Performance Engineering using MVAPICH and TAU

Sameer Shende
University of Oregon and ParaTools, Inc.

MVAPICH Users Group Conference
Tuesday, August 24, 2021, 3:30pm – 4:00pm EST
http://mug.mvapich.cse.ohio-state.edu

Slides:
http://tau.uoregon.edu/TAU_MUG21.pdf

Outline

- Introduction
- The MPI Tools Interfaces and Benefits
- Integrating TAU and MVAPICH2 with MPI_T

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

Overview of the MVAPICH2 Project

High Performance open-source MPI Library

Support for multiple interconnects

- InfiniBand, Omni-Path, Ethernet/iWARP, RDMA over Converged Ethernet (RoCE), and AWS EFA

Support for multiple platforms

- x86, OpenPOWER, ARM, Xeon-Phi, GPGPUs (NVIDIA and AMD)

Started in 2001, first open-source version demonstrated at SC ‘02

Supports the latest MPI-3.1 standard

http://mvapich.cse.ohio-state.edu

Additional optimized versions for different systems/environments:

- MVAPICH2-X (Advanced MPI + PGAS), since 2011
- MVAPICH2-GDR with support for NVIDIA GPGPUs, since 2014
- MVAPICH2-MIC with support for Intel Xeon-Phi, since 2014
- MVAPICH2-Virt with virtualization support, since 2015
- MVAPICH2-EA with support for Energy-Awareness, since 2015
- MVAPICH2-Azure for Azure HPC IB instances, since 2019
- MVAPICH2-X-AWS for AWS HPC+EFA instances, since 2019

Tools:

- OSU MPI Micro-Benchmarks (OMB), since 2003
- OSU InfiniBand Network Analysis and Monitoring (INAM), since 2015

- Used by more than 3,175 organizations in 89 countries
- More than 1.41 Million downloads from the OSU site directly
- Empowering many TOP500 clusters (June ‘21 ranking)
  - 4th , 10,649,600-core (Sunway TaihuLight) at NSC, Wuxi, China
  - 10th, 448, 448 cores (Frontera) at TACC
  - 20th, 288,288 cores (Lassen) at LLNL
  - 31st, 570,020 cores (Nurion) in South Korea and many others
- Available with software stacks of many vendors and Linux Distros (RedHat, SuSE, OpenHPC, and Spack)
- Partner in the 10th ranked TACC Frontera system
- Empowering Top500 systems for more than 15 years

TAU Performance System®

- **Tuning and Analysis Utilities (25+ 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**

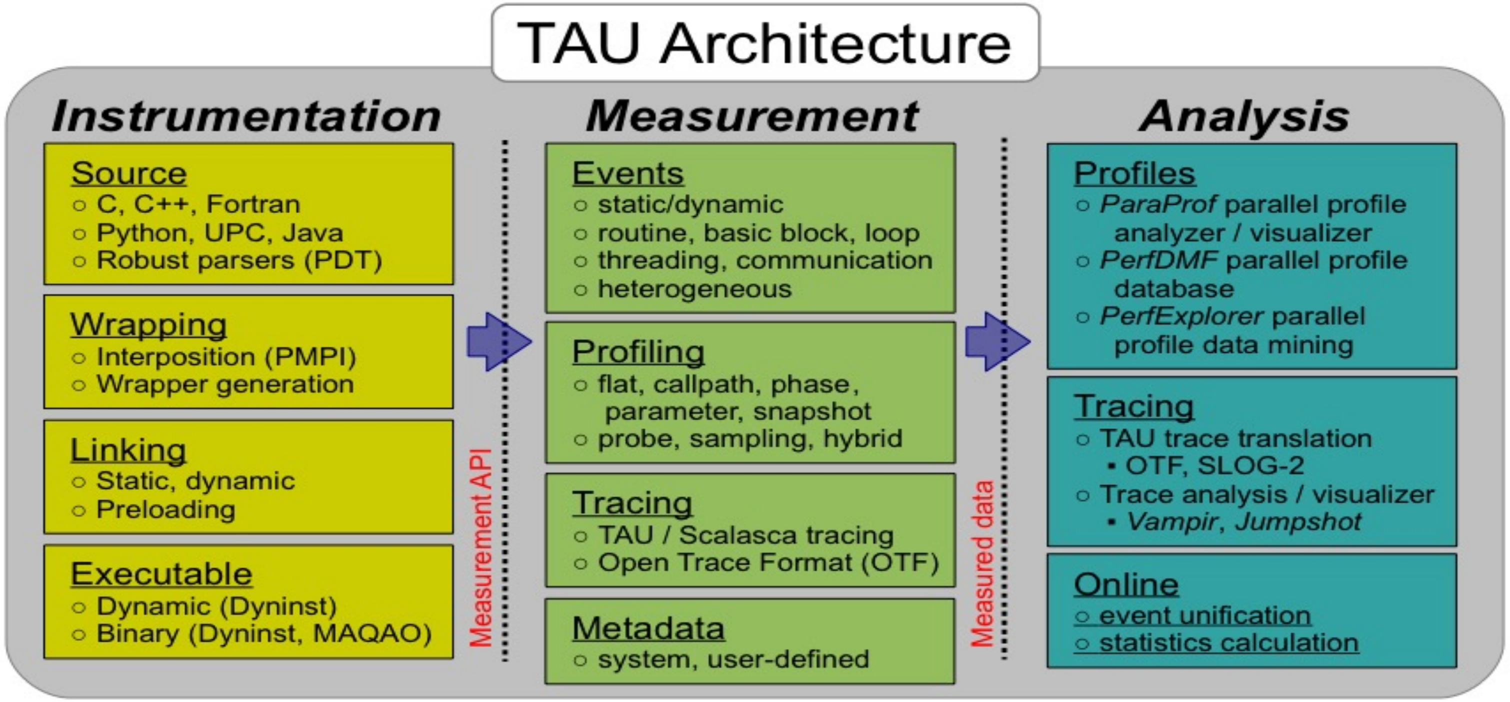
Understanding Application Performance using TAU

- **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?

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, uGNI
- pthreads, OpenMP, OMPT interface, hybrid, other thread models
- GPU, CUDA, OpenCL, Level Zero, ROCm, OpenACC
- Parallel profiling and tracing
- Interfaces with OTF2 and Score-P

Analysis

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

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

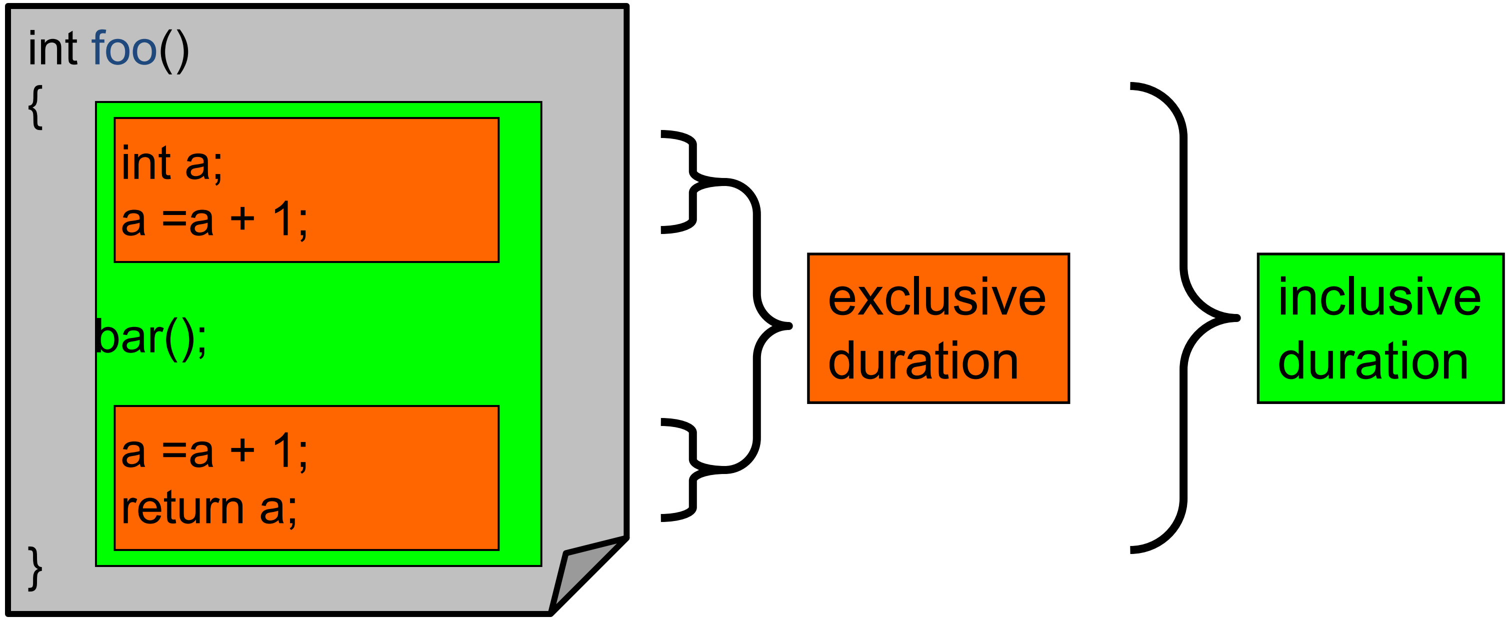
- Interval events (begin/end events)
  - Measures exclusive & inclusive durations between events
  - Metrics monotonically increase
- Atomic events (trigger with data value)
  - Used to capture performance data state
  - Shows extent of variation of triggered values (min/max/mean)

Code events

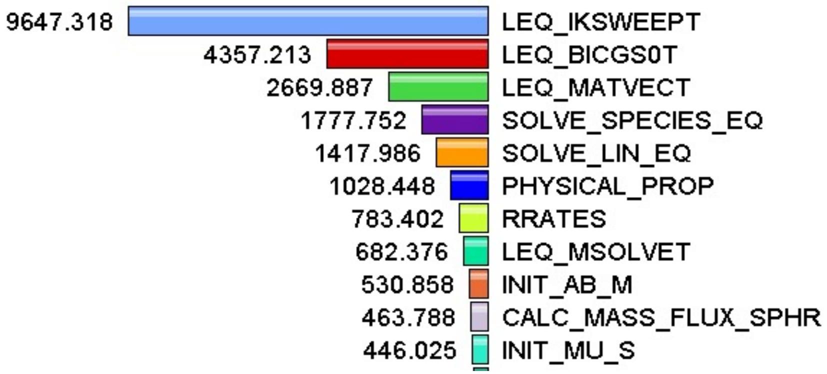
- Routines, classes, templates
- Statement-level blocks, loops

