TAU Potpourri
and Working with Open Components, Interfaces, and Environments

Scott Biersdorff, Chee Wai Lee, Allen D. Malony, Sameer Shende, Wyatt Spear
{scottb,cheele,al-malony,shende,wspear}@cs.uoregon.edu

Dept. Computer and Information Science
Performance Research Laboratory
University of Oregon

Potpourri: a mixture of dried petals and spices placed in a bowl, origin 17th century, from French, literally ‘rotten pot’
Petal and Spices

- Binary instrumentation: DyninstAPI and *tau_run*
- Hybrid performance measurement: TAUebs
- Library wrapping/interposition: *tau_wrap*, *tau_exec*, *PARMCI*
- Heterogeneous performance measurement: TAUcuda
- HPC program development and tools: Eclipse and TAU
- Monitoring running applications: TAUmon
- Potpourri smell test
The Pot
Binary Instrumentation: DyninstAPI and tau_run

- TAU and DyninstAPI are mature technologies for performance instrumentation, measurement and analysis
- TAU has been a long-time user of DyninstAPI
- Using DyninstAPI’s recent binary re-writing capabilities, created a binary re-writer tool for TAU (tau_run)
  - Supports TAU's performance instrumentation
  - Works with TAU instrumentation selection
    - files and routines based on exclude/include lists
  - TAU’s measurement library (DSO) is loaded by tau_run
- Runtime (pre-execution) and binary re-writing are both supported
- Simplifies code instrumentation and tool usage greatly!
- Included on POINT LiveDVD (tau.uoregon.edu/point.iso)
tau_run with NAS PBS

/home/livetau% cd ~/tutorial
/home/livetau/tutorial% # Build an uninstrumented bt NAS Parallel Benchmark
/home/livetau/tutorial% make bt CLASS=W NPROCS=4
/home/livetau/tutorial% cd bin
/home/livetau/tutorial/bin% # Run the instrumented code
/home/livetau/tutorial/bin% mpirun -np 4 ./bt_W.4
/home/livetau/tutorial/bin%
/home/livetau/tutorial/bin% # Instrument the executable using TAU with DyninstAPI
/home/livetau/tutorial/bin% tau_run ./bt_W.4 -o ./bt.i
/home/livetau/tutorial/bin% rm -rf profile.* MULT*
/home/livetau/tutorial/bin% mpirun -np 4 ./bt.i
/home/livetau/tutorial/bin% paraprof
/home/livetau/tutorial/bin% # Choose a different TAU configuration
/home/livetau/tutorial/bin% ls $TAU/libTAUsh
libTAUsh-depthlimit-mpi-pdt.so*
libTAUsh-mpi-pdt.so*
libTAUsh-mpi-pdt-upc.so*
libTAUsh-mpi-python-pdt.so*
libTAUsh-papi-mpi-pdt.so*
libTAUsh-papi-mpi-pdt-upc.so*
libTAUsh-papi-mpi-pdt-upc-udp.so*
libTAUsh-papi-mpi-pdt-vampirtrace-trace.so*
libTAUsh-papi-mpi-python-pdt.so*
/home/livetau/tutorial/bin% tau_run -XrunTAUsh-papi-mpi-pdt-vampirtrace-trace bt_W.4 -o bt.vpt
/home/livetau/tutorial/bin% setenv VT_METRICS PAPI_FP_INS:PAPI_L1_DCM
/home/livetau/tutorial/bin% mpirun -np 4 ./bt.vpt
/home/livetau/tutorial/bin% vampir bt.vpt.otf &
Going Forward

- Currently, `tau_run` only supports dynamic executables (v6.1)
- Would like support for static binary rewriting
- Would like support for rewriting shared objects
- Validation for compilers other than gcc
  - XLC, PathScale, Cray CCE, Intel, PGI,…
- Availability for more platforms
  - Apple Mac OS X, Windows, IBM BG/P, AIX, …
- Instrumentation at the loop level
- Interaction with generic binary instrumentation
Hybrid Performance Measurement: TAUebs

