Performance Engineering using MVAPICH and TAU

Sameer Shende
University of Oregon

SC18 Talk
The Ohio State University Booth (#4404)
Wednesday, November 14, 2018, 2pm – 2:30pm
http://tau.uoregon.edu/tau_osu_sc18.pdf
Outline

- Introduction
- The MPI Tools Interfaces and Benefits
- Integrating TAU and MVAPICH2 with MPI_T
- Use Cases
- TAU Performance System®
Acknowledgments

- The MVAPICH2 team The Ohio State University
  - http://mvapich.cse.ohio-state.edu
- TAU team at the University of Oregon
  - http://tau.uoregon.edu
TAU Performance System®

• Tuning and Analysis Utilities (22+ year project)
• Comprehensive performance profiling and tracing
  • Integrated, scalable, flexible, portable
  • Targets all parallel programming/execution paradigms

• Integrated performance toolkit
  • Instrumentation, measurement, analysis, visualization
  • Widely-ported performance profiling / tracing system
  • Performance data management and data mining
  • Open source (BSD-style license)
  • Uses performance and control variables to interface with MVAPICH2

• Integrates with application frameworks
• http://tau.uoregon.edu
Understanding Application Performance using TAU

• **How much time** is spent in each application routine and outer *loops*? Within loops, what is the contribution of each *statement*?

• **How many instructions** are executed in these code regions? Floating point, Level 1 and 2 *data cache misses*, hits, branches taken?

• **What is the memory usage** of the code? When and where is memory allocated/de-allocated? Are there any memory leaks?

• **What are the I/O characteristics** of the code? What is the peak read and write *bandwidth* of individual calls, total volume?

• **What is the contribution of each phase** of the program? What is the time wasted/spent waiting for collectives, and I/O operations in Initialization, Computation, I/O phases?

• **How does the application scale**? What is the efficiency, runtime breakdown of performance across different core counts?

• **How can I tune MPI for better performance**? What performance and control does MVAPICH2 export to observe and control its performance?
Overview of the MVAPICH2 Project
High Performance open-source MPI Library for InfiniBand, Omni-Path, Ethernet/iWARP, and RDMA over Converged Ethernet (RoCE)

• MVAPICH (MPI-1), MVAPICH2 (MPI-2.2 and MPI-3.1), Started in 2001, First version available in 2002
• MVAPICH2-X (MPI + PGAS), Available since 2011
• Support for GPGPUs (MVAPICH2-GDR) and MIC (MVAPICH2-MIC), Available since 2014
• Support for Virtualization (MVAPICH2-Virt), Available since 2015
• Support for Energy-Awareness (MVAPICH2-EA), Available since 2015
• Support for InfiniBand Network Analysis and Monitoring (OSU INAM) since 2015

• Used by more than 2,925 organizations in 86 countries
• More than 481,000 (> 0.48 million) downloads from the OSU site directly
• Empowering many TOP500 clusters (Jun ‘18 ranking)
  • 2nd, 10,649,600-core (Sunway TaihuLight) at National Supercomputing Center in Wuxi, China
  • 12th, 556,104 cores (Oakforest-PACS) in Japan
  • 15th, 367,024 cores (Stampede2) at TACC
  • 24th, 241,108-core (Pleiades) at NASA
  • 62nd, 76,032-core (Tsubame 2.5) at Tokyo Institute of Technology
• Available with software stacks of many vendors and Linux Distros (RedHat and SuSE)

• http://mvapich.cse.ohio-state.edu

Empowering Top500 systems for over a decade
Outline

• Introduction
• The MPI Tools Interfaces and Benefits
• Integrating TAU and MVAPICH2 with MPI_T
• Use Cases
• TAU Performance System®
MVAPICH2 and TAU

- TAU and MVAPICH2 are enhanced with the ability to generate recommendations and engineering performance report
- MPI libraries like MVAPICH2 are now “reconfigurable” at runtime
- TAU and MVAPICH2 communicate using the MPI-T interface
Why PMPI is not good enough?

- Takes a “black box” view of the MPI library
int MPI_T_init_thread(int required, int *provided);
int MPI_T_cvar_get_num(int *num_cvar);
int MPI_T_cvar_get_info(int cvar_index, char *name, int *name_len, int *verbosity, MPI_Datatype *datatype, MPI_T_enum *enumtype, char *desc, int *desc_len, int *bind, int *scope);
int MPI_T_pvar_session_create(MPI_T_pvar_session *session);
int MPI_T_pvar_handle_alloc(MPI_T_pvar_session *session, int pvar_index, void *obj_handle, MPI_T_pvar_handle *handle, int *count);
int MPI_T_pvar_start(MPI_T_pvar_session *session, MPI_T_pvar_handle handle);
int MPI_T_pvar_read(MPI_T_pvar_session *session, MPI_T_pvar_handle handle, void *buf);
int MPI_T_pvar_reset(MPI_T_pvar_session *session, MPI_T_pvar_handle handle);
int MPI_T_pvar_handle_free(MPI_T_pvar_session *session, MPI_T_pvar_handle *handle);
int MPI_T_pvar_session_free(MPI_T_pvar_session *session);
int MPI_T_finalize(void);
MPI_T support with MVAPICH2

- Support performance variables (PVAR)
  - Variables to track different components within the MPI library
- Initial support for Control Variables (CVAR)
  - Variables to modify the behavior of MPI Library

- Memory Usage:
  - current level
  - maximum watermark

- InfiniBand N/W:
  - #control packets
  - #out-of-order packets

- Pt-to-pt messages:
  - unexpected queue length
  - unexp. match attempts
  - recvq. length

- Registration cache:
  - hits
  - misses

- Shared-memory:
  - limic/ CMA
  - buffer pool size & usage

- Collective ops:
  - comm. creation
  - #algorithm invocations
  [Bcast – 8; Gather – 10]
Co-designing Applications to use MPI-T

Example Pseudo-code: Optimizing the eager limit dynamically:

```c
MPI_T_init_thread(..)
MPI_T_cvar_get_info(MV2_EAGER_THRESHOLD)
if (msg_size < MV2_EAGER_THRESHOLD + 1KB)
    MPI_T_cvar_write(MV2_EAGER_THRESHOLD, +1024)
MPI_Send(..)
MPI_T_finalize(..)
```
Outline

• Introduction
• The MPI Tools Interfaces and Benefits
• Integrating TAU and MVAPICH2 with MPI_T
• Use Cases
• TAU Performance System®
Interacting TAU with MVAPICH2 through MPI_T Interface

- Enhance existing support for MPI_T in MVAPICH2 to expose a richer set of performance and control variables
- Get and display MPI Performance Variables (PVARs) made available by the runtime in TAU
- Control the runtime’s behavior via MPI Control Variables (CVARs)
- Add support to MVAPICH2 and TAU for interactive performance engineering sessions
Plugin-based Infrastructure for Non-Interactive Tuning

- Performance data collected by TAU
  - Support for PVARs and CVARs
  - Setting CVARs to control MVAPICH2
  - Studying performance data in TAU’s ParaProf profile browser
- Multiple plugins available for
  - Tuning application at runtime and
  - Generate post-run recommendations
Enhancing MPI_T Support

- Introduced support for new MPI_T based CVARs to MVAPICH2
  - **MPIR_CVAR_MAX_INLINE_MSG_SZ**
    - Controls the message size up to which “inline” transmission of data is supported by MVAPICH2
  - **MPIR_CVAR_VBUF_POOL_SIZE**
    - Controls the number of internal communication buffers (VBUFs) MVAPICH2 allocates initially. Also, 
      **MPIR_CVAR_VBUF_POOL_REDUCED_VALUE[1] ([2…n])**
  - **MPIR_CVAR_VBUF_SECONDARY_POOL_SIZE**
    - Controls the number of VBUFs MVAPICH2 allocates when there are no more free VBUFs available
  - **MPIR_CVAR_IBA_EAGER_THRESHOLD**
    - Controls the message size where MVAPICH2 switches from eager to rendezvous protocol for large messages

- **TAU enhanced with support for setting MPI_T CVARs in a non-interactive mode for uninstrumented applications**
MVAPICH2