Inclusive and Exclusive Profiles

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

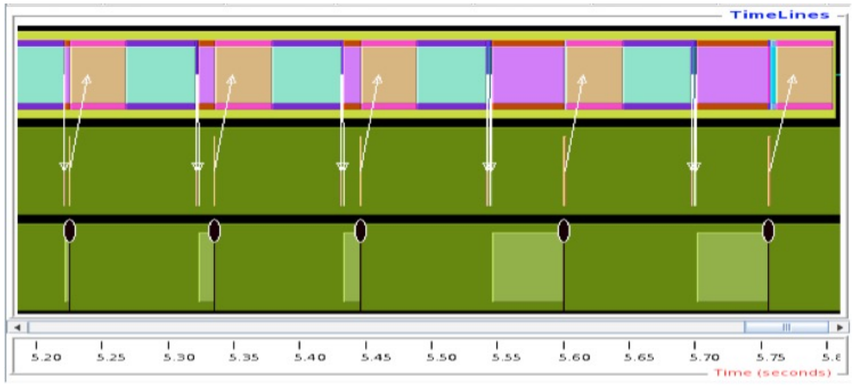


Profiling



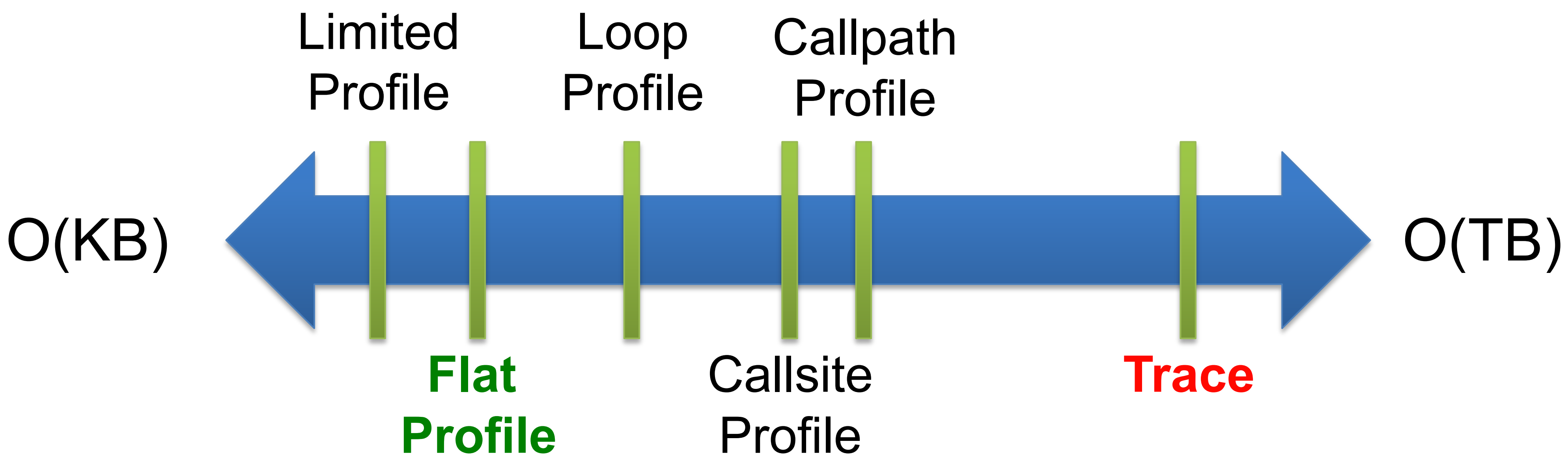
Shows
how much time was
spent in each routine

Tracing

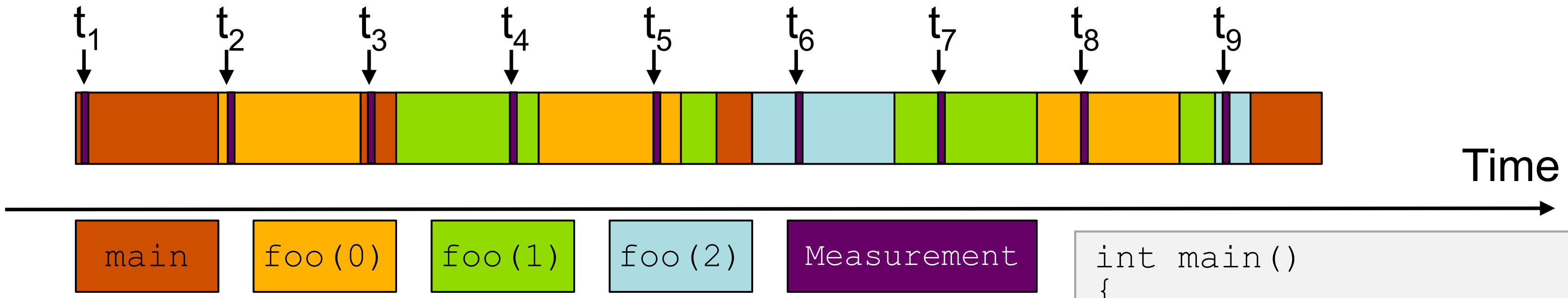


Shows
when events take place
on a timeline

How much data do you want?



Sampling




```
int main()
{
  int i;

  for (i=0; i < 3; i++)
    foo(i);

  return 0;
}

void foo(int i)
{

  if (i > 0)
    foo(i – 1);

}
```

Running program is periodically interrupted to take measurement

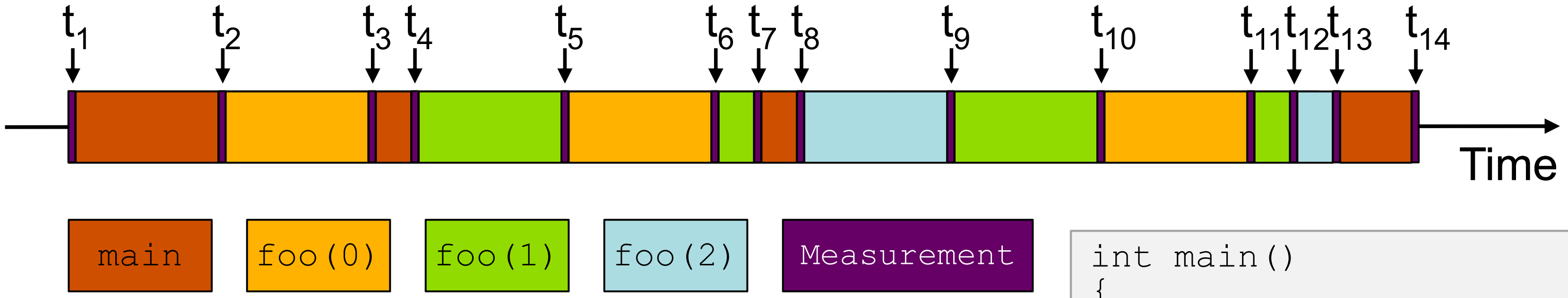
- Timer interrupt, OS signal, or HWC overflow
- Service routine examines return-address stack
- Addresses are mapped to routines using symbol table information

Statistical inference of program behavior

- Not very detailed information on highly volatile metrics
- Requires long-running applications

Works with unmodified executables

Instrumentation



Measurement code is inserted such that every event of interest is captured directly

- Can be done in various ways

Advantage:

- Much more detailed information

Disadvantage:

- Processing of source-code / executable necessary
- Large relative overheads for small functions

```
int main()
{
  int i;
  TAU_START("main");
  for (i=0; i < 3; i++)
    foo(i);
  TAU_STOP("main");
  return 0;
}

void foo(int i)
{
  TAU_START("foo");
  if (i > 0)
    foo(i - 1);
  TAU_STOP("foo");
}
```

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

TAU's Support for Runtime Systems

MPI

- PMPI profiling interface
- MPI_T tools interface using performance and control variables

Pthread

- Captures time spent in routines per thread of execution

OpenMP

- OMPT tools interface to track salient OpenMP runtime events
- Opari source rewriter
- Preloading wrapper OpenMP runtime library when OMPT is not supported

Intel Level Zero

- Captures time spent in kernels on GPUs using oneAPI Level Zero
- Captures time spent in Intel Level Zero runtime calls

OpenACC

- OpenACC instrumentation API
- Track data transfers between host and device (per-variable)
- Track time spent in kernels

TAU's Support for Runtime Systems (contd.)

OpenCL

- OpenCL profiling interface
- Track timings of kernels

CUDA

- Cuda Profiling Tools Interface (CUPTI)
- Track data transfers between host and GPU
- Track access to uniform shared memory between host and GPU

ROCm

- Rocprofiler and Roctracer instrumentation interfaces
- Track data transfers and kernel execution between host and GPU

Kokkos

- Kokkos profiling API
- Push/pop interface for region, kernel execution interface

Python

- Python interpreter instrumentation API
- Tracks Python routine transitions as well as Python to C transitions

Examples of Multi-Level Instrumentation

MPI + OpenMP

- MPI_T + PMPI + OMPT may be used to track MPI and OpenMP

MPI + CUDA

- PMPI + CUPTI interfaces

OpenCL + ROCm

- Rocprofiler + OpenCL instrumentation interfaces

Kokkos + OpenMP

- Kokkos profiling API + OMPT to transparently track events

Kokkos + pthread + MPI

- Kokkos + pthread wrapper interposition library + PMPI layer

Python + CUDA

- Python + CUPTI + pthread profiling interfaces (e.g., Tensorflow, PyTorch)

MPI + OpenCL

- PMPI + OpenCL profiling interfaces

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 Cray systems).

Simplifying the use of TAU!

Uninstrumented code:

- % make
- % mpirun -np 64 ./a.out

With TAU using event-based sampling (EBS):

- % mpirun –np 64 tau_exec –ebs ./a.out
- % paraprof   (GUI)
- % pprof –a | more

NOTE:

- Requires dynamic executables (-dynamic link flag on Cray XC systems).
- Source code should be compiled with –g for access to symbol table.
- Replace srun with mpirun on Attaway or your appropriate launch command.