- Integrate sampling-based and probe-based measurement
- TAUebs combines TAU, PerfSuite, and HPCToolkit
  - TAU for probe-based instrumentation and measurement
  - PerfSuite technology for timer-based sampling
  - HPCToolkit for call stack unwinding on fully-optimized codes
    - problems with StackWalkerAPI at the time ... will retry
- Foundation is TAU with linked SBM capabilities
  - "Context" linking between event stack and call stack
  - Augment PBM with SBM performance views
- TAUebs measurement
- Capture a trace of EBS samples, each containing:
  - Timestamp, TAUkey, PCkey, hardware counters, delta time
TAUebs Data Analysis (Profile)

- Process EBS trace in two ways: profile, trace
- Merged profile analysis with ParaProf
  - Augments TAU profile with PC call stack information
  - Merge stacks for each sample and update TAU profile
  - For all samples that match on TAUkey:
    - distribute TAU inclusive time across PC locations
  - Intermediate routine parent nodes will be inserted in profile
    - only compute inclusive time
  - Can aggregate callsites or show explicitly
- Instrumentation spectrum
  - Top-level on (main) then get profile entirely from EBS
  - All routines then get PC locations merged in profile
**TAUebs Data Analysis (Trace)**

- EBS to OTF trace converter
- Analyze EBS trace with powerful trace analysis tools
- For each sample
  - Place timestamp in trace record
  - Merge TAU event stack and PC call stack into merged call path
  - Create event ID for merged call path and put in trace record
  - Put collected PAPI metrics in trace record
  - Can store PC locations in trace record
Real World Examples

- MADNESS (quantum chemistry application)
  - Heavy use of C++ templates and new features
  - Assembly regions/files and lots of code in header files
  - Makes source instrumentation a challenge
    - TAU source instrumenter could handle a fair amount
  - Instrumentation overhead kicks TAU's butt
    - GNU compiler instrumentation saw 2901% overhead
    - many small routines (getter/setter)
    - TAU source instrumentation with selection (<6%)
      - introduces potential blind spots

- GPAW

- FLASH
TAUebs Profile for MADNESS

- 11 minute run on 8 threads produces 67 MB per thread
- Significant time in .TKLOOP16 which is an assembly file
- Profile for each thread
TAUebs Trace for MADNESS

- Vampir call stack color-coded by file name
- Flops rate
GPAW (Grid-Based Projector-Augmented Wave)

- Mixed Python, C, MPI run on 128 processes
- Python performance interface and LD_PRELOAD

Python routines

LAPACK
Library interposition/wrapping: tau_exec, tau_wrap

- Performance evaluation tools such as TAU provide a wealth of options to measure the performance of an application.
- Need to simplify TAU usage to easily evaluate performance properties, including I/O, memory, and communication.
- Designed a new tool (tau_exec) that leverages runtime instrumentation by pre-loading measurement libraries.
- Works on dynamic executables (default under Linux).
- Substitutes I/O, MPI, and memory allocation/deallocation routines with instrumented calls.
  - Interval events (e.g., time spent in write())
  - Atomic events (e.g., how much memory was allocated).
- Measure I/O and memory usage.
**TAU Execution Command (tau_exec)**

- **Uninstrumented execution**
  - `% mpirun –np 256 ./a.out`

- **Track MPI performance**
  - `% mpirun –np 256 tau_exec ./a.out`

- **Track I/O and MPI performance (MPI enabled by default)**
  - `% mpirun –np 256 tau_exec –io ./a.out`

- **Track memory operations**
  - `% setenv TAU_TRACK_MEMORY_LEAKS 1`
  - `% mpirun –np 256 tau_exec –memory ./a.out`

- **Track I/O performance and memory operations**
  - `% mpirun –np 256 tau_exec –io –memory ./a.out`
POSIX I/O Calls Supported

- Unbuffered I/O
  - `open`, `open64`, `close`, `read`, `write`, `readv`, `writev`, `creat`, `creat64`

- Buffered I/O
  - `fopen`, `fopen64`, `fdopen`, `freopen`, `fclose`
  - `fprintf`, `fscanf`, `fwrite`, `fread`

- Communication
  - `socket`, `pipe`, `socketpair`, `bind`, `accept`, `connect`
  - `recv`, `send`, `sendto`, `recvfrom`, `pclose`

- Control
  - `fcntl`, `rewind`, `lseek`, `lseek64`, `fseek`, `dup`, `dup2`, `mkstep`, `tmpfile`

