
</tr>
<tr>
<td>TAU_TRACK_HEAP</td>
<td>0</td>
<td>Set 1 to track heap memory or headroom available</td>
</tr>
<tr>
<td>TAU_TRACK_HEADROOM</td>
<td>0</td>
<td>Set 1 to track heap memory or headroom available</td>
</tr>
<tr>
<td>TAU_CALLPATH</td>
<td>0</td>
<td>Set 1 to start callpath profiling</td>
</tr>
<tr>
<td>TAU_COMM_MATRIX</td>
<td>0</td>
<td>Set 1 to generate communication matrix data</td>
</tr>
<tr>
<td>TAU_COMPENSATE</td>
<td>0</td>
<td>Set 1 to compensate instrumentation overhead</td>
</tr>
<tr>
<td>TAU_THROTTLE</td>
<td>1</td>
<td>Skip instrumenting functions called frequently</td>
</tr>
<tr>
<td>TRACEDIR</td>
<td>.</td>
<td>Location for tracing data</td>
</tr>
</tbody>
</table>
Real World Examples
Call Path Graph
Communication Matrix

TOTAL VOLUME BYTES

All Paths

Receivers

Senders
How and what each node is doing?
Flat profile of a real world case

MPI_Send and MPI_Wait seems to be the culprit for slowdown
But why?
How do we find?
Hardware Counters

- TAU allows integration with other tools such as PAPI (Performance API)
- PAPI is installed on every HPC node and can be used to instrument a code using hardware counters as the metric
- `papi_avail` command will give you a complete list of available hardware counters on a specific node

```bash
export COUNTER1=GET_TIME_OF_DAY #To measure runtime
export COUNTER2=PAPI_L1_DCM #To find level 1 cache miss
export COUNTER3=PAPI_L2_DCM #To find level 2 cache miss
export COUNTER4=PAPI_FLOPS #To measure FLOPS
```
## Hardware Counters

### AMD

<table>
<thead>
<tr>
<th>Timer</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6026E11</td>
<td>2.0737E11</td>
</tr>
<tr>
<td>6.8323E10</td>
<td>6.4667E10</td>
</tr>
<tr>
<td>2.9891E10</td>
<td>2.7642E10</td>
</tr>
<tr>
<td>2.3215E10</td>
<td>1.433E10</td>
</tr>
<tr>
<td>1.1921E10</td>
<td>1.0035E10</td>
</tr>
<tr>
<td>8.7521E9</td>
<td>4.8844E10</td>
</tr>
<tr>
<td>7.8236E9</td>
<td>4.6126E10</td>
</tr>
<tr>
<td>6.9363E9</td>
<td>2.8923E10</td>
</tr>
<tr>
<td>6.8124E9</td>
<td>2.8923E10</td>
</tr>
<tr>
<td>5.9189E9</td>
<td>2.294E10</td>
</tr>
<tr>
<td>5.7334E9</td>
<td>1.9494E10</td>
</tr>
<tr>
<td>5.5607E9</td>
<td>1.6507E10</td>
</tr>
<tr>
<td>5.2355E9</td>
<td>1.2631E10</td>
</tr>
<tr>
<td>4.7707E9</td>
<td>1.2489E10</td>
</tr>
<tr>
<td>4.1123E9</td>
<td>1.2062E10</td>
</tr>
<tr>
<td>3.8826E9</td>
<td>1.0934E10</td>
</tr>
</tbody>
</table>

### Intel

<table>
<thead>
<tr>
<th>Timer</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.6378E9</td>
<td>9.6378E9</td>
</tr>
<tr>
<td>8.6027E9</td>
<td>8.6027E9</td>
</tr>
<tr>
<td>8.2127E9</td>
<td>8.2127E9</td>
</tr>
<tr>
<td>8.0043E9</td>
<td>8.0043E9</td>
</tr>
<tr>
<td>7.8201E9</td>
<td>7.4747E9</td>
</tr>
<tr>
<td>7.4747E9</td>
<td>7.4747E9</td>
</tr>
<tr>
<td>6.8212E9</td>
<td>6.8212E9</td>
</tr>
<tr>
<td>6.3323E9</td>
<td>6.3323E9</td>
</tr>
</tbody>
</table>