TAU Execution Command (tau_exec)

Uninstrumented execution

- % mpirun -np 256 ./a.out

Track GPU operations

- % mpirun -np 256 tau_exec –rocm ./a.out
- % mpirun -np 256 tau_exec –l0 ./a.out
- % 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 I/O, and MPI performance (MPI enabled by default)

- % mpirun -np 256 tau_exec -io ./a.out

Track OpenMP and MPI execution (using OMPT for Intel v19)

- % export TAU_OMPT_SUPPORT_LEVEL=full;
  % mpirun -np 256 tau_exec –T ompt,v5,mpi -ompt ./a.out

Track memory operations

- % export TAU_TRACK_MEMORY_LEAKS=1
- % mpirun -np 256 tau_exec –memory_debug ./a.out (bounds check)

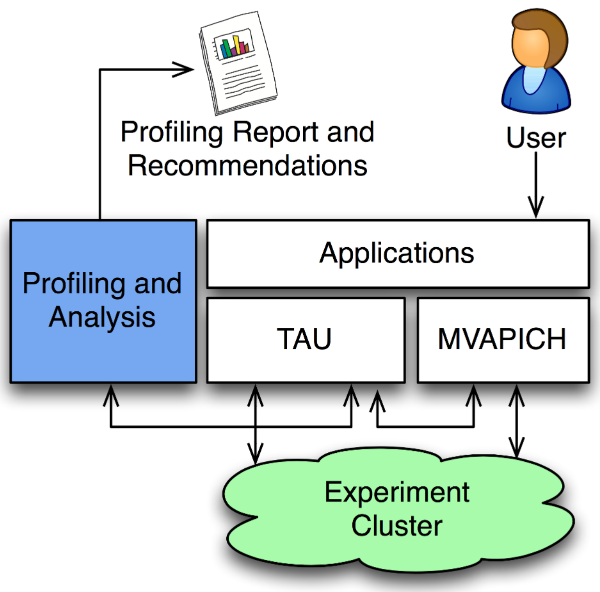
Use event based sampling (compile with –g)

- % mpirun -np 256 tau_exec –ebs ./a.out
- Also export TAU_METRICS=TIME,<PAPI_COUNTER> to use hardware perf. counters
- tau_exec -ebs_resolution=<file | function | line>

Outline

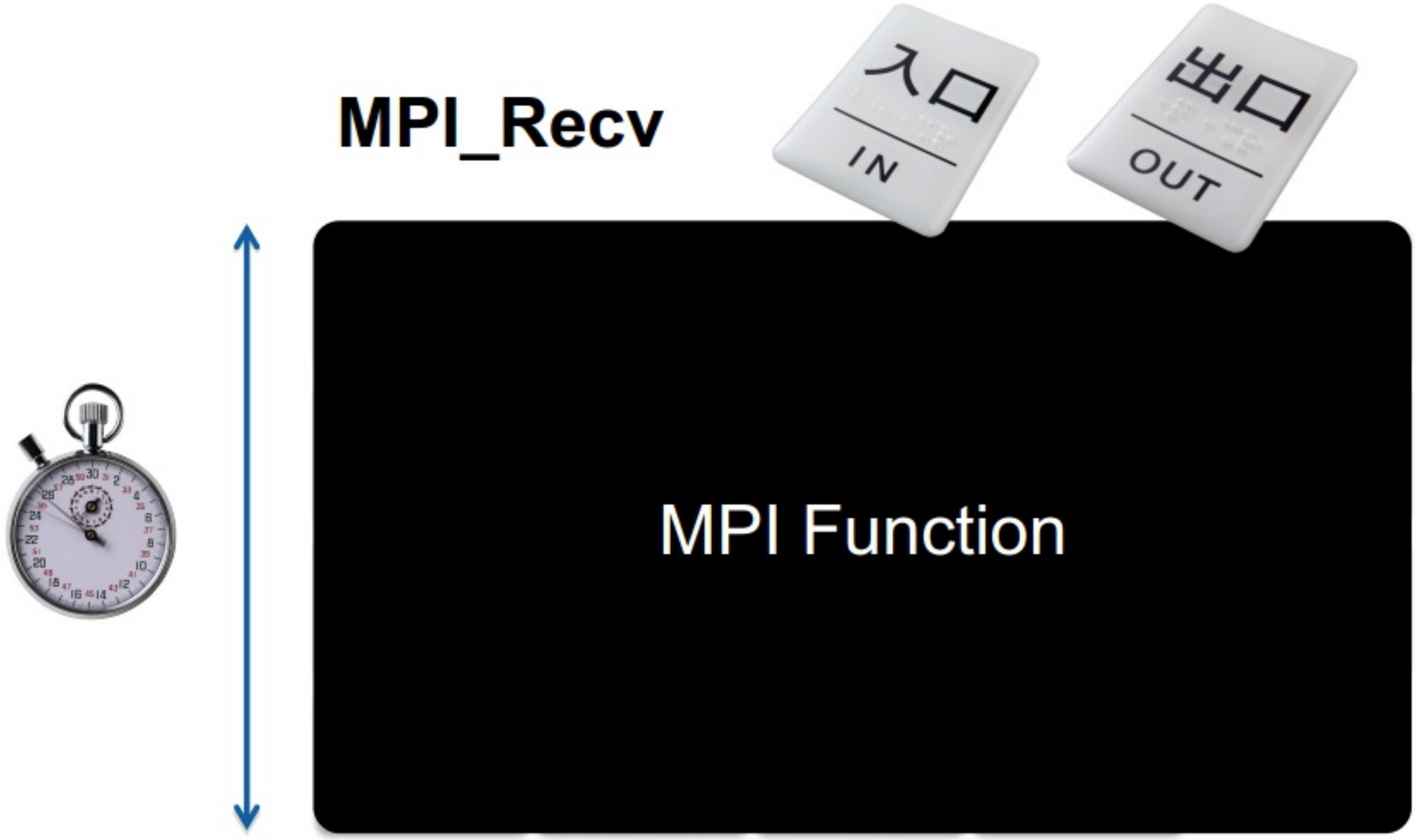
- Introduction
- The MPI Tools Interfaces and Benefits
- Integrating TAU and MVAPICH2 with MPI_T