- Asynchronous I/O
  - `aio_{read,write,suspend,cancel,return}`, `lio_listio`
HELIOS Rotorcraft Simulation

- HPC Institute for Advanced Rotorcraft Modeling and Simulation (HIARMS)
  - Andy Wissink, US Army, Aeroflight Dynamics Directorate, Ames Research

- Multi-language framework
  - Python (SIF)
  - C/C++
  - Fortran
HELIOS OBE Test

- I/O and memory measurements with Python wrapper

Uninstrumented Execution:

```bash
> mpirun -np 4 pyMPI ./obe.py
>
> echo "To use with TAU, set PYTHONPATH, and PATH"

To use with TAU, set PYTHONPATH, and PATH

> setenv PYTHONPATH $PET_HOME/pkgs/tau2/x86_64/lib/bindings-icpc-mpi-pthread-python-pdt/:${PYTHONPATH}
> set path=($PET_HOME/pkgs/tau2/x86_64/bin $path)
> cat wrapper.py

import tau

def OurMain():
    import obe

tau.run('OurMain()')

> mpirun -np 4 tau_exec -io -memory -T python pyMPI ./wrapper.py
> paraprof
> pprof
```
Helios OBE Profile
## Volume of I/O by File and Memory

<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>MeanValue</th>
<th>NumSamples</th>
<th>MinValue</th>
<th>MaxValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>.TAU application</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fopen64()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fclose()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OurMain()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>malloc size</td>
<td>25,235</td>
<td>1,097.174</td>
<td>23</td>
<td>11</td>
<td>12,032</td>
<td>2,851.143</td>
</tr>
<tr>
<td>free size</td>
<td>22,707</td>
<td>1,746.692</td>
<td>13</td>
<td>11</td>
<td>12,032</td>
<td>3,660.642</td>
</tr>
<tr>
<td>OurMain [{wrapper.py}[3]]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>malloc size</td>
<td>3,877</td>
<td>323.083</td>
<td>12</td>
<td>32</td>
<td>981</td>
<td>252.72</td>
</tr>
<tr>
<td>free size</td>
<td>1,536</td>
<td>219.429</td>
<td>7</td>
<td>32</td>
<td>464</td>
<td>148.122</td>
</tr>
<tr>
<td>fopen64()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fclose()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;module&gt; [{lode.py}[8]]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>writeRestartData [{samarcInterface.py}[145]]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>samarcWriteRestartData</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE Bandwidth (MB/s) &lt;file=&quot;samarc/restore.00002/nodes.00004/proc.00001&quot;&gt;</td>
<td>74.565</td>
<td>117</td>
<td>0</td>
<td>2,156.889</td>
<td>246.386</td>
<td></td>
</tr>
<tr>
<td>WRITE Bandwidth (MB/s) &lt;file=&quot;samarc/restore.00001/nodes.00004/proc.00001&quot;&gt;</td>
<td>77.594</td>
<td>117</td>
<td>0</td>
<td>1,941.2</td>
<td>228.366</td>
<td></td>
</tr>
<tr>
<td>WRITE Bandwidth (MB/s)</td>
<td>76.08</td>
<td>234</td>
<td>0</td>
<td>2,156.889</td>
<td>237.551</td>
<td></td>
</tr>
<tr>
<td>Bytes Written &lt;file=&quot;samarc/restore.00002/nodes.00004/proc.00001&quot;&gt;</td>
<td>2,097,552</td>
<td>17,927.795</td>
<td>117</td>
<td>1</td>
<td>1,048,576</td>
<td>133,362.946</td>
</tr>
<tr>
<td>Bytes Written &lt;file=&quot;samarc/restore.00001/nodes.00004/proc.00001&quot;&gt;</td>
<td>2,097,552</td>
<td>17,927.795</td>
<td>117</td>
<td>1</td>
<td>1,048,576</td>
<td>133,362.946</td>
</tr>
<tr>
<td>Bytes Written</td>
<td>4,195,104</td>
<td>17,927.795</td>
<td>234</td>
<td>1</td>
<td>1,048,576</td>
<td>133,362.946</td>
</tr>
</tbody>
</table>
# Memory Leaks in MPI