- Several new MPI_T based PVARs added to MVAPICH2
  - mv2_vbuf_max_use, mv2_total_vbuf_memory etc
- Enhanced TAU with support for tracking of MPI_T PVARs and CVARs for uninstrumented applications
  - ParaProf, TAU’s visualization front end, enhanced with support for displaying PVARs and CVARs
  - TAU provides tau_exec, a tool to transparently instrument MPI routines
    - Uninstrumented:
      % mpirun –np 1024 ./a.out
    - Instrumented:
      % mpirun –np 1024 tau_exec [options] ./a.out
      % paraprof
<table>
<thead>
<tr>
<th>PVARs Exposed by MVAPICH2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Field</strong></td>
</tr>
<tr>
<td>MPI_T_PVAR0() mem_allocated</td>
</tr>
<tr>
<td>MPI_T_PVAR1() mv2_num_2level_comm_success</td>
</tr>
<tr>
<td>MPI_T_PVAR11() mv2_num_shmem_coll_calls</td>
</tr>
<tr>
<td>MPI_T_PVAR12() mptr_progress_poll</td>
</tr>
<tr>
<td>MPI_T_PVAR13() mv2_memp_read_progress_poll</td>
</tr>
<tr>
<td>MPI_T_PVAR14() mv2_memp_write_progress_poll</td>
</tr>
<tr>
<td>MPI_T_PVAR15() mv2_memp_read_progress_poll_suc</td>
</tr>
<tr>
<td>MPI_T_PVAR15() mv2_memp_write_progress_poll_suc</td>
</tr>
<tr>
<td>MPI_T_PVAR17() rdma_ud_retransmissions</td>
</tr>
<tr>
<td>MPI_T_PVAR18() mv2_coll_bcast_bimodal</td>
</tr>
<tr>
<td>MPI_T_PVAR19() mv2_coll_bcast_scatter absorbing_all</td>
</tr>
<tr>
<td>MPI_T_PVAR20() mv2_coll_bcast_scatter absorbing</td>
</tr>
<tr>
<td>MPI_T_PVAR21() mv2_coll_bcast_scatter_ring algo</td>
</tr>
<tr>
<td>MPI_T_PVAR22() mv2_coll_bcast_shmem</td>
</tr>
<tr>
<td>MPI_T_PVAR23() mv2_coll_bcast_kromial_internode</td>
</tr>
<tr>
<td>MPI_T_PVAR24() mv2_coll_bcast_kromial_intranode</td>
</tr>
<tr>
<td>MPI_T_PVAR25() mv2_coll_bcast_mcst_intranode</td>
</tr>
<tr>
<td>MPI_T_PVAR26() mv2_coll_bcast_doubled</td>
</tr>
<tr>
<td>MPI_T_PVAR27() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR28() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR29() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR30() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR31() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR32() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR33() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR34() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR35() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR36() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR37() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR38() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR39() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR40() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR41() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR42() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR43() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR44() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR45() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR46() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR47() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR48() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR49() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR50() mv2_coll_bcast_doubled_all</td>
</tr>
<tr>
<td>MPI_T_PVAR51() mv2_coll_bcast_doubled_all</td>
</tr>
</tbody>
</table>

The Ohio State University

UNIVERSITY OF OREGON
<table>
<thead>
<tr>
<th>CVAR Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVAPI_CH3_PORT_RANGE</td>
<td>The MVAPI_CH3_PORT_RANGE environment variable allows you to specify the range of TCP ports to use for MVAPI operations.</td>
</tr>
<tr>
<td>MVAPI_CH3_EDGEX_Ix_EAGER_SIZE</td>
<td>The MVAPI_CH3_EDGEX_Ix_EAGER_SIZE environment variable allows you to specify how many words in the eager state should be considered for eager transfer.</td>
</tr>
<tr>
<td>MVAPI_CH3 to MVAPI interface</td>
<td>Enables the eager state transfer for MVAPI operations.</td>
</tr>
</tbody>
</table>

**Example Usage:**
- MVAPI_CH3_PORT_RANGE = 10000-10010
- MVAPI_CH3_EDGEX_Ix_EAGER_SIZE = 1024

This configuration ensures efficient use of network resources and reduces latency for MVAPI operations.
Outline

• Introduction
• The MPI Tools Interfaces and Benefits
• Integrating TAU and MVAPICH2 with MPI_T
• Use Cases
  • Designing Dynamic and Adaptive MPI Point-to-point Protocols
• TAU Performance System®
Point-to-point Communication Protocols in MPI

- **Eager Protocol**
  - Best communication performance for smaller messages
- **Rendezvous Protocol**
  - Best communication performance for larger messages
Analyzing Communication Costs of Point-to-point Protocols

- **Eager Protocol**
  - Best communication performance for smaller messages
Analyzing Communication Costs of Point-to-point Protocols (Cont.)

- **Rendezvous Protocol**
  - Best communication performance for larger messages
Studying the Performance and Overlap of 3D Stencil Benchmark

- **Default**: Uses eager protocol for small messages and rendezvous for large
- **Manually Tuned**: Forces the use of eager for all message sizes
- **Manually Tuned** has degradation in raw communication performance
- **Manually Tuned** has significant benefits for overlap
- **Manually Tuned** better for overall application execution time
Analyzing Overlap Potential of Eager Protocol

• Application processes schedule communication operation
• Network adapter progresses communication in the background
• Application process free to perform useful compute in the foreground
• Overlap of computation and communication => Better Overall Application Performance
• Increased buffer requirement
• Poor communication performance if used for all types of communication operations

Impact of changing Eager Threshold on performance of multi-pair message-rate benchmark with 32 processes on Stampede
Analyzing Overlap Potential of Rendezvous Protocol

- Application processes schedule communication operation
- Application process free to perform useful compute in the foreground
- Little communication progress in the background
- All communication takes place at final synchronization

- Reduced buffer requirement
- Good communication performance if used for large message sizes and operations where communication library is progressed frequently

- Poor overlap of computation and communication => Poor Overall Application Performance
But... What if Applications Progress Communication Frequently?

- Application processes schedule communication operation
- Application process free to perform useful compute in the foreground

- Overlap of computation and communication => Better Overall Application Performance
- Reduced buffer requirement
- Good communication performance as communication library is progressed frequently
- Harder to create such programs that progress communication at the exact time without causing overhead
- Communication support entities (threads, hardware engines, etc.) have their own complexities
Can we design *dynamic and adaptive* point-to-point communication mechanisms that can deliver the best

1. Communication performance
2. Overlap of computation and communication
3. Memory footprint
Proposed Designs and Expected Benefits at a High-level

Eager Threshold for Example Communication Pattern with Different Designs

<table>
<thead>
<tr>
<th>Process Pair</th>
<th>Eager Threshold (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 4</td>
<td>32</td>
</tr>
<tr>
<td>1 – 5</td>
<td>64</td>
</tr>
<tr>
<td>2 – 6</td>
<td>128</td>
</tr>
<tr>
<td>3 – 7</td>
<td>32</td>
</tr>
</tbody>
</table>

- Default
  - Poor overlap; Low memory requirement
- Manually Tuned
  - Good overlap; High memory requirement
- Dynamic + Adaptive
  - Good overlap; Optimal memory requirement
Identifying the New Eager-Threshold and Allocating Resources for Change

- \[ \text{Threshold}_{\text{new}} = 2^\left\lfloor \log_2 \left( \frac{\sum \text{sizeof(Rndv Msg} + \text{Pkt Header})}{\text{Number of RndvMsgs}} \right) \right\rfloor + \text{offset} \]; Failure: “-1”
- Allocate larger internal communication buffers of larger size and receive queues
Designing Dynamic and Adaptive Point-to-point Protocols

- Process pair always has one active connection
- Messages will not have to wait for connection establishment
Mitigating Memory Footprint Requirements

- Increase in memory allocated is a concern
- Proposed design attempts to free them if
  - Buffer has not been in use continually for user defined period of time
  - Uses weights to determine which set of buffers are least recently used
  - Prevents trashing behavior where library gets into continuous loop of allocation and deallocation
- **Significantly** reduces memory overhead of dynamic and adaptive designs to less than 50% of what the manually tuned designs can offer.
Using MVAPICH2 and TAU