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

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

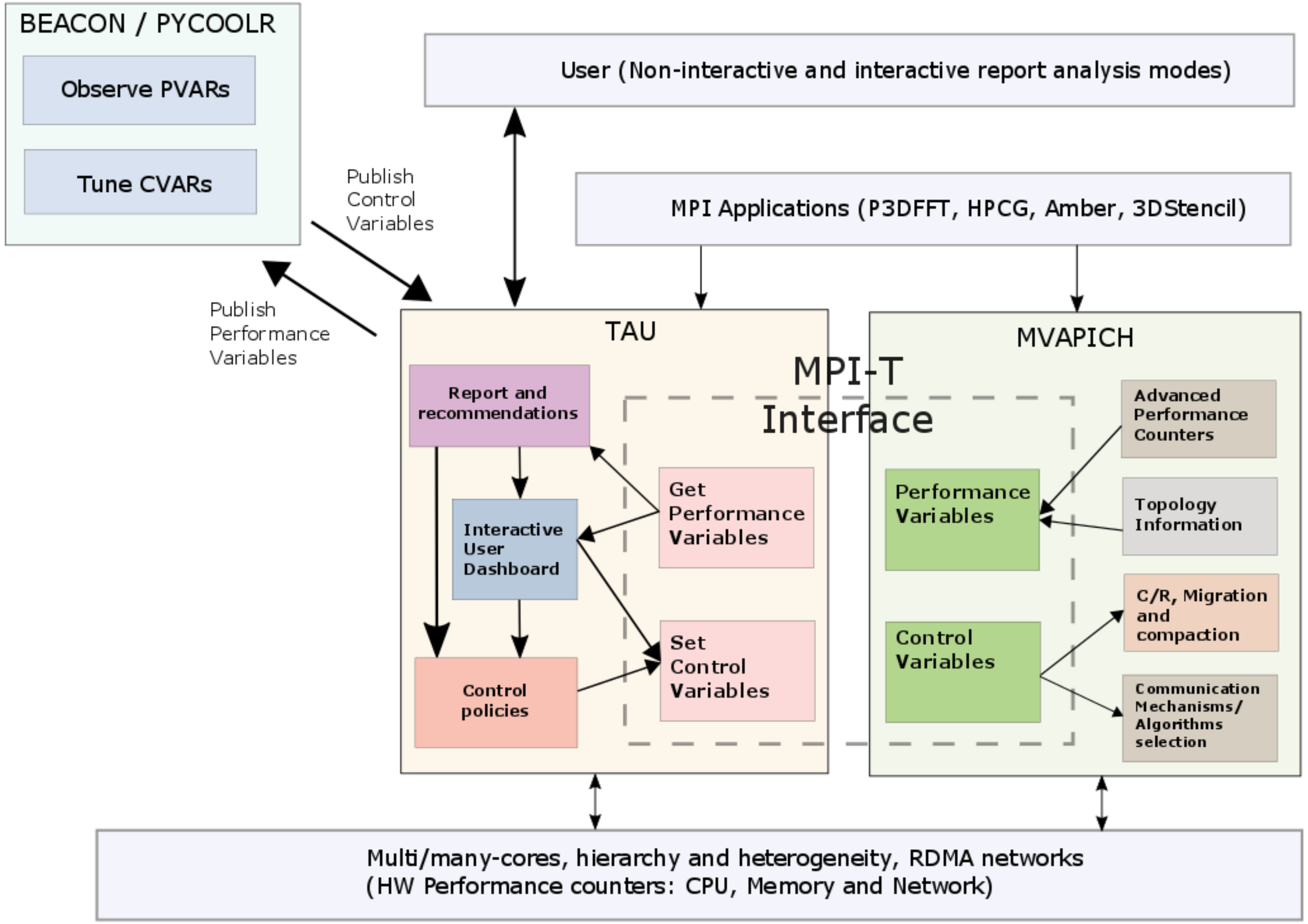
Example Pseudo-code: Optimizing the eager limit dynamically:

```
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

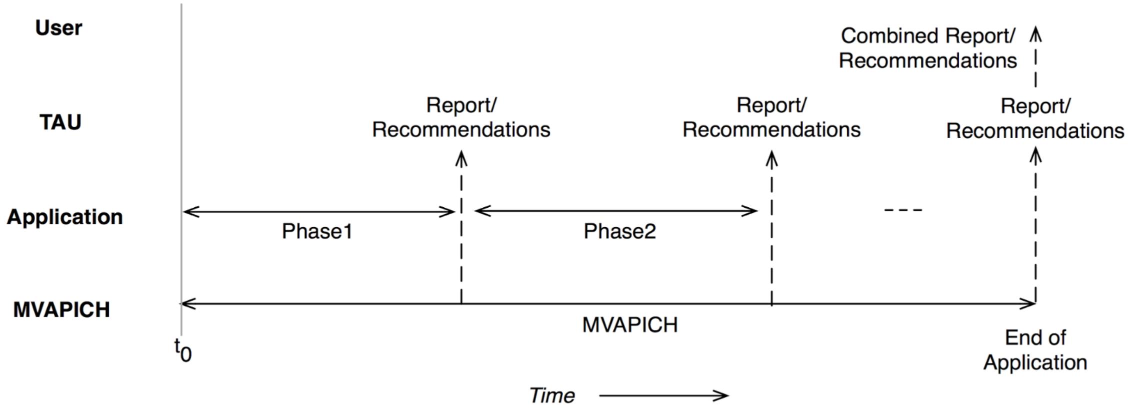
- Introduction
- The MPI Tools Interfaces and Benefits
- Integrating TAU and MVAPICH2 with MPI_T

Integrating TAU with MVAPICH2 through MPI_T Interface

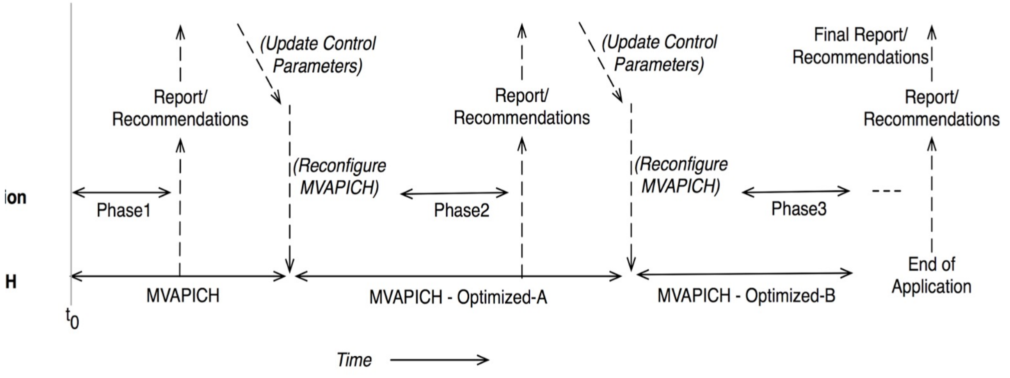


- Enhance existing support for MPI_T in MVAPICH2 to expose a richer set of performance and control variables
- Get and display MPI Performance Variables (PVARs) made available by the runtime in TAU
- Control the runtime's behavior via MPI Control Variables (CVARs)
- Add support to MVAPICH2 and TAU for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

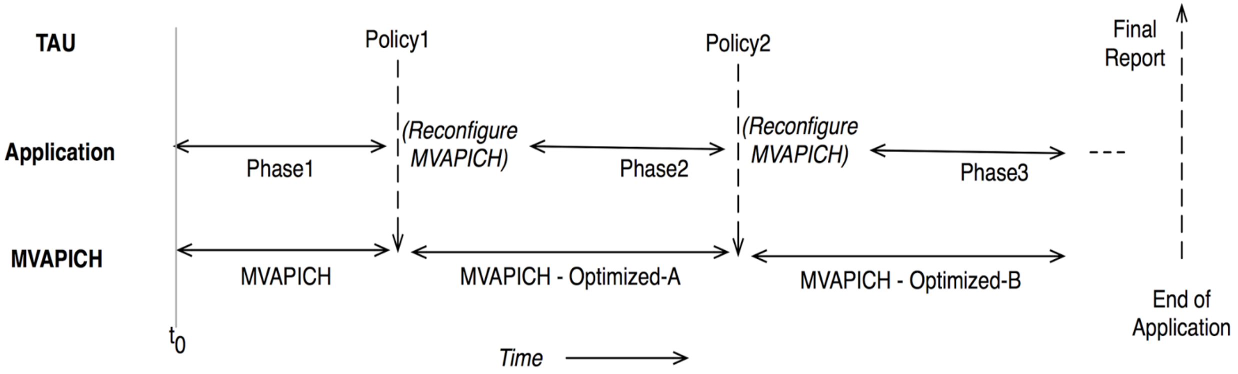
Three Scenarios for Integration



Scenario 1: Non-interactive mode

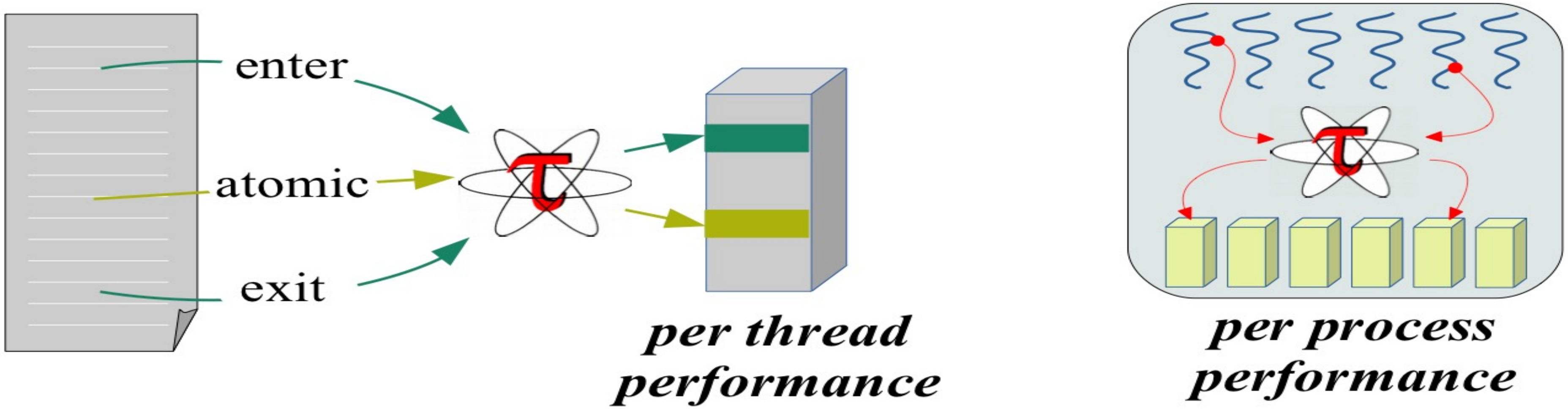


Scenario 2: User-interactive mode



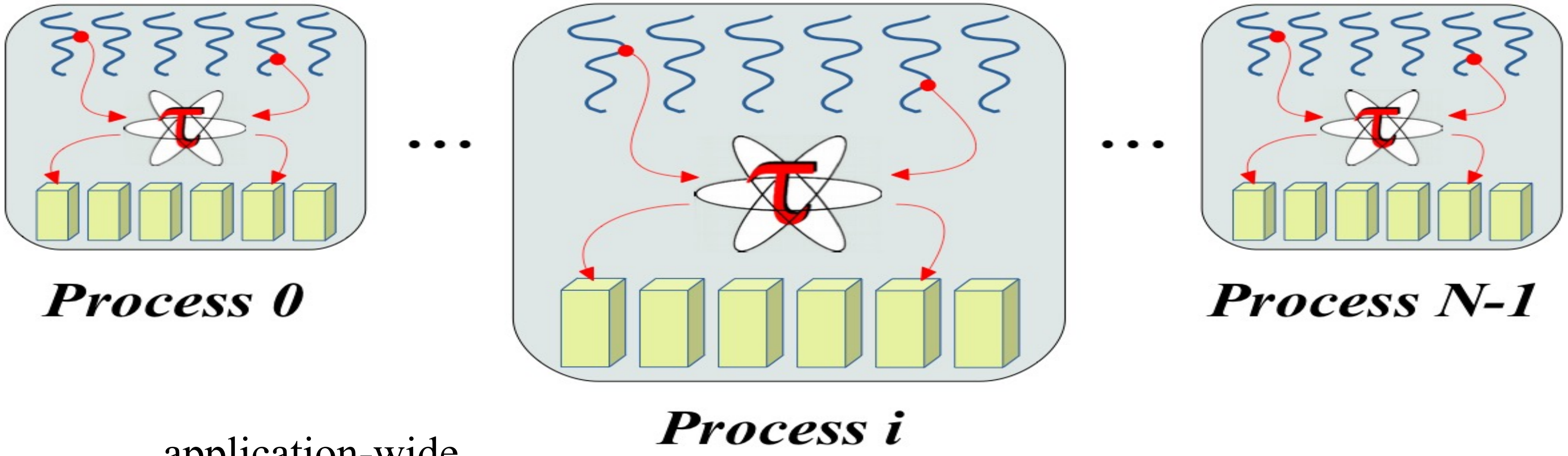
Scenario 3: Policy driven mode

TAU Performance Measurement Model



enter/exit events
are “interval” events

(in shared memory)



application-wide
performance data

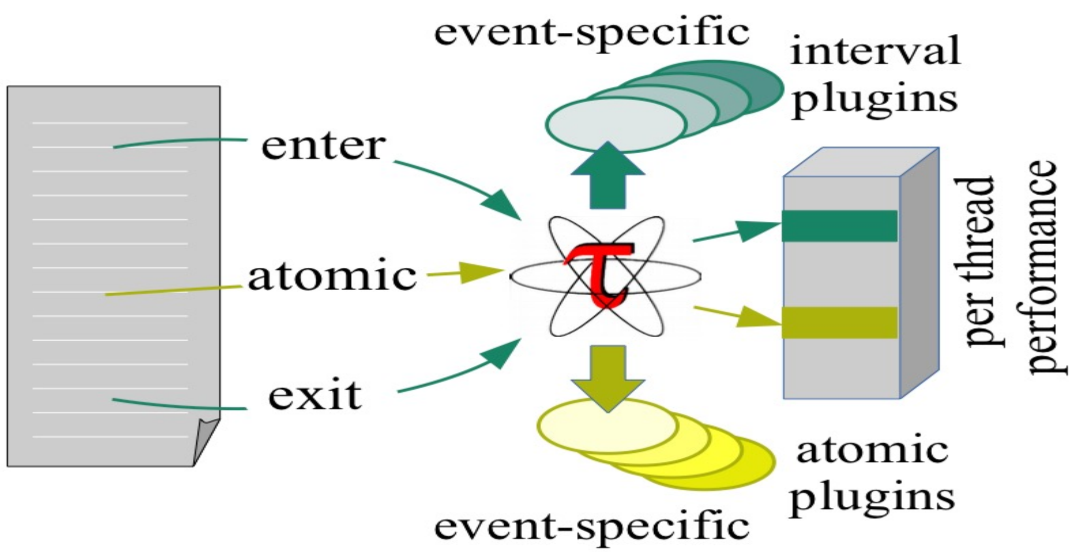
TAU Plugin Architecture

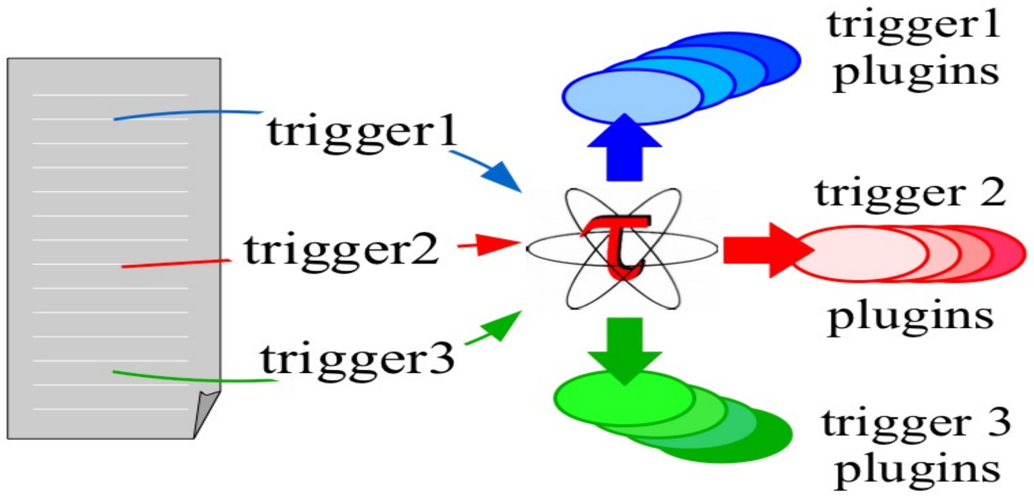
Extend TAU *event* interface for plugins

- Events: *interval, atomic*
- Specialized on event ID
- Synchronous operation

Create TAU interface for *trigger* plugins

- Named trigger
- Pass application data
- Synchronous
- Asynchronous using agent plugin





TAU Plugin Architecture

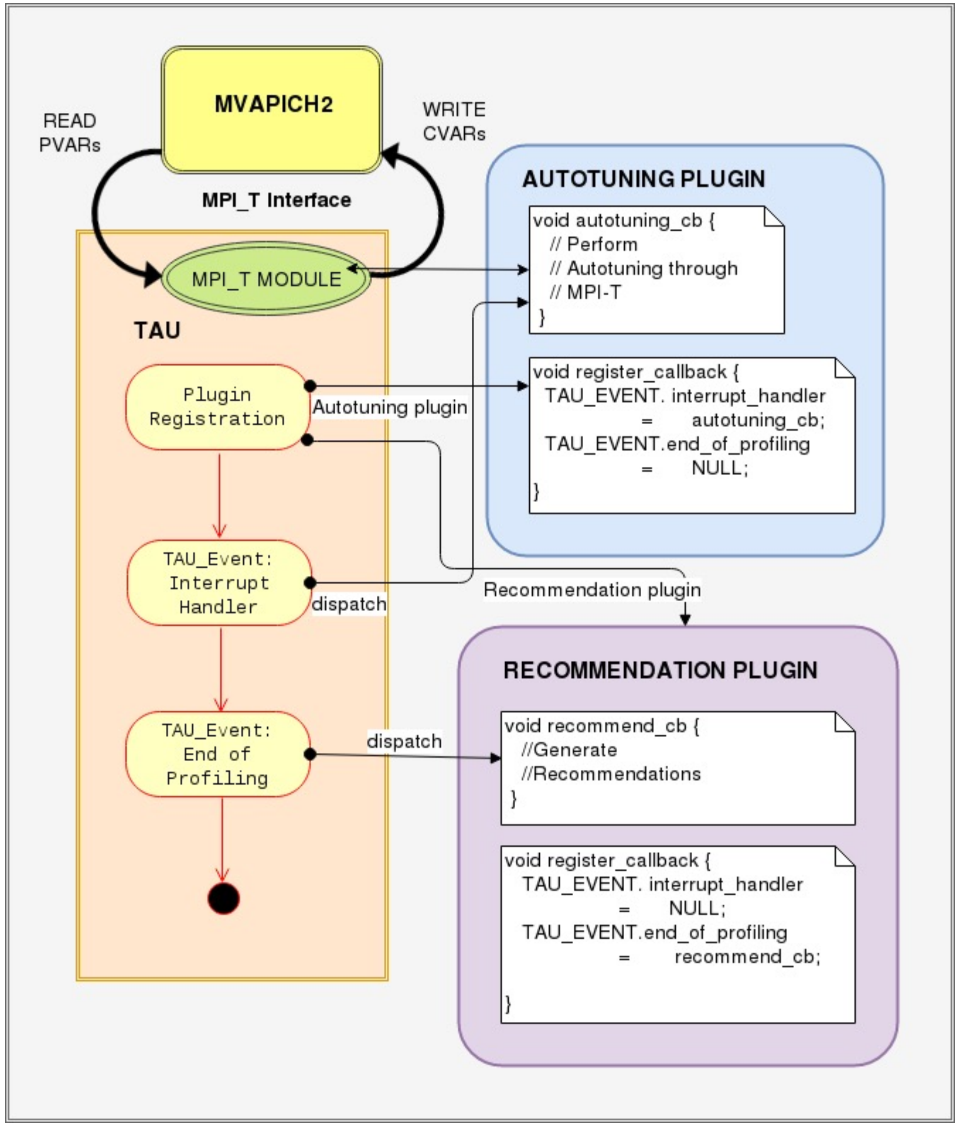
- **Both event and trigger plugins are synchronous**
  - Directly called from the application
  - Execute inline with the application
  - Use an application's thread of execution
- **Consider utilizing a separate thread of execution to perform performance analysis functions**
  - For instance, periodic daemon to sample performace
- **Design an *agent* plugin mechanism**
  - Create an execution thread to execute plugin
  - Register plugin with this execution thread

TAU Plugin Architecture

- **Parallel performance systems do not typically do runtime analytics when making measurements**
- **Want to extend a performance system with additional analytics functionality WITHOUT building it directly into the performance system**
- **Apply a plugin architecture approach**
  - Develop analytics plugins (common, application)
  - Register (load) them with the performance system
- **Plugins have access to performance data state**
- **Plugins can utilize the parallel execution context**

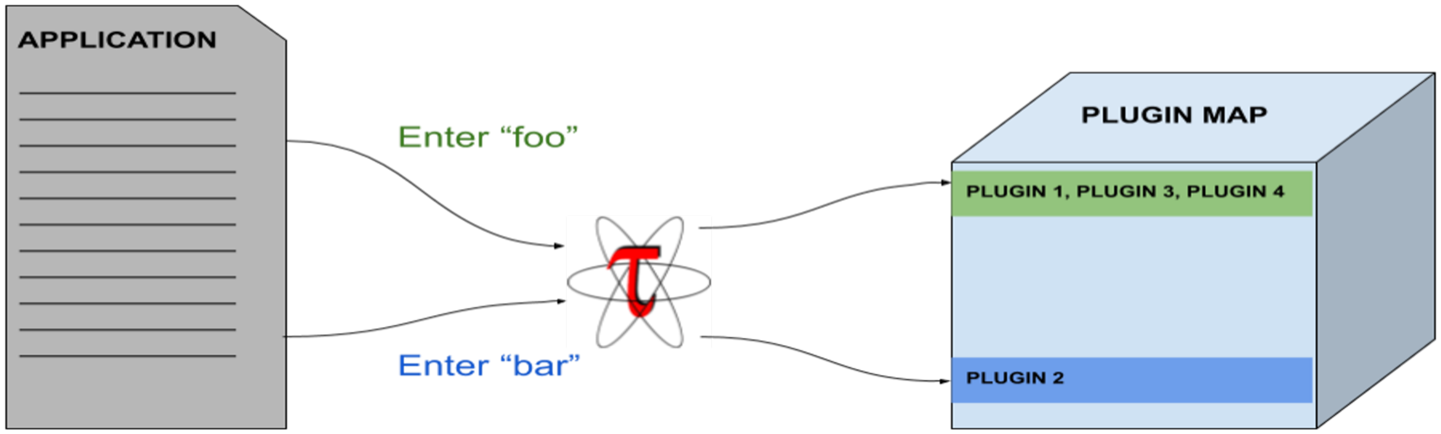
Plugin-based Infrastructure for Non-Interactive Tuning

- **TAU supports a *fully-customizable* plugin infrastructure based on callback event handler registration for salient states inside TAU:**
  - Function Registration / Entry / Exit
  - Phase Entry / Exit
  - Atomic Event Registration / Trigger
  - Init / Finalize Profiling
  - Interrupt Handler
  - *MPI_T*
- **Application can define its own "trigger" states and associated plugins**
  - Pass arbitrary data to trigger state plugins



TAU Customization

- **TAU states can be *named* or *generic***
- **TAU distinguishes named states in a way that allows for separation of occurrence of a state from the action associated with it**
  - Function entry for “foo” and “bar” represent distinguishable states in TAU
- **TAU maintains an internal map of a list of plugins associated with each state**



TAU Runtime Control of Plugin

- **TAU defines a plugin API to deliver access control to the internal plugin map**
- **User can specify a regular expression to control plugins executed for a class of named states at runtime**
  - Access to map on a process is serialized: application is expected to access map through main thread