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>MeanValue</th>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TAU application</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>MPI_Finalize()</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size</td>
<td>23,901,253</td>
<td>22,719.822</td>
<td>1,052</td>
<td>2,099,200</td>
<td>2</td>
<td>186,920.948</td>
</tr>
<tr>
<td>malloc size</td>
<td>5,013,902</td>
<td>65,972.395</td>
<td>76</td>
<td>5,000,000</td>
<td>2</td>
<td>569,732.815</td>
</tr>
<tr>
<td>MEMORY LEAK!</td>
<td>5,000,264</td>
<td>500,026.4</td>
<td>10</td>
<td>5,000,000</td>
<td>3</td>
<td>1,499,991.2</td>
</tr>
<tr>
<td><strong>read()</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bytes Read</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>READ Bandwidth (MB/s)  &lt;file=&quot;pipe&quot;&gt;</td>
<td>0.308</td>
<td>1</td>
<td>0.308</td>
<td>0.308</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Bytes Read &lt;file=&quot;pipe&quot;&gt;</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>READ Bandwidth (MB/s)</td>
<td>0.308</td>
<td>1</td>
<td>0.308</td>
<td>0.308</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>write()</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE Bandwidth (MB/s) &lt;file=&quot;/dev/infiniband/rdma_cm&quot;&gt;</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>0</td>
<td>1.472</td>
</tr>
<tr>
<td>Bytes Written</td>
<td>1,456</td>
<td>14.275</td>
<td>102</td>
<td>28</td>
<td>0</td>
<td>5.149</td>
</tr>
<tr>
<td>Bytes Written &lt;file=&quot;/dev/infiniband/uverbs0&quot;&gt;</td>
<td>0.528</td>
<td>97</td>
<td>12</td>
<td>0.089</td>
<td>1.32</td>
<td></td>
</tr>
<tr>
<td>WRITE Bandwidth (MB/s) &lt;file=&quot;/dev/infiniband/rdma_cm&quot;&gt;</td>
<td>64</td>
<td>16</td>
<td>4</td>
<td>28</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>Bytes Written &lt;file=&quot;/dev/infiniband/uverbs0&quot;&gt;</td>
<td>1.714</td>
<td>97</td>
<td>1</td>
<td>1.714</td>
<td>0</td>
<td>4.562</td>
</tr>
<tr>
<td>WRITE Bandwidth (MB/s) &lt;file=&quot;/pipe&quot;&gt;</td>
<td>1,368</td>
<td>14.103</td>
<td>97</td>
<td>24</td>
<td>12</td>
<td>2.644</td>
</tr>
<tr>
<td><strong>writev()</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE Bandwidth (MB/s) &lt;file=&quot;/dev/infiniband/uverbs0&quot;&gt;</td>
<td>2.967</td>
<td>4</td>
<td>5.6</td>
<td>0</td>
<td>0</td>
<td>2.644</td>
</tr>
<tr>
<td>Bytes Written</td>
<td>1,368</td>
<td>14.103</td>
<td>97</td>
<td>24</td>
<td>12</td>
<td>4.562</td>
</tr>
<tr>
<td>Bytes Written &lt;file=&quot;/dev/infiniband/uverbs0&quot;&gt;</td>
<td>2.967</td>
<td>4</td>
<td>5.6</td>
<td>0</td>
<td>0</td>
<td>2.644</td>
</tr>
<tr>
<td><strong>readv()</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bytes Read</td>
<td>112</td>
<td>28</td>
<td>4</td>
<td>36</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>READ Bandwidth (MB/s) &lt;file=&quot;/socket&quot;&gt;</td>
<td>25.5</td>
<td>4</td>
<td>36</td>
<td>10</td>
<td>11.079</td>
<td></td>
</tr>
<tr>
<td>Bytes Read &lt;file=&quot;/socket&quot;&gt;</td>
<td>112</td>
<td>28</td>
<td>4</td>
<td>36</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>READ Bandwidth (MB/s)</td>
<td>25.5</td>
<td>4</td>
<td>36</td>
<td>10</td>
<td>11.079</td>
<td></td>
</tr>
<tr>
<td><strong>MPI_Comm_free()</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size</td>
<td>10,952</td>
<td>195.571</td>
<td>56</td>
<td>1,024</td>
<td>48</td>
<td>255.353</td>
</tr>
<tr>
<td><strong>fopen64()</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size</td>
<td>231,314</td>
<td>263,456</td>
<td>878</td>
<td>568</td>
<td>35</td>
<td>221,272</td>
</tr>
<tr>
<td>MEMORY LEAK!</td>
<td>1,105,956</td>
<td>1,686.169</td>
<td>592</td>
<td>7,200</td>
<td>32</td>
<td>3,078.574</td>
</tr>
<tr>
<td>malloc size</td>
<td>1,358,286</td>
<td>901.318</td>
<td>1,507</td>
<td>7,200</td>
<td>32</td>
<td>2,087.737</td>
</tr>
</tbody>
</table>