- To set CVARs or read PVARs using TAU for an uninstrumented binary:
  % export TAU_TRACK_MPI_T_PVARS=1
  % export TAU_MPI_T_CVAR_METRICS=
    MPIR_CVAR_VBUF_POOL_REDUCED_VALUE[1],
    MPIR_CVAR_IBA_EAGER_THRESHOLD
  % export TAU_MPI_T_CVAR_VALUES=32,64000
  % export PATH=/path/to/tau/x86_64/bin:$PATH
  % mpirun -np 1024 tau_exec -T mvapich2,mpit .a.out
  % paraprof
VBUF usage without CVARs

<table>
<thead>
<tr>
<th>Name</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
<th>NumSamples</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv2_total_vbuf_memory (Total amount of memory in bytes used for VBUFs)</td>
<td>3,313,056</td>
<td>3,313,056</td>
<td>3,313,056</td>
<td>0</td>
<td>1</td>
<td>3,313,056</td>
</tr>
<tr>
<td>mv2_ud_vbuf_allocated (Number of UD VBUFs allocated)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mv2_ud_vbuf_available (Number of UD VBUFs available)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mv2_ud_vbuf_freed (Number of UD VBUFs freed)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mv2_ud_vbuf_inuse (Number of UD VBUFs inuse)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mv2_ud_vbuf_max_use (Maximum number of UD VBUFs used)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mv2_vbuf_allocated (Number of VBUFs allocated)</td>
<td>320</td>
<td>320</td>
<td>320</td>
<td>0</td>
<td>1</td>
<td>320</td>
</tr>
<tr>
<td>mv2_vbuf_available (Number of VBUFs available)</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>1</td>
<td>255</td>
</tr>
<tr>
<td>mv2_vbuf_freed (Number of VBUFs freed)</td>
<td>25,545</td>
<td>25,545</td>
<td>25,545</td>
<td>0</td>
<td>1</td>
<td>25,545</td>
</tr>
<tr>
<td>mv2_vbuf_inuse (Number of VBUFs inuse)</td>
<td>65</td>
<td>65</td>
<td>65</td>
<td>0</td>
<td>1</td>
<td>65</td>
</tr>
<tr>
<td>mv2_vbuf_max_use (Maximum number of VBUFs used)</td>
<td>65</td>
<td>65</td>
<td>65</td>
<td>0</td>
<td>1</td>
<td>65</td>
</tr>
<tr>
<td>num_mallocCalls (Number of MPIT_malloc calls)</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>0</td>
<td>1</td>
<td>89</td>
</tr>
<tr>
<td>num_freeCalls (Number of MPIT_free calls)</td>
<td>47,801</td>
<td>47,801</td>
<td>47,801</td>
<td>0</td>
<td>1</td>
<td>47,801</td>
</tr>
<tr>
<td>num_mallocCalls (Number of MPIT_malloc calls)</td>
<td>49,258</td>
<td>49,258</td>
<td>49,258</td>
<td>0</td>
<td>1</td>
<td>49,258</td>
</tr>
<tr>
<td>num_memalignCalls (Number of MPIT_memalign calls)</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>0</td>
<td>1</td>
<td>34</td>
</tr>
<tr>
<td>num_memalign_freeCalls (Number of MPIT_memalign_free calls)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
VBUF usage with CVARs

Total memory used by VBUFs is reduced from 3,313,056 to 1,815,056
VBUF Memory Usage Without CVAR
VBUF Memory Usage With CVAR

% export TAU_TRACK_MPI_T_PVARS=1
% export TAU_MPI_T_CVAR_METRICS=MPIR_CVAR_VBUF_POOL_SIZE
% export TAU_MPI_T_CVAR_VALUES=16
% mpirun -np 1024 tau_exec -T mvapich2 .a.out
Outline

• Introduction
• The MPI Tools Interfaces and Benefits
• Integrating TAU and MVAPICH2 with MPI_T
• Use Cases
  • TAU Performance System®
Parallel performance framework and toolkit

- Supports all HPC platforms, compilers, runtime system
- Provides portable instrumentation, measurement, analysis
TAU Performance System

Instrumentation

• Fortran, C++, C, UPC, Java, Python, Chapel, Spark
• Automatic instrumentation

Measurement and analysis support

• MPI, OpenSHMEM, ARMCI, PGAS, DMAPP
• pthreads, OpenMP, OMPT interface, hybrid, other thread models
• GPU, CUDA, OpenCL, OpenACC
• Parallel profiling and tracing
• Use of Score-P for native OTF2 and CUBEX generation

Analysis

• Parallel profile analysis (ParaProf), data mining (PerfExplorer)
• Performance database technology (TAUdb)
• 3D profile browser
Instrumentation

Add hooks in the code to perform measurements

Source instrumentation using a preprocessor

- Add timer start/stop calls in a copy of the source code.
- Use Program Database Toolkit (PDT) for parsing source code.
- Requires recompiling the code using TAU shell scripts (tau_cc.sh, tau_f90.sh)
- Selective instrumentation (filter file) can reduce runtime overhead and narrow instrumentation focus.

Compiler-based instrumentation

- Use system compiler to add a special flag to insert hooks at routine entry/exit.
- Requires recompiling using TAU compiler scripts (tau_cc.sh, tau_f90.sh…)

Runtime preloading of TAU’s Dynamic Shared Object (DSO)

- No need to recompile code! Use mpirun tau_exec ./app with options.
- Requires dynamic executable (link using –dynamic on Theta).
TAU Instrumentation Approach

Supports both direct and indirect performance observation

- Direct instrumentation of program (system) code (probes)
- Instrumentation invokes performance measurement
- Event measurement: performance data, meta-data, context
- Indirect mode supports sampling based on periodic timer or hardware performance counter overflow based interrupts

Support for user-defined events

- **Interval** (Start/Stop) events to measure exclusive & inclusive duration
- **Atomic events** (Trigger at a single point with data, e.g., heap memory)
  - Measures total, samples, min/max/mean/std. deviation statistics
- **Context events** (are atomic events with executing context)
  - Measures above statistics for a given calling path
Direct Observation: Events

Event types

- Interval events (begin/end events)
  - Measures exclusive & inclusive durations between events
  - Metrics monotonically increase
- Atomic events (trigger with data value)
  - Used to capture performance data state
  - Shows extent of variation of triggered values (min/max/mean)

Code events

- Routines, classes, templates
- Statement-level blocks, loops
Inclusive and Exclusive Profiles

- Performance with respect to code regions
- Exclusive measurements for region only
- Inclusive measurements includes child regions

```c
int foo()
{
    int a;
    a = a + 1;
    bar();
    a = a + 1;
    return a;
}
```
How much data do you want?

- Limited Profile
- Loop Profile
- Callpath Profile
- Flat Profile
- Callsite Profile
- Trace

O(KB) to O(TB)
Types of Performance Profiles

**Flat profiles**
- Metric (e.g., time) spent in an event
- Exclusive/inclusive, # of calls, child calls, ...

**Callpath profiles**
- Time spent along a calling path (edges in callgraph)
- “main=>f1=>f2=>MPI_Send”
- Set the TAU_CALLPATH and TAU_CALLPATH_DEPTH environment variables

**Callsite profiles**
- Time spent along in an event at a given source location
- Set the TAU_CALLSITE environment variable

**Phase profiles**
- Flat profiles under a phase (nested phases allowed)
- Default “main” phase
- Supports static or dynamic (e.g. per-iteration) phases
ParaProf Profile Browser

Click “node X” next to see details
TAU – Flat Profile
ParaProf Thread Statistics Table

Right click over “node X” and choose Show Thread Statistics Table
TAU – Callsite Profiling

% export TAU_CALLSITE=1
% export TAU_CALLSITE=1
 Callsite Profiling and Tracing
Callsite Profiling and Tracing
TAU – Callstack Sampling

% export TAU_SAMPLING=1; export TAU_EBS_UNWIND=1
TAU – Event Based Sampling (EBS)