TAU Phase Based Recommendations

- **MiniAMR: Benefits from hardware offloading using SHArP hardware offload protocol supported by MVAPICH2 for MPI_Allreduce operation**
- **Recommendation Plugin:**
  - Registers callback for *“Phase Exit”* event
  - Monitors message size through PMPI interface
  - If message size is low and execution time inside MPI_Allreduce is significant, a recommendation is generated on ParaProf (TAU’s GUI) for the user to set the CVAR enabling SHArP

TAU Per-Phase Recommendations in ParaProf

Metadata for n,c,t 7,0,0

Name	Value
TAU MEMDBG PROTECT BELOW	off
TAU MEMDBG PROTECT FREE	off
TAU MPI T ENABLE USER TUNING POLICY	off
TAU OPENMP RUNTIME	on
TAU OPENMP RUNTIME EVENTS	on
TAU OPENMP RUNTIME STATES	off
TAU OUTPUT CUDA CSV	off
TAU PAPI MULTIPLEXING	off
TAU PROFILE	on
TAU PROFILE FORMAT	profile
TAU RECOMMENDATION PHASE ALLOCATE	MPI T RECOMMEND SHARP USAGE: No perfomance benefit foreseen with SHArP usage
TAU RECOMMENDATION PHASE DEALLOCATE	MPI T RECOMMEND SHARP USAGE: You could see potential improvement in performance by enabling MV2 ENABLE SHARP in MVAPICH version 2.3a and above
TAU RECOMMENDATION PHASE DRIVER	MPI T RECOMMEND SHARP USAGE: You could see potential improvement in performance by enabling MV2 ENABLE SHARP in MVAPICH version 2.3a and above
TAU RECOMMENDATION PHASE INIT	MPI T RECOMMEND SHARP USAGE: No perfomance benefit foreseen with SHArP usage
TAU RECOMMENDATION PHASE PROFILE	MPI T RECOMMEND SHARP USAGE: You could see potential improvement in performance by enabling MV2 ENABLE SHARP in MVAPICH version 2.3a and above
TAU REGION ADDRESSES	off
TAU SAMPLING	off
TAU SHOW MEMORY FUNCTIONS	off
TAU SIGNALS GDB	off
TAU THROTTLE	on
TAU THROTTLE NUMCALLS	100000
TAU THROTTLE PERCALL	10
TAU TRACE	off
TAU TRACE FORMAT	tau
TAU TRACK CUDA CDP	off
TAU TRACK CUDA ENV	off
TAU TRACK CUDA INSTRUCTIONS	
TAU TRACK CUDA SASS	off
TAU TRACK HEADROOM	off
TAU TRACK HEAP	off
TAU TRACK IO PARAMS	off
TAU TRACK MEMORY FOOTPRINT	off

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:
      - % 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,mpit* ./a.out

PVARs Exposed by MVAPICH2

TAU: ParaProf Manager

File Options Help

- Applications
  - Standard Applications
    - Default App
      - Default Exp
        - lulesh.ppk
          - TIME
  - Default (jdbc:h2:/home

TrialField	Value
MPI_T PVAR[0]: mem_allocated	Current level of allocated memory within the MPI library
MPI_T PVAR[10]: mv2_num_2level_comm_success	Number of successful 2-level comm creations
MPI_T PVAR[11]: mv2_num_shmem_coll_calls	Number of times MV2 shared-memory collective calls were invoked
MPI_T PVAR[12]: mpit_progress_poll	CH3 RDMA progress engine polling count
MPI_T PVAR[13]: mv2_smp_read_progress_poll	CH3 SMP read progress engine polling count
MPI_T PVAR[14]: mv2_smp_write_progress_poll	CH3 SMP write progress engine polling count
MPI_T PVAR[15]: mv2_smp_read_progress_poll_success	Unsucessful CH3 SMP read progress engine polling count
MPI_T PVAR[16]: mv2_smp_write_progress_poll_succ...	Unsucessful CH3 SMP write progress engine polling count
MPI_T PVAR[17]: rdma_ud_retransmissions	CH3 RDMA UD retransmission count
MPI_T PVAR[18]: mv2_coll_bcast_binomial	Number of times MV2 binomial bcast algorithm was invoked
MPI_T PVAR[19]: mv2_coll_bcast_scatter_doubling_all...	Number of times MV2 scatter+double allgather bcast algorithm was invoked
MPI_T PVAR[1]: mem_allocated	Maximum level of memory ever allocated within the MPI library
MPI_T PVAR[20]: mv2_coll_bcast_scatter_ring_allgather	Number of times MV2 scatter+ring allgather bcast algorithm was invoked
MPI_T PVAR[21]: mv2_coll_bcast_scatter_ring_allgath...	Number of times MV2 scatter+ring allgather shm bcast algorithm was invoked
MPI_T PVAR[22]: mv2_coll_bcast_shmem	Number of times MV2 shmem bcast algorithm was invoked