CScADS 2010  TAU Potpourri
Library wrapping: tau_wrap

- How to instrument an external library without source?
  - Source may not be available
  - Library may be too cumbersome to build (with instrumentation)
- Build a library wrapper tools
  - Used PDT to parse header files
  - Generate new header files with instrumentation files
- Application is instrumented
- Add the -I<wrapper> directory to the command line
- C pre-processor will substitute our headers
  - Redirects references at routine callsite to a wrapper call
  - Wrapper internally calls the original
  - Wrapper has TAU measurement code
HDF5 Library Wrapping

Usage:

```
[sameer@zorak]$ tau_wrap hdf5.h.pdb hdf5.h -o hdf5.inst.c -f select.tau -g hdf5
```

- instrumented wrapper library source (hdf5.inst.c)
- instrumentation specification file (select.tau)
- group (hdf5)
- creates the wrapper/ directory

<table>
<thead>
<tr>
<th>%Time</th>
<th>Exclusive msec</th>
<th>Inclusive total msec</th>
<th>#Call</th>
<th>#Subrs</th>
<th>Inclusive Name usec/call</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>0.057</td>
<td>0.057</td>
<td>1</td>
<td>13</td>
<td>1236 int main(void) C</td>
</tr>
<tr>
<td>70.8</td>
<td>0.875</td>
<td>0.875</td>
<td>1</td>
<td>0</td>
<td>875 hid_t H5Fcreate()</td>
</tr>
<tr>
<td>9.7</td>
<td>0.12</td>
<td>0.12</td>
<td>1</td>
<td>0</td>
<td>120 herr_t H5Fclose()</td>
</tr>
<tr>
<td>6.0</td>
<td>0.074</td>
<td>0.074</td>
<td>1</td>
<td>0</td>
<td>74 hid_t H5Dcreate()</td>
</tr>
<tr>
<td>3.1</td>
<td>0.038</td>
<td>0.038</td>
<td>1</td>
<td>0</td>
<td>38 herr_t H5Dwrite()</td>
</tr>
<tr>
<td>2.6</td>
<td>0.032</td>
<td>0.032</td>
<td>1</td>
<td>0</td>
<td>32 herr_t H5Dclose()</td>
</tr>
<tr>
<td>2.1</td>
<td>0.026</td>
<td>0.026</td>
<td>1</td>
<td>0</td>
<td>26 herr_t H5check_version()</td>
</tr>
<tr>
<td>0.6</td>
<td>0.008</td>
<td>0.008</td>
<td>1</td>
<td>0</td>
<td>8 hid_t H5Screate_simple()</td>
</tr>
<tr>
<td>0.2</td>
<td>0.002</td>
<td>0.002</td>
<td>1</td>
<td>0</td>
<td>2 herr_t H5Tset_order()</td>
</tr>
<tr>
<td>0.2</td>
<td>0.002</td>
<td>0.002</td>
<td>1</td>
<td>0</td>
<td>2 hid_t H5Tcopy()</td>
</tr>
<tr>
<td>0.1</td>
<td>0.001</td>
<td>0.001</td>
<td>1</td>
<td>0</td>
<td>1 herr_t H5Sclose()</td>
</tr>
<tr>
<td>0.1</td>
<td>0.001</td>
<td>0.001</td>
<td>2</td>
<td>0</td>
<td>0 herr_t H5open()</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0 herr_t H5Tclose()</td>
</tr>
</tbody>
</table>
NWChem and One-sided Communication

- NWChem relies on Global Arrays (GA)
  - GA is a PGAS programming model
  - provides a global view of a physically distributed array
  - one-sided access to arbitrary patches of data
  - developed as a library
  - fully interoperable with MPI