% export TAU_SAMPLING=1
TAU – Callpath Profiling

% export TAU_CALLPATH=1; export TAU_CALLPATH_DEPTH=100
## TAU Atomic Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes Written &lt;file=stdout&gt;</td>
<td>911</td>
<td>62</td>
<td>21</td>
<td>1</td>
<td>14.694</td>
<td>7.441</td>
</tr>
<tr>
<td>Bytes Written &lt;file=pipe&gt;</td>
<td>22</td>
<td>22</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Bytes Written &lt;file=Process_Output/VelRsdl.dat&gt;</td>
<td>7,826</td>
<td>100</td>
<td>302</td>
<td>76</td>
<td>78.26</td>
<td>22.487</td>
</tr>
<tr>
<td>Bytes Written &lt;file=Process_Output/MomRsdl.dat&gt;</td>
<td>7,826</td>
<td>100</td>
<td>302</td>
<td>76</td>
<td>78.26</td>
<td>22.487</td>
</tr>
<tr>
<td>Bytes Written &lt;file=Process_Output/MassRsdl.dat&gt;</td>
<td>11,325</td>
<td>100</td>
<td>435</td>
<td>110</td>
<td>113.25</td>
<td>32.337</td>
</tr>
<tr>
<td>Bytes Written &lt;file=Grid_Output/bodyYBndry.dat&gt;</td>
<td>9,724</td>
<td>5</td>
<td>8.192</td>
<td>4</td>
<td>1,944.8</td>
<td>3,174.201</td>
</tr>
<tr>
<td>Bytes Written &lt;file=/home/sameer/apps/sukra/RotCFD_Regression/case_catalog/UNS2D/NI/</td>
<td>45</td>
<td>1</td>
<td>45</td>
<td>45</td>
<td>45</td>
<td>0</td>
</tr>
<tr>
<td>Bytes Written &lt;file=./ReStarts/Restart_History/NACA0012_LargeGrid_00010.Rst&gt;</td>
<td>44,619,720</td>
<td>5,484</td>
<td>8.192</td>
<td>4</td>
<td>8,136,346</td>
<td>640,325</td>
</tr>
<tr>
<td>Bytes Written &lt;file=./ReStarts/Restart_History/NACA0012_LargeGrid_00005.Rst&gt;</td>
<td>44,619,720</td>
<td>5,484</td>
<td>8.192</td>
<td>4</td>
<td>8,136,346</td>
<td>640,325</td>
</tr>
<tr>
<td>Bytes Written &lt;file=./ReStarts/NACA0012_LargeGrid.Rst&gt;</td>
<td>44,619,720</td>
<td>5,484</td>
<td>8.192</td>
<td>4</td>
<td>8,136,346</td>
<td>640,325</td>
</tr>
<tr>
<td>Bytes Written &lt;file=./Process_Output/TurbRsdl.dat&gt;</td>
<td>4,271</td>
<td>72</td>
<td>224</td>
<td>57</td>
<td>59,319</td>
<td>19,544</td>
</tr>
<tr>
<td>Bytes Written &lt;file=./Process_Output/Solver.out&gt;</td>
<td>2,039</td>
<td>13</td>
<td>797</td>
<td>43</td>
<td>156,846</td>
<td>191,359</td>
</tr>
<tr>
<td>Bytes Written &lt;file=./Field_Solutions/Solution_History/NACA0012_LargeGrid_00010.Sln&gt;</td>
<td>4,356,976</td>
<td>534</td>
<td>8,192</td>
<td>4</td>
<td>8,159,131</td>
<td>501,319</td>
</tr>
<tr>
<td>Bytes Written &lt;file=./Field_Solutions/Solution_History/NACA0012_LargeGrid_00005.Sln&gt;</td>
<td>4,356,976</td>
<td>534</td>
<td>8,192</td>
<td>4</td>
<td>8,159,131</td>
<td>501,319</td>
</tr>
<tr>
<td>Bytes Written &lt;file=./Field_Solutions/NACA0012_LargeGrid.Sln&gt;</td>
<td>4,356,976</td>
<td>534</td>
<td>8,192</td>
<td>4</td>
<td>8,159,131</td>
<td>501,319</td>
</tr>
<tr>
<td>Bytes Written &lt;file=./Body_Pressure/NACA0012_LargeGrid_00010_body.Prs&gt;</td>
<td>65,986</td>
<td>9</td>
<td>8,190</td>
<td>1,300</td>
<td>7,331,778</td>
<td>2,133,204</td>
</tr>
<tr>
<td>Bytes Written &lt;file=./Body_Pressure/NACA0012_LargeGrid_00005_body.Prs&gt;</td>
<td>65,986</td>
<td>9</td>
<td>8,190</td>
<td>1,300</td>
<td>7,331,778</td>
<td>2,133,204</td>
</tr>
<tr>
<td>Bytes Written &lt;file=./Body_Pressure/FrcMnt.out&gt;</td>
<td>1,497</td>
<td>3</td>
<td>1,185</td>
<td>108</td>
<td>499</td>
<td>486,658</td>
</tr>
<tr>
<td>Bytes Written</td>
<td>147,107,546</td>
<td>18,550</td>
<td>8.192</td>
<td>1</td>
<td>7,930,326</td>
<td>1,420,352</td>
</tr>
</tbody>
</table>
## TAU – Context Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>Mean Value</th>
<th>NumSamples</th>
<th>Min Value</th>
<th>Max Value</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write bandwidth per file</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bytes written to each file</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Interval and Atomic Events in TAU

<table>
<thead>
<tr>
<th>Time</th>
<th>Exclusive</th>
<th>Inclusive</th>
<th>#Call</th>
<th>#Subrs</th>
<th>Inclusive Name, msec/call</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>0.187</td>
<td>1.105</td>
<td>1</td>
<td>44</td>
<td>1105659 int main(int, char **) C</td>
</tr>
<tr>
<td>93.2</td>
<td>1.030</td>
<td>1.030</td>
<td>1</td>
<td>0</td>
<td>1030854 MPI_Init()</td>
</tr>
<tr>
<td>5.9</td>
<td>0.879</td>
<td>65</td>
<td>40</td>
<td>320</td>
<td>1637 void func(int, int) C</td>
</tr>
<tr>
<td>4.6</td>
<td>51</td>
<td>51</td>
<td>40</td>
<td>0</td>
<td>1277 MPI_Barrier()</td>
</tr>
<tr>
<td>1.2</td>
<td>13</td>
<td>13</td>
<td>120</td>
<td>0</td>
<td>111 MPI_Recv()</td>
</tr>
<tr>
<td>0.8</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>9328 MPI_Finalize()</td>
</tr>
<tr>
<td>0.0</td>
<td>0.137</td>
<td>0.137</td>
<td>120</td>
<td>0</td>
<td>1 MPI_Send()</td>
</tr>
<tr>
<td>0.0</td>
<td>0.086</td>
<td>0.086</td>
<td>40</td>
<td>0</td>
<td>2 MPI_Bcast()</td>
</tr>
<tr>
<td>0.0</td>
<td>0.002</td>
<td>0.002</td>
<td>1</td>
<td>0</td>
<td>2 MPI_Comm_size()</td>
</tr>
<tr>
<td>0.0</td>
<td>0.001</td>
<td>0.001</td>
<td>1</td>
<td>0</td>
<td>1 MPI_Comm_rank()</td>
</tr>
</tbody>
</table>

Interval events e.g., routines (start/stop) show duration

Atomic events (triggered with value) show extent of variation (min/max/mean)

% export TAU_CALLPATH_DEPTH=0
% export TAU_TRACK_HEAP=1
Atomic Events, Context Events

<table>
<thead>
<tr>
<th>ZTime</th>
<th>Exclusive msec</th>
<th>Inclusive total msec</th>
<th>#Call</th>
<th>#Subr</th>
<th>Inclusive Name</th>
<th>Name</th>
<th>%</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>0.253</td>
<td>1.106</td>
<td>1</td>
<td>44</td>
<td>1106701</td>
<td>int main(int, char **) C</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>93.2</td>
<td>1.091</td>
<td>1.091</td>
<td>1</td>
<td>0</td>
<td>109311</td>
<td>MPI_Init()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.0</td>
<td>1</td>
<td>66</td>
<td>40</td>
<td>320</td>
<td>1650</td>
<td>void func(int, int) C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td>63</td>
<td>63</td>
<td>40</td>
<td>9119</td>
<td>1588</td>
<td>MPI_BARRIER()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>MPI_Finalize()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>1</td>
<td>120</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>MPI_RECV()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.141</td>
<td>0.141</td>
<td>120</td>
<td>0</td>
<td>2</td>
<td>MPI_SEND()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.001</td>
<td>0.001</td>
<td>40</td>
<td>0</td>
<td>1</td>
<td>MPI_COMM_SIZE()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>MPI_COMM_RANK()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Atomic events**
- Atomic events + executing context
- Controls depth of executing context shown in profiles