MPI_T PVAR[23]: mv2_coll_bcast_knomial_internode	Number of times MV2 knomial internode bcast algorithm was invoked
MPI_T PVAR[24]: mv2_coll_bcast_knomial_intranode	Number of times MV2 knomial intranode bcast algorithm was invoked
MPI_T PVAR[25]: mv2_coll_bcast_mcast_internode	Number of times MV2 mcast internode bcast algorithm was invoked
MPI_T PVAR[26]: mv2_coll_bcast_pipelined	Number of times MV2 pipelined bcast algorithm was invoked
MPI_T PVAR[27]: mv2_coll_alltoall_inplace	Number of times MV2 in-place alltoall algorithm was invoked
MPI_T PVAR[28]: mv2_coll_alltoall_bruck	Number of times MV2 brucks alltoall algorithm was invoked
MPI_T PVAR[29]: mv2_coll_alltoall_rd	Number of times MV2 recursive-doubling alltoall algorithm was invoked
MPI_T PVAR[2]: num_malloc_calls	Number of MPIT_malloc calls
MPI_T PVAR[30]: mv2_coll_alltoall_sd	Number of times MV2 scatter-destination alltoall algorithm was invoked
MPI_T PVAR[31]: mv2_coll_alltoall_pw	Number of times MV2 pairwise alltoall algorithm was invoked
MPI_T PVAR[32]: mpit_alltoallv_mv2_pw	Number of times MV2 pairwise alltoallv algorithm was invoked
MPI_T PVAR[33]: mv2_coll_allreduce_shm_rd	Number of times MV2 shm rd allreduce algorithm was invoked
MPI_T PVAR[34]: mv2_coll_allreduce_shm_rs	Number of times MV2 shm rs allreduce algorithm was invoked
MPI_T PVAR[35]: mv2_coll_allreduce_shm_intra	Number of times MV2 shm intra allreduce algorithm was invoked
MPI_T PVAR[36]: mv2_coll_allreduce_intra_p2p	Number of times MV2 intra p2p allreduce algorithm was invoked
MPI_T PVAR[37]: mv2_coll_allreduce_2lvl	Number of times MV2 two-level allreduce algorithm was invoked
MPI_T PVAR[38]: mv2_coll_allreduce_shmem	Number of times MV2 shmem allreduce algorithm was invoked
MPI_T PVAR[39]: mv2_coll_allreduce_mcast	Number of times MV2 multicast-based allreduce algorithm was invoked
MPI_T PVAR[3]: num_calloc_calls	Number of MPIT_calloc calls
MPI_T PVAR[40]: mv2_reg_cache_hits	Number of registration cache hits
MPI_T PVAR[41]: mv2_reg_cache_misses	Number of registration cache misses
MPI_T PVAR[42]: mv2_vbuf_allocated	Number of VBUFs allocated
MPI_T PVAR[43]: mv2_vbuf_allocated_array	Number of VBUFs allocated
MPI_T PVAR[44]: mv2_vbuf_freed	Number of VBUFs freed
MPI_T PVAR[45]: mv2_ud_vbuf_allocated	Number of UD VBUFs allocated
MPI_T PVAR[46]: mv2_ud_vbuf_freed	Number of UD VBUFs freed
MPI_T PVAR[47]: mv2_vbuf_free_attempts	Number of time we attempted to free VBUFs
MPI_T PVAR[48]: mv2_vbuf_free_attempt_success_time	Average time for number of times we sucessfully freed VBUFs
MPI_T PVAR[49]: mv2_vbuf_free_attempt_success_time	Average time for number of times we sucessfully freed VBUFs
MPI_T PVAR[4]: num_memalign_calls	Number of MPIT_memalign calls
MPI_T PVAR[50]: mv2_vbuf_allocate_time	Average time for number of times we allocated VBUFs
MPI_T PVAR[51]: mv2_vbuf_allocate_time	Average time for number of times we allocated VBUFs

CVARs Exposed by MVAPICH2

TAU: ParaProf Manager

File Options Help

- Applications
  - Standard Applications
    - Default App
      - Default Exp
        - lulesh.ppk
          - TIME
  - Default (jdbc:h2:/home

TrialField	Value
Local Time	2016-08-16T10:11:04-07:00
MPI Processor Name	cerberus.nic.uoregon.edu
MPIR_CVAR_ABORT_ON_LEAKED_HANDLES	If true, MPI will call MPI_Abort at MPI_Finalize if any MPI object handles have been leaked. For example,...
MPIR_CVAR_ALLGATHERV_PIPELINE_MSG_SIZE	The smallest message size that will be used for the pipelined, large-message, ring algorithm in the MPI_...
MPIR_CVAR_ALLGATHER_LONG_MSG_SIZE	For MPI_Allgather and MPI_Allgatherv, the long message algorithm will be used if the send buffer size is ...
MPIR_CVAR_ALLGATHER_SHORT_MSG_SIZE	For MPI_Allgather and MPI_Allgatherv, the short message algorithm will be used if the send buffer size is...
MPIR_CVAR_ALLREDUCE_SHORT_MSG_SIZE	the short message algorithm will be used if the send buffer size is <= this value (in bytes)