- Aggregate Remote Memory Copy Interface (ARMCI) is the
  - GA communication substrate for one-sided communication
  - portable high-performance one-sided communication library
  - rich set of remote memory access primitives

- Difficult to test representative workloads for NWChem
  - Lack of use cases for one-side programming models
NWChem Characterization

- Strong-scaling of modest problems helps to understand the behavior of larger scientifically significant problems
  - represent behavior of real calculations on future systems

- Understand interplay between data-server and compute processes as a function of scaling
  - Large numerical computation per node at small scale can obscure the cost of maintaining passive-target progress
  - Larger scale decreases numerical work per node and increases the fragmentation of data, increasing messages
  - Vary #nodes, cores-per-node, and memory buffer pinning

- Understand trade-off of core allocation
  - all to computation versus some to communication
NWChem Instrumentation

- Source-base instrumentation of NWChem application routines
- Developed an ARMCI interposition library (PARMCI)
  - defines weak symbols and name-shifted PARMCI interface
  - similar to PMPI for MPI
- Developed a TAU PARMCI library
  - intervals events around interface routines
  - atomic events to capture communication size and destination
- Wrapped external libraries
  - BLAS (DGEMM)
- Need portable instrumentation to conduct cross-platform experiments
FUSION Tests with Varying Cores

- Scaling on 24, 32, 48, 64, 96 and 128 nodes
- Test on 8 and 7 cores with pinning disabled
  - Dedicated data server with 7 cores
- Relative ARMI communication overhead increases with greater number of nodes (cores)
**Intrepid Tests**

- Scaling on 64, 128, 256 and 512 nodes
- Tests with interrupt or communication helper thread (CHT)
  - CHT requires a core to be allocated
- ARMCI calls are barely noticeable
- DAXPY calculation shows up more
- CHT performs better in both SMP and DUAL modes
Heterogeneous Systems Measurement: TAUcuda

- Want to create performance views that capture heterogeneous concurrency and execution behavior
  - Reflect interactions between heterogeneous parts
  - Capture performance semantics relative to computation model
  - Assimilate performance for all execution paths for shared view

- What perspective do we have of the parts?
  - Determines the semantics of the measurement data
  - Determines assumptions about behavior and interactions
  - Performance views may have to work with reduced data

- Need to work with heterogeneous system components

- Developed TAUcuda for CUDA performance measurement
  - TAUcuda v1 discussed at CScADS 2009
TAUcuda Performance Measurement (Version 2)

- Overcome TAUcuda (v1) deficiencies
  - Required source code instrumentation
  - Event interface only perspectives
    - could not see memory transfer or CUDA system execution
- CUDA system architecture
  - Implemented by CUDA libraries
    - driver and device (cuXXX) libraries
    - runtime (cudaYYY) library
  - Tools support (Parallel Nsight (Nexus), CUDA Profiler)
    - not intended to integrate with other HPC performance tools
- TAUcuda (v2) built on experimental Linux CUDA driver
  - Linux CUDA driver R190.86 supports a callback interface!!!
TAUcuda Architecture

[Diagram showing the architecture of TAUcuda, including nodes, TAUcuda events, and the relationship with TAU and CUDA systems.]
CUDA Linpack Profile (4 processes, 4 GPUs)

- Measure performance of heterogeneous parallel applications
- GPU-accelerated Linpack benchmark (M. Fatica, NVIDIA)
CUDA Linpack Trace

CUDA memory transfer (white)  MPI communication (yellow)
**SHOC Stencil2D (512 iterations, 4 CPUxGPU)**

- Scalable Heterogeneous Computing benchmark suite
  - CUDA / OpenCL kernels and microbenchmarks (ORNL)

CUDA memory transfer (white)
**TAU and Eclipse**

- How to make performance measurement, analysis, and tuning a part of the software development cycle?
- Multi-year work with Eclipse IDE (www.eclipse.org)
  - Benefits: portable, project transition: familiar interface, supports multiple languages (Java, C/C++, Fortran, …)
  - Features: syntax highlighting, refactoring, code management
  - Modular plug-in based architecture allows for easy extension
  - Environments: JDT, CDT, PTP (www.eclipse.org/{jdt,cdt,ptp})
- High-performance software development environments
  - IDE features for parallel programming + parallel tools
  - Eclipse PTP: integrate features and interface with parallel tools
**TAU and Eclipse**