**Context events**
- Context events = atomic event + executing context

% export TAU_CALLPATH_DEPTH=1
% export TAU_TRACK_HEAP=1
Context Events (Default)

% export TAU_CALLPATH_DEPTH=2
% export TAU_TRACK_HEAP=1
Profiling

Shows how much time was spent in each routine

Tracing

Shows when events take place on a timeline
Types of Performance Profiles

**Flat profiles**
- Metric (e.g., time) spent in an event
- Exclusive/inclusive, # of calls, child calls, ...

**Callpath profiles**
- Time spent along a calling path (edges in callgraph)
- "main=> f1 => f2 => MPI_Send"
- Set the TAU_CALLPATH_DEPTH environment variable

**Phase profiles**
- Flat profiles under a phase (nested phases allowed)
- Default “main” phase
- Supports static or dynamic (e.g. per-iteration) phases
How much data do you want?

Limited Profile
Loop Profile
Callpath Profile

O(KB)  O(TB)

Flat Profile
Phase Profile
Trace

All levels support multiple metrics/counters
Performance Data Measurement

Direct via Probes

call `TAU_START`('potential')
// code
call `TAU_STOP`('potential')

- Exact measurement
- Fine-grain control
- Calls inserted into code

Indirect via Sampling

- No code modification
- Minimal effort
- Relies on debug symbols (-g option)
- `TAU_SAMPLING=1`
HPCLinux OVA: VirtualBox

Install VirtualBox from the USB Stick (dmg/exe file)
File -> Import Appliance -> Click browse -> HPCLinux -> <LITE>.ova -> Finish
Click on the appliance in VirtualBox -> Run
Username: livetau
Password:

% which paraprof
Step 1: Log-in to the Cluster (ri.cse.ohio-state.edu)

**Linux/OS X (Mac)**

Open your favorite Terminal

```bash
ssh -Y <username>@ri.cse.ohio-state.edu
```

Enter password

```bash
$ ls -l workshop
```

**Example:**

```bash
$ ssh -Y ritutXX@ri.cse.ohio-state.edu
```

(Replace XX with appropriate number from handout)

Enter Password:

```bash
$ ls
```

**Windows**

**Download Putty**

- https://www.putty.org/

**How to?**

- https://mediatemple.net/community/products/dv/204404604/using-ssh-in-putty-
- https://www.ssh.com/ssh/putty/windows/
Driving Example (3D Stencil)

3D Stencil benchmark

• Each process talks to at most **six** neighbors
• Two in each Cartesian dimension
  • X-right, X-left
  • Y-right, Y-left
  • Z-right, Z-left
• Repeat same communication pattern for multiple iterations
3Dstencil on RI (OSU)

cd ~/workshop/3Dstencil
sbatch -N 16 -p batch ./demo.sh
ls *.ppk
Copy ppk files to your laptop and use VirtualBox image
% paraprof *.ppk &
Example Codes On Stampede2.tacc.utexas.edu

• % cp ~tg457572/pkgs/workshop.tgz .
• % tar zxf workshop.tgz
• % source ~tg457572/tau.bashrc
• % idev -m 50 -r pearc-tau
  (requests a 50 minute interactive node with reservation name pearc-tau)
• % cd workshop/NPB3.1
• % make suite
• % cd bin
• % mpirun -np 64 ./lu.A.64
• % tau_pebil_rewrite lu.A.64 lu.i
• % mpirun -np 64 ./lu.i
• % pprof ; paraprof
Simplifying the use of TAU!

Uninstrumented code:

• % mpif90 –g –O3 matmult.f90
• % mpirun –np 16 ./a.out

With TAU:

• % mpirun –np 16 tau_exec ./a.out
• % paraprof

For more information at the statement level:

• % mpirun –np 16 tau_exec –ebs ./a.out (or use TAU_SAMPLING=1)
• To rewrite the binary to instrument individual functions (using PEBIL):
  • % tau_pebil_rewrite a.out a.inst; mpirun –np 16 ./a.inst (beta)
• % pprof -a | more
• % paraprof (GUI)
TAU for Heterogeneous Measurement

Multiple performance perspectives
Integrate Host-GPU support in TAU measurement framework
  • Enable use of each measurement approach
  • Include use of PAPI and CUPTI
  • Provide profiling and tracing support

Tutorial
  • Use TAU library wrapping of libraries
  • Use `tau_exec` to work with binaries
    % ./a.out (uninstrumented)
    % tau_exec –T <configuration tags> –cupti ./a.out
    % paraprof
TAU Execution Command (tau_exec)

Uninstrumented execution
• % mpirun -np 256 ./a.out

Track GPU operations
• % mpirun –np 256 tau_exec –cupti ./a.out
• % mpirun –np 256 tau_exec –cupti –um ./a.out (for Unified Memory)
• % mpirun –np 256 tau_exec –opencl ./a.out
• % mpirun –np 256 tau_exec –openacc ./a.out

Track MPI performance
• % mpirun -np 256 tau_exec ./a.out

Track OpenMP, and MPI performance (MPI enabled by default)
• % export TAU_OMPT_SUPPORT_LEVEL=full;
  % export TAU_OMPT_RESOLVE_ADDRESS_EAGERLY=1
• % mpirun -np 256 tau_exec –T ompt,tr6,mpi –ompt ./a.out

Track memory operations
• % export TAU_TRACK_MEMORY_LEAKS=1
• % mpirun –np 256 tau_exec –memory_debug ./a.out (bounds check)

Use event based sampling (compile with –g)
• % mpirun –np 256 tau_exec –ebs ./a.out
• Also –ebs_source=<PAPI_COUNTER> -ebs_period=<overflow_count>
Using TAU

TAU supports several measurement and thread options

Phase profiling, profiling with hardware counters (papi), MPI library, CUDA, Beacon (backplane for event notification – online monitoring), PDT (automatic source instrumentation) ...

Each measurement configuration of TAU corresponds to a unique stub makefile and library that is generated when you configure it

To instrument source code automatically using PDT

Choose an appropriate TAU stub makefile in <arch>/lib:

% source ~tg457572/tau.bashrc
% export TAU_MAKEFILE=$TAU/Makefile.tau-mvapich2-icpc-mpi-pdt
% export TAU_OPTIONS='--optVerbose ...' (see tau_compiler.sh)

Use tau_f90.sh, tau_cxx.sh, tau_upc.sh, or tau_cc.sh as F90, C++, UPC, or C compilers respectively:

% mpif90 foo.f90 changes to
% tau_f90.sh foo.f90

Set runtime environment variables, execute application and analyze performance data:

% pprof (for text based profile display)
% paraprof (for GUI)
Choosing TAU_MAKEFILE

% source ~tg457572/tau.bashrc
% ls $TAU/Makefile.*
/home1/00494/tg457572/pkgs/tau_latest/x86_64/lib/Makefile.tau-mvapich2-icpc-mpi-pdt
/home1/00494/tg457572/pkgs/tau_latest/x86_64/lib/Makefile.tau-mvapich2-icpc-mpi-pthread-pdt
/home1/00494/tg457572/pkgs/tau_latest/x86_64/lib/Makefile.tau-mvapich2-icpc-ompt-mpi-pdt-openmp
/home1/00494/tg457572/pkgs/tau_latest/x86_64/lib/Makefile.tau-mvapich2-icpc-papi-mpi-pdt-mpit
/home1/00494/tg457572/pkgs/tau_latest/x86_64/lib/Makefile.tau-mvapich2-icpc-papi-ompt-mpi-pdt-openmp
/home1/00494/tg457572/pkgs/tau_latest/x86_64/lib/Makefile.tau-pdt