MPIR_CVAR_ALLTOALL_MEDIUM_MSG_SIZE	the medium message algorithm will be used if the per-destination message size (sendcount*size(sendtyp...
MPIR_CVAR_ALLTOALL_SHORT_MSG_SIZE	the short message algorithm will be used if the per-destination message size (sendcount*size(sendtype)) ...
MPIR_CVAR_ALLTOALL_THROTTLE	max no. of irecvs/isends posted at a time in some alltoall algorithms. Setting it to 0 causes all irecvs/isen...
MPIR_CVAR_ASYNC_PROGRESS	If set to true, MPICH will initiate an additional thread to make asynchronous progress on all communicati...
MPIR_CVAR_BCAST_LONG_MSG_SIZE	Let's define short messages as messages with size < MPIR_CVAR_BCAST_SHORT_MSG_SIZE, and mediu...
MPIR_CVAR_BCAST_MIN_PROCS	Let's define short messages as messages with size < MPIR_CVAR_BCAST_SHORT_MSG_SIZE, and mediu...
MPIR_CVAR_BCAST_SHORT_MSG_SIZE	Let's define short messages as messages with size < MPIR_CVAR_BCAST_SHORT_MSG_SIZE, and mediu...
MPIR_CVAR_CH3_EAGER_MAX_MSG_SIZE	This cvar controls the message size at which CH3 switches from eager to rendezvous mode.
MPIR_CVAR_CH3_ENABLE_HCOLL	If true, enable HCOLL collectives.
MPIR_CVAR_CH3_INTERFACE_HOSTNAME	If non-NULL, this cvar specifies the IP address that other processes should use when connecting to this pr...
MPIR_CVAR_CH3_NOLOCAL	If true, force all processes to operate as though all processes are located on another node. For example,...
MPIR_CVAR_CH3_ODD_EVEN_CLIQUES	If true, odd procs on a node are seen as local to each other, and even procs on a node are seen as local t...
MPIR_CVAR_CH3_PORT_RANGE	The MPIR_CVAR_CH3_PORT_RANGE environment variable allows you to specify the range of TCP ports ...
MPIR_CVAR_CH3_RMA_ACC_IMMED	Use the immediate accumulate optimization
MPIR_CVAR_CH3_RMA_GC_NUM_COMPLETED	Threshold for the number of completed requests the runtime finds before it stops trying to find more co...
MPIR_CVAR_CH3_RMA_GC_NUM_TESTED	Threshold for the number of RMA requests the runtime tests before it stops trying to check more reques...
MPIR_CVAR_CH3_RMA_LOCK_IMMED	Issue a request for the passive target RMA lock immediately. Default behavior is to defer the lock reque...
MPIR_CVAR_CH3_RMA_MERGE_LOCK_OP_UNLOCK	Enable/disable an optimization that merges lock, op, and unlock messages, for single-operation passive ta...
MPIR_CVAR_CH3_RMA_NREQUEST_NEW_THRESHOLD	Threshold for the number of new requests since the last attempt to complete pending requests. Higher ...
MPIR_CVAR_CH3_RMA_NREQUEST_THRESHOLD	Threshold at which the RMA implementation attempts to complete requests while completing RMA oper...
MPIR_CVAR_CHOP_ERROR_STACK	If >0, truncate error stack output lines this many characters wide. If 0, do not truncate, and if <0 use a ...
MPIR_CVAR_COLL_ALIAS_CHECK	Enable checking of aliasing in collective operations
MPIR_CVAR_COMM_SPLIT_USE_QSORT	Use qsort(3) in the implementation of MPI_Comm_split instead of bubble sort.
MPIR_CVAR_CTXID_EAGER_SIZE	The MPIR_CVAR_CTXID_EAGER_SIZE environment variable allows you to specify how many words in th...
MPIR_CVAR_DEBUG_HOLD	If true, causes processes to wait in MPI_Init and MPI_Initthread for a debugger to be attached. Once the ...
MPIR_CVAR_DEFAULT_THREAD_LEVEL	Sets the default thread level to use when using MPI_INIT.
MPIR_CVAR_DUMP_PROVIDERS	If true, dump provider information at init
MPIR_CVAR_ENABLE_COLL_FT_RET	DEPRECATED! Will be removed in MPICH-3.2 Collectives called on a communicator with a failed process...
MPIR_CVAR_ENABLE_SMP_ALLREDUCE	Enable SMP aware allreduce.
MPIR_CVAR_ENABLE_SMP_BARRIER	Enable SMP aware barrier.
MPIR_CVAR_ENABLE_SMP_BCAST	Enable SMP aware broadcast (See also: MPIR_CVAR_MAX_SMP_BCAST_MSG_SIZE)
MPIR_CVAR_ENABLE_SMP_COLLECTIVES	Enable SMP aware collective communication.
MPIR_CVAR_ENABLE_SMP_REDUCE	Enable SMP aware reduce.
MPIR_CVAR_ERROR_CHECKING	If true, perform checks for errors, typically to verify valid inputs to MPI routines. Only effective when M...