- Provide an interface for configuring TAU’s automatic instrumentation within Eclipse’s build system
- Manage runtime configuration settings and environment variables for execution of TAU instrumented programs
Integration Features

- Chose different TAU configurations
- Select options for control of instrumentation and compilation
- Integrated interface for generating and choosing selective instrumentation
- PAPI counter selection
- Profile data generated in Eclipse is stored in a PerfDMF database
- Performance databases can be browsed from within Eclipse
  - Trials loaded in the ParaProf
- Source callback allows quick navigation
Dynamic Tool Definitions

- Developed *External Tools Framework* (ETFw)
  - Initially to extend and generalize the TAU plug-ins
  - Considered for general tool integration in Eclipse
- TAU plug-ins’ functionality was generalized to XML for:
  - Portability and ease of modification
  - Simpler alternative to Eclipse plug-in for tool integration
  - Use additionally for workflow creation
- Tools selected/configured in a launch configuration window
- ETFx adds Eclipse support for analysis tools including:
  - Valgrind, PerfSuite, Scalasca, VampirTrace
- Other tool developers are leveraging the ETFw
Monitoring running Applications: TAUmon

- Scalable access to a running application’s performance information is valuable
- Access can happen after an application completes (but before parallel teardown) or while an application is still running
- Two-way access needed for support of advanced operations
- TAUmon
  - Design as a transport-neutral application monitoring framework
  - Base on prior/existing work with various transport systems:
    - Supermon, MRNet, MPI
- Recent work by Chee Wai Lee
Overall design principles

- Modular and transparent access to parallel transport systems
- Support for minimal user intervention with different system-specific launch mechanisms
- Modular support for scalable monitoring operations
  - Based on aggregation algorithms and techniques
  - Simple overall statistics: mean, min, max, standard deviation
  - Histograms
  - Clustering results (various types)
- Modular support for data delivery to output locations
  - Local or remote visualization/analysis tools
  - Local or remote storage
Current implementation and API

- `TAU_ONLINE_DUMP()` collective operations in application
  - Called by all thread / processes
  - Works with parallel profiles
- Appropriate version of TAU selected for transport system
- User instruments application with TAU support for desired monitoring transport system
- User submits instrumented application to parallel job system
- Other launch systems must be submitted along with the application to the job scheduler as needed
  - Currently supported through different machine-specific job-submission scripts
TAUmon and MRNet

- Overview
  - Scripts set up runtime infrastructure
  - TAU frontend coordinates gathering operations when requested
  - Application backends collectively initiate operations in a push-based approach
  - MRNet tree nodes facilitate scalable gather operations
MRNet Network Configuration

- Scripts used to set up MRNet network configuration
  - Given $P = \text{number of cores}$ for the application, the user can choose an appropriate $N = \text{number of tree nodes}$ and $K = \text{fanout}$ for deciding how to allocate sufficient computing resources for both application and MRNet
  - Number of network leaves can be computed as $(N/K)*(K-1)$
  - Probe processes discover and partition computing resources between the application and MRNet
  - `mrnet_topgen` utility will write a topology file given $K$ and $N$ and a list of processor hosts available exclusively for MRNet
  - TAU frontend reads topology file to create the MRNet tree and then write a new file to inform application how it can connect to the leaves of the tree
Monitoring Operation with MRNet

- Application collectively invokes `TAU_ONLINE_DUMP()` to start monitoring operations using current performance information.
- TAU data is accessed and sent through MRNet’s communication API via streams and filters.
- Filters perform appropriate aggregation operations on data.
- TAU frontend is responsible for collecting the data, storing it, and eventual delivery to a consumer.
**Experiences with MRNet - 1**

- Parallel system-specific (e.g. Cray XT5 and BG/P) launch mechanisms required

- Key technical challenges:
  - Efficient data offload from application to MRNet tree
  - Support for user control of MRNet tree for performance

- Other challenges:
  - Current compiler-related incompatibility on the Cray
  - Providing uniform launch scripts across different parallel machines
Experiences with MRNet - 2