For an MPI+F90 application with MPI, you may choose

Makefile.tau-mvapich2-icpc-mpi-pdt

- Supports MPI instrumentation, papi, and PDT for automatic source instrumentation

% export TAU_MAKEFILE=$TAU/Makefile.tau-mvapich2-icpc-mpi-pdt
% tau_f90.sh matrix.f90 -o matrix

OR with build systems:
% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
% cmake -DCMAKE_Fortran_COMPILER=tau_f90.sh
-DCMAKE_C_COMPILER=tau_cc.sh
-DCMAKE_CXX_COMPILER=tau_cxx.sh
% mpirun -np 1024 ./matrix
% paraprof
Configuration tags for tau_exec

% ./configure -pdt=<dir> -mpi -papi=<dir>; make install
Creates in $TAU:
Makefile.tau-papi-mpi-pdt (Configuration parameters in stub makefile)
shared-papi-mpi-pdt/libTAU.so

% ./configure -pdt=<dir> -mpi; make install creates
Makefile.tau-mpi-pdt
shared-mpi-pdt/libTAU.so

To explicitly choose preloading of shared-<options>/libTAU.so change:
% mpirun -np 256 ./a.out to
% mpirun -np 256 tau_exec -T <comma_separated_options> ./a.out

% mpirun -np 256 tau_exec -T papi,mpi,pdt ./a.out
Preloads $TAU/shared-papi-mpi-pdt/libTAU.so
% mpirun -np 256 tau_exec -T papi ./a.out
Preloads $TAU/shared-papi-mpi-pdt/libTAU.so by matching.
% mpirun -np 256 tau_exec -T papi,mpi,pdt -s ./a.out
Does not execute the program. Just displays the library that it will preload if executed
without the -s option.
NOTE: -mpi configuration is selected by default. Use -T serial for Sequential programs.
TAU’s Static Analysis System: Program Database Toolkit (PDT)

Application / Library

C / C++ parser

Fortran parser F77/90/95

IL

C / C++ IL analyzer

Fortran IL analyzer

Program Database Files

DUCTAPE

TAU instrumentor

Automatic source instrumentation
Automatic Source Instrumentation using PDT

- **TAU source analyzer**
- **Parsed program**
- **tau_instrumentor**
- **Instrumentation specification file**
- **Application source**
- **Instrumented copy of source**
Automatic Instrumentation

- **Use TAU’s compiler wrappers**
  - Simply replace `CXX` with `tau_cxx.sh`, etc.
  - Automatically instruments source code, links with TAU libraries.
- **Use `tau_cc.sh` for C, `tau_f90.sh` for Fortran, `tau_upc.sh` for UPC, etc.**

### Before

% cat Makefile
CXX = mpicxx
F90 = mpif90
CXXFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o ... fn.o

app: $(OBJ)
  $(CXX) $(LDFLAGS) $(OBJ) -o $@
  $(LIBS)
.cpp.o:
  $(CXX) $(CXXFLAGS) -c $<

% make

### After

% cat Makefile
CXX = tau_cxx.sh
F90 = tau_f90.sh
CXXFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o ... fn.o

app: $(OBJ)
  $(CXX) $(LDFLAGS) $(OBJ) -o $@
  $(LIBS)
.cpp.o:
  $(CXX) $(CXXFLAGS) -c $<

% export TAU_MAKEFILE=
  $TAU/Makefile.tau-papi-mpi-pdt
% make
Selective Instrumentation File: Compile time, Runtime

% export TAU_OPTIONS=‘-optTauSelectFile=select.tau ...’
% cat select.tau
BEGIN_INCLUDE_LIST
int main#
int dgemm#
END_INCLUDE_LIST
BEGIN_FILE_INCLUDE_LIST
Main.c
Blas/*.f77
END_FILE_INCLUDE_LIST
# replace include with exclude list

BEGIN_INSTRUMENT_SECTION
loops routine=“foo”
loops routine=“int main#”
END_INSTRUMENT_SECTION
# It can be used at compile time or at runtime:
% export TAU_SELECT_FILE = select.tau
Installing and Configuring TAU

• Installing PDT:
  • wget tau.uoregon.edu/pdt_lite.tgz
  • ./configure –prefix=<dir>; make ; make install

• Installing TAU:
  • wget tau.uoregon.edu/tau.tgz; tar zxf tau.tgz; cd tau-2.<ver>
  • wget http://tau.uoregon.edu/ext.tgz
  • ./configure –mpi -bfd=download -pdt=<dir> -papi=<dir> ...
  • make install

• Using TAU:
  • export TAU_MAKEFILE=<taudir>/x86_64/
    lib/Makefile.tau-<TAGS>
  • % export TAU_OPTIONS=‘-optTauSelectFile=select.tau’
  • make CC=tau_cc.sh  CXX=tau_cxx.sh  F90=tau_f90.sh
INSTALLING TAU on Laptops

Installing TAU under Mac OS X:
- Download Java
  - http://tau.uoregon.edu/java.dmg
  - Install java.dmg
  - wget http://tau.uoregon.edu/tau.dmg
  - Install tau.dmg

Installing TAU under Windows
- http://tau.uoregon.edu/tau.exe

Installing TAU under Linux
- http://tau.uoregon.edu/tau.tgz
  - ./configure; make install
  - export PATH=<taudir>/x86_64/bin:$PATH
NPB-MZ-MPI Suite

The NAS Parallel Benchmark suite (MPI+OpenMP version)

- Available from:
  
  http://www.nas.nasa.gov/Software/NPB

- 3 benchmarks in Fortran77
- Configurable for various sizes & classes

Move into the NPB3.3-MZ-MPI root directory

```
% ls
bin/  common/  jobscript/  Makefile  README.install  SP-MZ/
BT-MZ/  config/  LU-MZ/  README  README.tutorial  sys/
```

Subdirectories contain source code for each benchmark

- plus additional configuration and common code

The provided distribution has already been configured for the tutorial, such that it's ready to “make” one or more of the benchmarks and install them into a (tool-specific) “bin” subdirectory
NPB-MZ-MPI / BT (Block Tridiagonal Solver)

What does it do?

- Solves a discretized version of the unsteady, compressible Navier-Stokes equations in three spatial dimensions
- Performs 200 time-steps on a regular 3-dimensional grid

Implemented in 20 or so Fortran77 source modules

Uses MPI & OpenMP in combination

- 16 processes each with 4 threads should be reasonable
- bt-mz.B.16 should take around 1 minute
NPB-MZ-MPI / BT: config/make.def

#                      SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS.
#---------------------------------------------------------------

# Configured for generic MPI with GCC compiler
#---------------------------------------------------------------
#OPENMP = -fopenmp       # GCC compiler
OPENMP = -qopenmp -extend-source       # Intel compiler
...

# The Fortran compiler used for MPI programs
#---------------------------------------------------------------

F77 = mpif90       # Intel compiler

# Alternative variant to perform instrumentation
...
Building an NPB-MZ-MPI Benchmark

```
% make
===========================================
= NAS PARALLEL BENCHMARKS 3.3 =
= MPI+OpenMP Multi-Zone Versions =
= F77 =
===========================================

To make a NAS multi-zone benchmark type

    make <benchmark-name> CLASS=<class> NPROCS=<nprocs>

where <benchmark-name> is "bt-mz", "lu-mz", or "sp-mz"
<class> is "S", "W", "A" through "F"
<nprocs> is number of processes

[...]  

******************************************************************************
* Custom build configuration is specified in config/make.def  *
* Suggested tutorial exercise configuration for HPC systems:  *
*  
* make bt-mz CLASS=C NPROCS=8
*******************************************************************************
```
TAU Source Instrumentation

Edit `config/make.def` to adjust build configuration

- Uncomment specification of compiler/linker: `F77 = tau_f77.sh` or use `make F77=tau_f77.sh`

Make clean and build new tool-specific executable

Change to the directory containing the new executable before running it with the desired tool configuration
$ tau_exec