### Metrics

- **PAPI_L1_DCM**
  - **Value:** Exclusive
  - **Units:** counts
### Measuring FLOPS

**MULTI_PAPI_FLOPS/profile.1.0.0**

<table>
<thead>
<tr>
<th>%Time</th>
<th>Exclusive</th>
<th>Inclusive</th>
<th>#Call</th>
<th>#Subrs</th>
<th>Count/Call Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>7.366E+06</td>
<td>5.984E+07</td>
<td>1</td>
<td>2</td>
<td>59837472 MATMUL_CACHE</td>
</tr>
<tr>
<td>43.8</td>
<td>7739</td>
<td>2.624E+07</td>
<td>1</td>
<td>1</td>
<td>26235629 CACHE_MISS</td>
</tr>
<tr>
<td>43.8</td>
<td>8066</td>
<td>2.624E+07</td>
<td>1</td>
<td>1</td>
<td>26235546 CACHE_NO_MISS</td>
</tr>
<tr>
<td>43.8</td>
<td>2.623E+07</td>
<td>2.623E+07</td>
<td>1</td>
<td>0</td>
<td>26227890 Loop: CACHE_MISS</td>
</tr>
<tr>
<td>43.8</td>
<td>2.623E+07</td>
<td>2.623E+07</td>
<td>1</td>
<td>0</td>
<td>26227480 Loop: CACHE_NO_MISS</td>
</tr>
</tbody>
</table>

**MULTI_GET_TIME_OF_DAY/profile.-1.0.0**

<table>
<thead>
<tr>
<th>%Time</th>
<th>Exclusive</th>
<th>Inclusive</th>
<th>#Call</th>
<th>#Subrs</th>
<th>Inclusive Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>2</td>
<td>79</td>
<td>1</td>
<td>2</td>
<td>79073 MATMUL_CACHE</td>
</tr>
<tr>
<td>88.6</td>
<td>0.046</td>
<td>70</td>
<td>1</td>
<td>1</td>
<td>70078 CACHE_MISS</td>
</tr>
<tr>
<td>88.6</td>
<td>70</td>
<td>70</td>
<td>1</td>
<td>0</td>
<td>70032 Loop: CACHE_MISS</td>
</tr>
<tr>
<td>7.9</td>
<td>0.02</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>6230 CACHE_NO_MISS</td>
</tr>
<tr>
<td>7.9</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>6210 Loop: CACHE_NO_MISS</td>
</tr>
</tbody>
</table>

**CACHE_MISS 375 MFLOPS**  
**CACHE_NO_MISS 4.4 GFLOPS**
program matmul_cache
   !Just calling the following functions here
end program matmul_cache

real function cache_miss(a, b, n)
   integer :: i, j
   real :: a(1024,1024), b(1024,1024)
   do i = 1, n
      do j = 1, n
         a(i,j) = b(i,j)
      enddo
   enddo
end function cache_miss

real function cache_no_miss(a, b, n)
   integer :: i, j
   real :: a(1024,1024), b(1024,1024)
   do j = 1, n
      do i = 1, n
         a(i,j) = b(i,j)
      enddo
   enddo
end function cache_no_miss
Hardware Counters

• To measure more counters at the same time,
  
  ```
  export TAU_METRICS=TIME:PAPI_FP_INS:PAPI_L1_DCM
  ```

• Each counter will generate one profile in separate subdirectories that look like,
  
  ```
  MULTI__GET_TIME_OF_DAY, MULTI__PAPI_L1_DCM
  ```

• Intel and AMD nodes have different counters

• Up to 25 counters/events can then be recorded at a time
Tracing

- What happens in my code at a given time? When?
- Use Jumpshot to visualize results
- Significant overhead. Turn off profiling!

```bash
$ export TAU_MAKEFILE=/panfs/storage.local/\> opt/hpc/gnu/openmpi/tau/x86_64/lib/\> Makefile.tau-papi-mpi-pdt-trace
And compile your code, run
After job is finished, cd to TRACEDIR
$ tau_treemerge.pl
$ tau2slog2 tau.trc tau.edf -o tau.slog2
$ jumpshot tau.slog2
```
Tracing – Jumpshot view