- Extra computing resources must be dedicated to MRNet tree
  - This can be viewed as an advantage or limitation
- Resources required are system-dependent:
  - On Cray systems, MRNet front-end resides on login node, intermediate tree nodes reside on dedicated (set aside by user) compute nodes, application processes (backends) reside on the remaining compute nodes
  - On BG/P systems, MRNet front-end (and possibly some tree nodes) reside on login node, intermediate tree nodes reside on IO nodes (not known a-priori until after compute nodes are launched), backends reside on compute nodes
TAUmon and MPI

- Also developing TAUmon to use MPI-based transport
  - No separate launch mechanisms required
  - Parallel gather operations implemented as a binomial heap with staged MPI point-to-point calls (Rank 0 serves as root)

- Limitations:
  - Application shares the same parallel infrastructure with monitoring transport
  - Monitoring operations may cause performance intrusion
  - Currently, no flexibility for user control of transport network configuration
TAUMon: Early results with PFLOTRAN (Cray)

- MRNet as transport
  - Only exclusive time is being monitored

<table>
<thead>
<tr>
<th>XT5 Nodes (Total)</th>
<th>Cores (Total)</th>
<th>Cores (Application Only)</th>
<th>Mean Aggregation Time (per iteration)</th>
<th>Histogram Generation Time (per iteration)</th>
</tr>
</thead>
<tbody>
<tr>
<td>374</td>
<td>4,488</td>
<td>4,104</td>
<td>0.2204s</td>
<td>0.07313s</td>
</tr>
<tr>
<td>559</td>
<td>6,708</td>
<td>6,144</td>
<td>0.3308s</td>
<td>0.1411s</td>
</tr>
<tr>
<td>746</td>
<td>8,952</td>
<td>8,196</td>
<td>0.4586s</td>
<td>0.1864s</td>
</tr>
<tr>
<td>1,118</td>
<td>13,416</td>
<td>12,288</td>
<td>0.6439s</td>
<td>0.2839s</td>
</tr>
</tbody>
</table>
TAUMon: Early results with PFLOTRAN (Cray)

- MPI as transport
- Only exclusive time is being monitored

<table>
<thead>
<tr>
<th>XT5 Nodes</th>
<th>Cores</th>
<th>Unification Time (per iteration)</th>
<th>Mean Aggregation Time (per iteration)</th>
<th>Histogram Generation Time (per iteration)</th>
<th>Total Operation Time (per iteration)</th>
</tr>
</thead>
<tbody>
<tr>
<td>342</td>
<td>4,104</td>
<td>0.02514s</td>
<td>0.01677s</td>
<td>2.339s</td>
<td>2.384s</td>
</tr>
<tr>
<td>512</td>
<td>6,144</td>
<td>0.02244s</td>
<td>0.02215s</td>
<td>2.06s</td>
<td>2.115s</td>
</tr>
<tr>
<td>683</td>
<td>8,196</td>
<td>0.04067s</td>
<td>0.03347s</td>
<td>3.651s</td>
<td>4.278s</td>
</tr>
<tr>
<td>1,024</td>
<td>12,288</td>
<td>0.07241s</td>
<td>0.02621s</td>
<td>0.8643s</td>
<td>0.9676s</td>
</tr>
<tr>
<td>1,366</td>
<td>16,392</td>
<td>0.03382s</td>
<td>0.01431s</td>
<td>1.861s</td>
<td>3.053s</td>
</tr>
<tr>
<td>2,048</td>
<td>24,576</td>
<td>0.02976s</td>
<td>0.03569s</td>
<td>0.6238s</td>
<td>0.6921s</td>
</tr>
</tbody>
</table>
**TAUMon: Early results with PFLOTRAN (Cray)**

- MRNet as transport
  - 4104 cores running with 374 extra cores for MRNet transport
- Each line bar shows the mean profile of an iteration
TAUmon: Early visualization with ParaProf

- A quick side-by-side look at data monitored using MPI (left column) and MRNet (right column)
- MPI_Allreduce (blue triangle) appears to grow inordinately over time when PFLOTRAN is executed on 8,196 cores
Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science
  - ASC/NNSA

- Department of Defense (DoD)
  - HPC Modernization Office (HPCMO)

- NSF Software Development for Cyberinfrastructure (SDCI)

- Research Centre Juelich

- Argonne National Laboratory

- Technical University Dresden

- ParaTools, Inc.

- NVIDIA