Usage: tau_exec [options] [-v] <exe> <exe options>

Options:
- v       Verbose mode
- s       Show what will be done but don't actually do anything (dryrun)
- qsub    Use qsub mode (BG/P only, see below)
- i o     Track I/O
- memory  Track memory allocation/deallocation
- memory_debug Enable memory debugger
- cuda    Track GPU events via CUDA
- cupti   Track GPU events via CUPTI (Also see env. variable TAU_CUPTI_API)
- opencl  Track GPU events via OpenCL
- openacc  Track GPU events via OpenACC (currently PGI only)
- ompt    Track OpenMP events via OMPT interface
- armci   Track ARMCI events via PARMCI
- ebs     Enable event-based sampling
- ebs_period=<count> Sampling period (default 1000)
- ebs_source=<counter> Counter (default itimer)
- um      Enable Unified Memory events via CUPTI
- T <DISABLE,GNU,ICPC,MPI,OMPT,OPENMP,PAPI,PDT,PROFILE,PTHREAD,SCOREP,SERIAL> : Specify TAU tags
- loadlib=<file.so> : Specify additional load library
- XrunTAUsh=<options> : Specify TAU library directly
- gdb     Run program in the gdb debugger

Notes:
Defaults if unspecified: -T MPI
MPI is assumed unless SERIAL is specified

No need to recompile the application!
tau_exec can enable event-based sampling while launching the executable using env TAU_SAMPLING=1 or tau_exec -ebs

Example:
```
mpirun -np 2 tau_exec -T icpc,ompt,mpi -ompt ./a.out
mpirun -np 2 tau_exec -io ./a.out
```

Example - event-based sampling with samples taken every 1,000,000 FP instructions
```
mpirun -np 8 tau_exec -ebs -ebs_period=1000000 -ebs_source=PAPI_FP_INS ./ring
```

Examples - GPU:
```
tau_exec -T serial,cupti -cupti ./matmult (Preferred for CUDA 4.1 or later)
tau_exec -openacc ./a.out
tau_exec -T serial -opencl ./a.out (OPENCL)
mpirun -np 2 tau_exec -T mpi,cupti,papi -cupti -um ./a.out (Unified Virtual Memory in CUDA 6.0+)
```

qsub mode (IBM BG/Q only):

Original:
```
qsub -n 1 --mode smp -t 10 ./a.out
```

With TAU:
```
tau_exec -qsub -io -memory -- qsub -n 1 ... -t 10 ./a.out
```

Memory Debugging:
```
-memory option:
Tracks heap allocation/deallocation and memory leaks.
-memory_debug option:
Detects memory leaks, checks for invalid alignment, and checks for array overflow. This is exactly like setting TAU_TRACK_MEMORY_LEAKS=1 and TAU_MEMDBG_PROTECT_ABOVE=1 and running with -memory
```
Event Based Sampling with TAU

Launch paraprof

% cd MZ-NPB3.3-MPI; cat README
% make clean;
% make suite
% cd bin
% idev -m 50 -r pearc-tau
% source ~tg457572/tau.bashrc
% export OMP_NUM_THREADS=4
% mpirun -np 4 tau_exec -T ompt -ebs ./bt-mz.B.4
% On head node:
% source ~tg457572/tau.bashrc
% paraprof

Right Click on Node 0 and choose Show Thread Statistics Table
ParaProf

Click on Columns: to sort by incl time

Open binvrchs

Click on Sample
<table>
<thead>
<tr>
<th>Name</th>
<th>Exclusive TIME</th>
<th>Inclusive TIME</th>
<th>Calls</th>
<th>Child Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/bin/ld64_f77-2.19</td>
<td>0.619</td>
<td>0.167</td>
<td>2,432</td>
<td>1</td>
</tr>
<tr>
<td>/usr/bin/ld64_f77-2.19</td>
<td>0.598</td>
<td>0.098</td>
<td>288</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/ld64_f77-2.19</td>
<td>0.138</td>
<td>0.028</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/ld64_f77-2.19</td>
<td>0.098</td>
<td>0.028</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/ld64_f77-2.19</td>
<td>0.087</td>
<td>0.028</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/ld64_f77-2.19</td>
<td>0.05</td>
<td>0.05</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/ld64_f77-2.19</td>
<td>0.047</td>
<td>0.047</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/ld64_f77-2.19</td>
<td>0.04</td>
<td>0.04</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/ld64_f77-2.19</td>
<td>0.04</td>
<td>0.04</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/ld64_f77-2.19</td>
<td>0.03</td>
<td>0.03</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/ld64_f77-2.19</td>
<td>0.03</td>
<td>0.03</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/ld64_f77-2.19</td>
<td>0.03</td>
<td>0.03</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/ld64_f77-2.19</td>
<td>0.03</td>
<td>0.03</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
Create a Selective Instrumentation File, Re-instrument, Re-run
ParaProf with Optimized Instrumentation
3D Visualization with ParaProf
ParaProf: Node 0

Optimized instrumentation!
Compile-Time Options

Optional parameters for the TAU_OPTIONS environment variable:
% tau_compiler.sh

- **-optVerbose**  
  Turn on verbose debugging messages
- **-optComplInst**  
  Use compiler based instrumentation
- **-optNoComplInst**  
  Do not revert to compiler instrumentation if source instrumentation fails.
- **-optTrackIO**  
  Wrap POSIX I/O call and calculates vol/bw of I/O operations  
  (Requires TAU to be configured with –iowrapper)
- **-optTrackGOMP**  
  Enable tracking GNU OpenMP runtime layer (used without –opari)
- **-optMemDbg**  
  Enable runtime bounds checking (see TAU_MEMDBG_* env vars)
- **-optKeepFiles**  
  Does not remove intermediate .pdb and .inst.* files
- **-optPreProcess**  
  Preprocess sources (OpenMP, Fortran) before instrumentation
- **-optTauSelectFile”=<file>”**  
  Specify selective instrumentation file for tau_instrumentor
- **-optTauWrapFile”=<file>”**  
  Specify path to link_options.tau generated by tau_gen_wrapper
- **-optHeaderInst**  
  Enable Instrumentation of headers
- **-optTrackUPCR**  
  Track UPC runtime layer routines (used with tau_upc.sh)
- **-optLinking”=“**  
  Options passed to the linker. Typically  
  $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
- **-optCompile”=“**  
  Options passed to the compiler. Typically  
  $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
- **-optPdtF95Opts”=“**  
  Add options for Fortran parser in PDT (f95parse/gfpars) …
Compile-Time Options (contd.)

Optional parameters for the TAU_OPTIONS environment variable:
% tau_compiler.sh

- optShared
  Use TAU’s shared library (libTAU.so) instead of static library (default)

- optPdtCxxOpts=""
  Options for C++ parser in PDT (cxxparse).

- optPdtF90Parser=""
  Specify a different Fortran parser

- optPdtCleanscapeParser
  Specify the Cleanscape Fortran parser instead of GNU gfparser

- optTau=""
  Specify options to the tau_instrumentor

- optTrackDMAPP
  Enable instrumentation of low-level DMAPP API calls on Cray

- optTrackPthread
  Enable instrumentation of pthread calls

See tau_compiler.sh for a full list of TAU_OPTIONS.
Measuring Memory Footprint

% export TAU_TRACK_MEMORY_FOOTPRINT=1
Paraprof:
Right click on a node -> Show Context Event Window -> see memory events
Usage Scenarios with MVAPICH2

• TAU measures the high water mark of total memory usage (TAU_TRACK_MEMORY_FOOTPRINT=1), finds that it is at 98% of available memory, and queries MVAPICH2 to find out how much memory it is using. Based on the number of pools allocated and used, it requests it to reduce the number of VBUF pools and controls the size of these pools using the MPI-T interface. The total memory footprint of the application reduces.

• TAU tracks the message sizes of messages (TAU_COMM_MATRIX=1), detects excessive time spent in MPI_Wait and other synchronization operations. It compares the average message size with the eager threshold and sets the new eager threshold value to match the message size. This could be done offline by re-executing the application with the new CVAR setting for eager threshold or online.

• TAU uses Beacon (backplane for event and control notification) to observe the performance of a running application (for e.g., vbuf pool statistics, high water mark of total and vbuf memory usage, message size statistics).
Performance/Overlap with 128KB Messages at Different Process Counts

- Dynamic Threshold has degradation in raw communication performance
- Dynamic Threshold has significant benefits for overlap
- Dynamic Threshold better for overall application execution time
Introspecting Impact of Eager Threshold on 3D Stencil Benchmark

Default

Optimized

Less Overlap, Less Useful work done by Application!

More Overlap, More Useful work done by Application!
Performance of Amber at Different Process Counts

- Optimal values selected manually (Manual Tuning) changes based on job size and problem size
  - Cumbersome, Error prone, and Impractical
- Dynamic Threshold delivers performance on par with best manually tuned version for all job/problem size
  - High Performance + High Productivity
Performance of Amber at Different Process Counts (Cont.)

Relative Memory Consumption

- Default design gives best memory scalability
  - Unable to deliver the best performance
- Dynamic Threshold able to keep memory footprint to what is absolutely needed to obtain performance benefits
Analyzing Dynamic Eager-Threshold Changes

- Number of Eager switches correspond to larger communication requirements at the application level

• Number of Eager switches correspond to larger communication requirements at the application level
**Time taken to Switch Thresholds and Allocate/Free Communication Buffers**

- Maximum overhead of establishing new connection is very low (~40 ms)
- Maximum cumulative time spent by each process for eager-threshold switching very low (< 0.5 s)
  - Less than 0.1% of overall execution time
- Time for dynamically allocating and freeing internal communication buffers also very low
  - Only a negligible percentage of the overall execution time
## Other Runtime Environment Variables

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU_TRACE</td>
<td>0</td>
<td>Setting to 1 turns on tracing</td>
</tr>
<tr>
<td>TAU_CALLPATH</td>
<td>0</td>
<td>Setting to 1 turns on callpath profiling</td>
</tr>
<tr>
<td>TAU_TRACK_MEMORY_FOOTPRINT</td>
<td>0</td>
<td>Setting to 1 turns on tracking memory usage by sampling periodically the resident set size and high water mark of memory usage</td>
</tr>
<tr>
<td>TAU_SELECT_FILE</td>
<td></td>
<td>Specify the path to runtime selective instrumentation file for filtering events using exclude and include lists of routines and/or files.</td>
</tr>
<tr>
<td>TAU_CALLPATH_DEPTH</td>
<td>2</td>
<td>Specifies depth of callpath. Setting to 0 generates no callpath or routine information, setting to 1 generates flat profile and context events have just parent information (e.g., Heap Entry: foo)</td>
</tr>
<tr>
<td>TAU_SAMPLING</td>
<td>0</td>
<td>Setting to 1 enables event-based sampling</td>
</tr>
<tr>
<td>TAU_TRACK_SIGNALS</td>
<td>0</td>
<td>Setting to 1 generate debugging callstack info when a program crashes</td>
</tr>
<tr>
<td>TAU_COMM_MATRIX</td>
<td>0</td>
<td>Setting to 1 generates communication matrix display using context events</td>
</tr>
<tr>
<td>TAU_THROTTLE</td>
<td>1</td>
<td>Setting to 0 turns off throttling. Enabled by default to remove instrumentation in lightweight routines that are called frequently</td>
</tr>
<tr>
<td>TAU_THROTTLE_NUMCALLS</td>
<td>100000</td>
<td>Specifies the number of calls before testing for throttling</td>
</tr>
<tr>
<td>TAU_THROTTLE_PERCALL</td>
<td>10</td>
<td>Specifies value in microseconds. Throttle a routine if it is called over 100000 times and takes less than 10 usec of inclusive time per call</td>
</tr>
<tr>
<td>TAU_COMPENSATE</td>
<td>0</td>
<td>Setting to 1 enables runtime compensation of instrumentation overhead</td>
</tr>
<tr>
<td>TAU_PROFILE_FORMAT</td>
<td>Profile</td>
<td>Setting to “merged” generates a single file. “snapshot” generates xml format</td>
</tr>
<tr>
<td>TAU_METRICS</td>
<td>TIME</td>
<td>Setting to a comma separated list generates other metrics. (e.g., TIME,ENERGY,PAPI_FP_INS,PAPI_NATIVE_&lt;event&gt;:&lt;subevent&gt;)</td>
</tr>
<tr>
<td>Environment Variable</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TAU_TRACK_MEMORYLeaks</td>
<td>0</td>
<td>Tracks allocates that were not de-allocated (needs –optMemDbg or tau_exec –memory)</td>
</tr>
<tr>
<td>TAU_EBS_SOURCE</td>
<td>TIME</td>
<td>Allows using PAPI hardware counters for periodic interrupts for EBS (e.g., TAU_EBS_SOURCE=PAPI_TOT_INS when TAU_SAMPLING=1)</td>
</tr>
<tr>
<td>TAU_EBS_PERIOD</td>
<td>100000</td>
<td>Specifies the overflow count for interrupts</td>
</tr>
<tr>
<td>TAU_MEMDBG_ALLOC_MIN/MAX</td>
<td>0</td>
<td>Byte size minimum and maximum subject to bounds checking (used with TAU_MEMDBG_PROTECT_*)</td>
</tr>
<tr>
<td>TAU_MEMDBG_OVERHEAD</td>
<td>0</td>
<td>Specifies the number of bytes for TAU’s memory overhead for memory debugging.</td>
</tr>
<tr>
<td>TAU_MEMDBG_PROTECT_BELOW/ ABOVE</td>
<td>0</td>
<td>Setting to 1 enables tracking runtime bounds checking below or above the array bounds (requires –optMemDbg while building or tau_exec –memory)</td>
</tr>
<tr>
<td>TAU_MEMDBG_ZERO_MALLOC</td>
<td>0</td>
<td>Setting to 1 enables tracking zero byte allocations as invalid memory allocations.</td>
</tr>
<tr>
<td>TAU_MEMDBG_PROTECT_FREE</td>
<td>0</td>
<td>Setting to 1 detects invalid accesses to deallocated memory that should not be referenced until it is reallocated (requires –optMemDbg or tau_exec –memory)</td>
</tr>
<tr>
<td>TAU_MEMDBG_ATTEMPT_CONTINUE</td>
<td>0</td>
<td>Setting to 1 allows TAU to record and continue execution when a memory error occurs at runtime.</td>
</tr>
<tr>
<td>TAU_MEMDBG_FILL_GAP</td>
<td>Undefined</td>
<td>Initial value for gap bytes</td>
</tr>
<tr>
<td>TAU_MEMDBG_ALINGMENT</td>
<td>sizeof(int)</td>
<td>Byte alignment for memory allocations</td>
</tr>
<tr>
<td>TAU_EVENT_THRESHOLD</td>
<td>0.5</td>
<td>Define a threshold value (e.g., .25 is 25%) to trigger marker events for min/ max</td>
</tr>
</tbody>
</table>
Evaluating Extent of Vectorization on MIC

% export TAU_MAKEFILE=$TAUROOT/mic_linux/lib/Makefile.tau-papi-mpi-pdt
% export TAU_METRICS=TIME, PAPI_NATIVE_VPU_ELEMENTS_ACTIVE, PAPI_NATIVE_VPU_INSTRUCTIONS_EXECUTED
ParaProf’s Topology Display Window (BGQ)
ParaProf’s Scalable 3D Visualization (BGQ)

786,432 ranks
ParaProf 3D Profile Browser
Download TAU from U. Oregon

http://www.hpclinux.com [OVA file]
http://tau.uoregon.edu/tau.tgz
for more information

Free download, open source, BSD license
PRL, University of Oregon, Eugene

www.uoregon.edu
Support Acknowledgments

US Department of Energy (DOE)
- ANL
- Office of Science contracts, ECP
- SciDAC, LBL contracts
- LLNL-LANL-SNL ASC/NNSA contract
- Battelle, PNNL and ORNL contract

CEA, France

Department of Defense (DoD)
- PETTT, HPCMP

National Science Foundation (NSF)
- SI2-SSI, Glassbox

Intel Corporation

NASA

Partners:
- University of Oregon
- The Ohio State University
- ParaTools, Inc.
- University of Tennessee, Knoxville
- T.U. Dresden, GWT
- Jülich Supercomputing Center