MPIR_CVAR_GATHERV_INTER_SSEND_MIN_PROCS	Use Ssend (synchronous send) for intercommunicator MPI_Gatherv if the "group B" size is >= this value....
MPIR_CVAR_GATHER_INTER_SHORT_MSG_SIZE	use the short message algorithm for intercommunicator MPI_Gather if the send buffer size is < this value...
MPIR_CVAR_GATHER_VSMALL_MSG_SIZE	use a temporary buffer for intracommunicator MPI_Gather if the send buffer size is < this value (in bytes...
MPIR_CVAR_IBA_EAGER_THRESHOLD	0 (old) -> 204800 (new), This set the switch point between eager and rendezvous protocol
MPIR_CVAR_MAX_INLINE_SIZE	This set the maximum inline size for data transfer
MPIR_CVAR_MAX_SMP_ALLREDUCE_MSG_SIZE	Maximum message size for which SMP-aware allreduce is used. A value of '0' uses SMP-aware allreduce ...

Using MVAPICH2 and TAU with Multiple CVARs

- 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

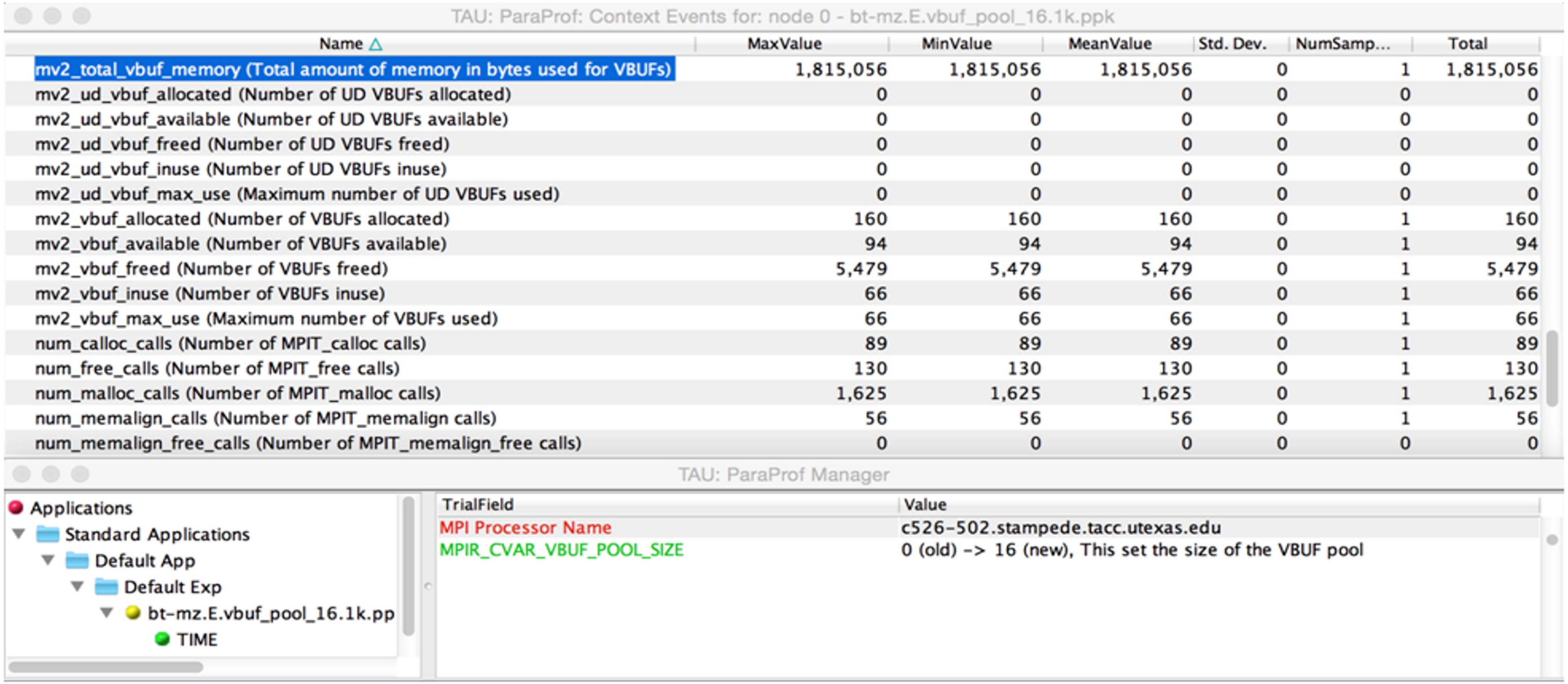
  % paraprof

VBUF usage without CVARs

TAU: ParaProf: Context Events for: node 0 - mpit_withoutcvar_bt.C.1k.ppk

Name △	MaxValue	MinValue	MeanValue	Std. Dev.	NumSamples	Total
mv2_total_vbuf_memory (Total amount of memory in bytes used for VBUFs)	3,313,056	3,313,056	3,313,056	0	1	3,313,056
mv2_ud_vbuf_allocated (Number of UD VBUFs allocated)	0	0	0	0	0	0
mv2_ud_vbuf_available (Number of UD VBUFs available)	0	0	0	0	0	0
mv2_ud_vbuf_freed (Number of UD VBUFs freed)	0	0	0	0	0	0
mv2_ud_vbuf_inuse (Number of UD VBUFs inuse)	0	0	0	0	0	0
mv2_ud_vbuf_max_use (Maximum number of UD VBUFs used)	0	0	0	0	0	0
mv2_vbuf_allocated (Number of VBUFs allocated)	320	320	320	0	1	320
mv2_vbuf_available (Number of VBUFs available)	255	255	255	0	1	255
mv2_vbuf_freed (Number of VBUFs freed)	25,545	25,545	25,545	0	1	25,545
mv2_vbuf_inuse (Number of VBUFs inuse)	65	65	65	0	1	65
mv2_vbuf_max_use (Maximum number of VBUFs used)	65	65	65	0	1	65
num_calloc_calls (Number of MPIT_calloc calls)	89	89	89	0	1	89
num_free_calls (Number of MPIT_free calls)	47,801	47,801	47,801	0	1	47,801
num_malloc_calls (Number of MPIT_malloc calls)	49,258	49,258	49,258	0	1	49,258
num_memalign_calls (Number of MPIT_memalign calls)	34	34	34	0	1	34
num_memalign_free_calls (Number of MPIT_memalign_free calls)	0	0	0	0	0	0

VBUF usage with CVARs

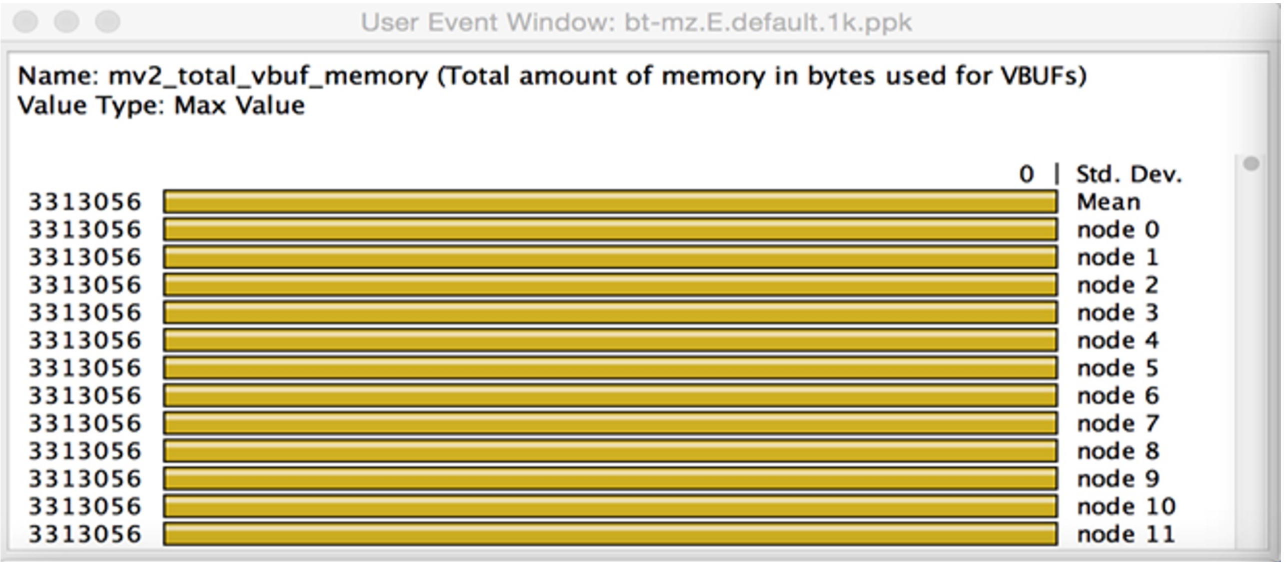


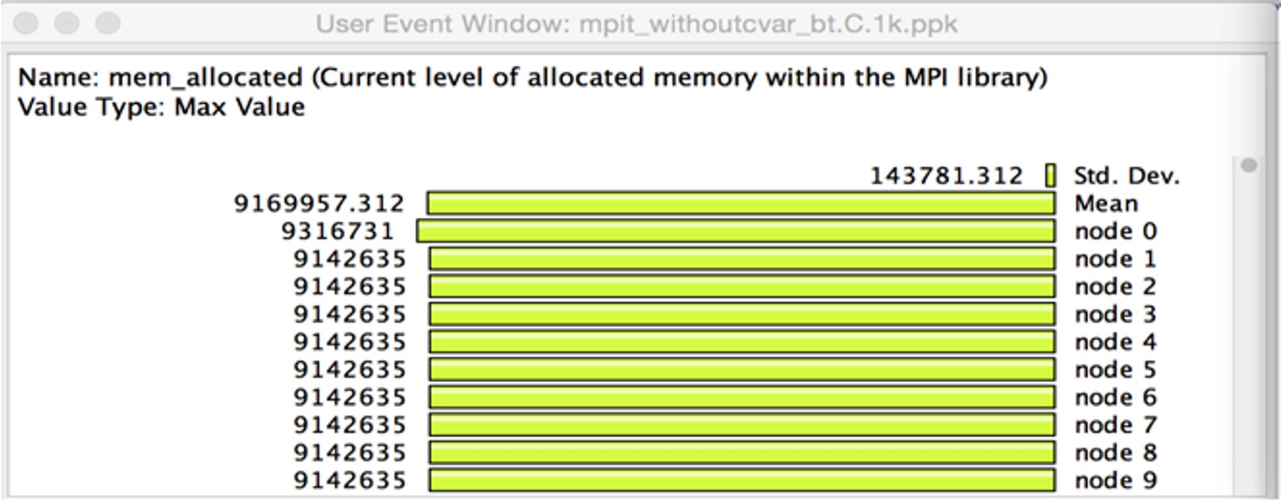
Name △	MaxValue	MinValue	MeanValue	Std. Dev.	NumSamp...	Total
mv2_total_vbuf_memory (Total amount of memory in bytes used for VBUFs)	1,815,056	1,815,056	1,815,056	0	1	1,815,056
mv2_ud_vbuf_allocated (Number of UD VBUFs allocated)	0	0	0	0	0	0
mv2_ud_vbuf_available (Number of UD VBUFs available)	0	0	0	0	0	0
mv2_ud_vbuf_freed (Number of UD VBUFs freed)	0	0	0	0	0	0
mv2_ud_vbuf_inuse (Number of UD VBUFs inuse)	0	0	0	0	0	0
mv2_ud_vbuf_max_use (Maximum number of UD VBUFs used)	0	0	0	0	0	0
mv2_vbuf_allocated (Number of VBUFs allocated)	160	160	160	0	1	160
mv2_vbuf_available (Number of VBUFs available)	94	94	94	0	1	94
mv2_vbuf_freed (Number of VBUFs freed)	5,479	5,479	5,479	0	1	5,479
mv2_vbuf_inuse (Number of VBUFs inuse)	66	66	66	0	1	66
mv2_vbuf_max_use (Maximum number of VBUFs used)	66	66	66	0	1	66
num_calloc_calls (Number of MPIT_calloc calls)	89	89	89	0	1	89
num_free_calls (Number of MPIT_free calls)	130	130	130	0	1	130
num_malloc_calls (Number of MPIT_malloc calls)	1,625	1,625	1,625	0	1	1,625
num_memalign_calls (Number of MPIT_memalign calls)	56	56	56	0	1	56
num_memalign_free_calls (Number of MPIT_memalign_free calls)	0	0	0	0	0	0

TrialField	Value
MPI Processor Name	c526-502.stampede.tacc.utexas.edu
MPIR_CVAR_VBUF_POOL_SIZE	0 (old) -> 16 (new), This set the size of the VBUF pool

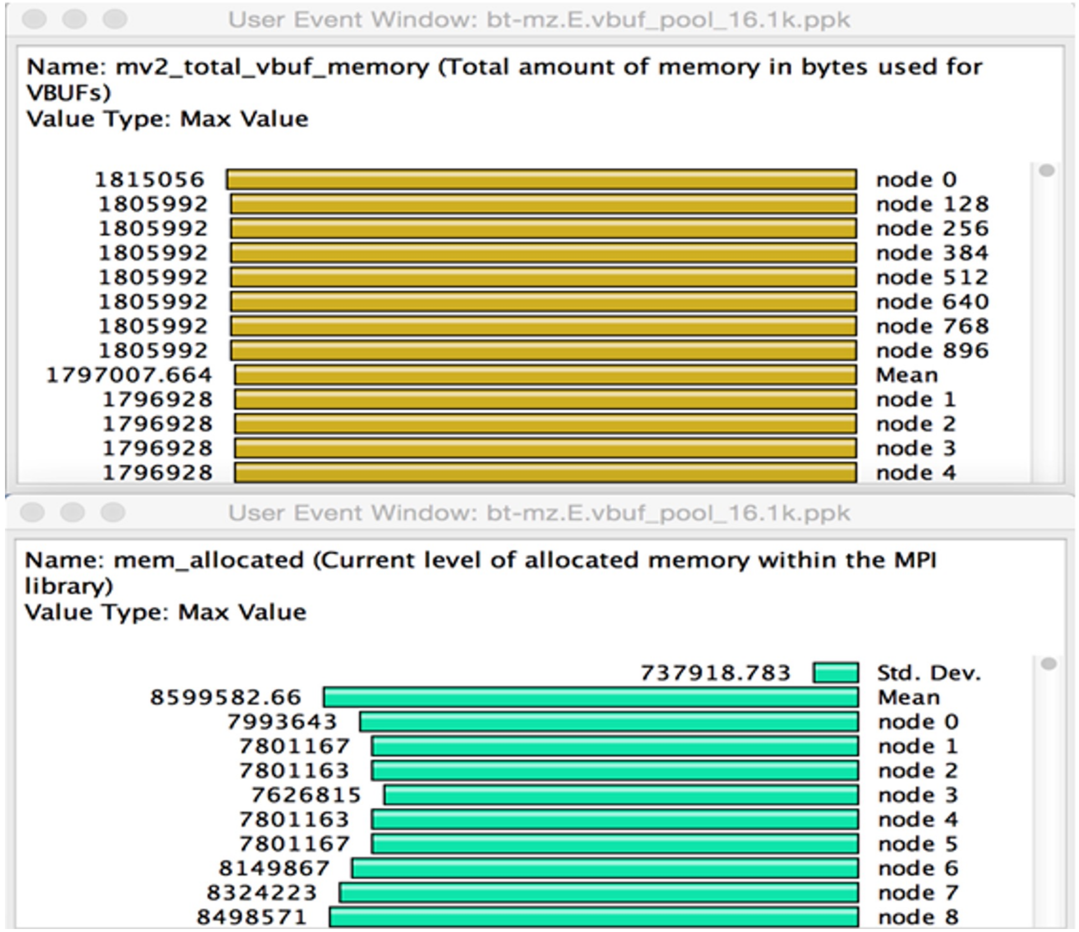
Total memory used by VBUFs is reduced from 3,313,056 to 1,815,056

VBUF Memory Usage Without CVAR





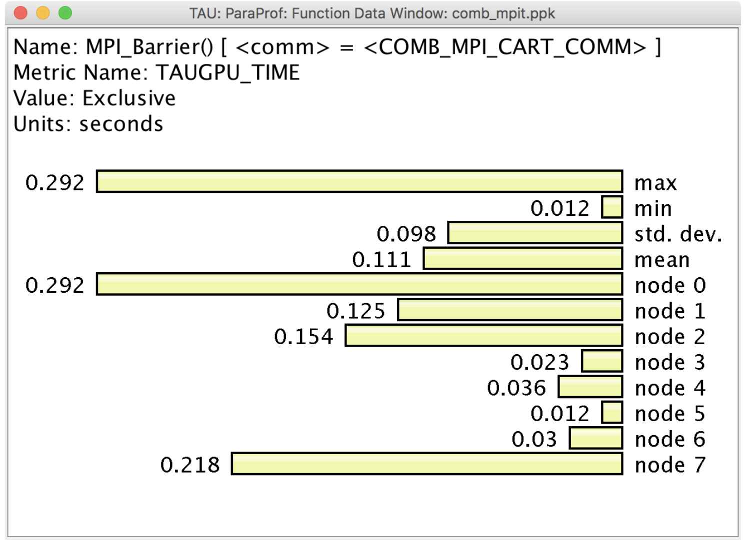
VBUF Memory Usage With CVAR



% 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

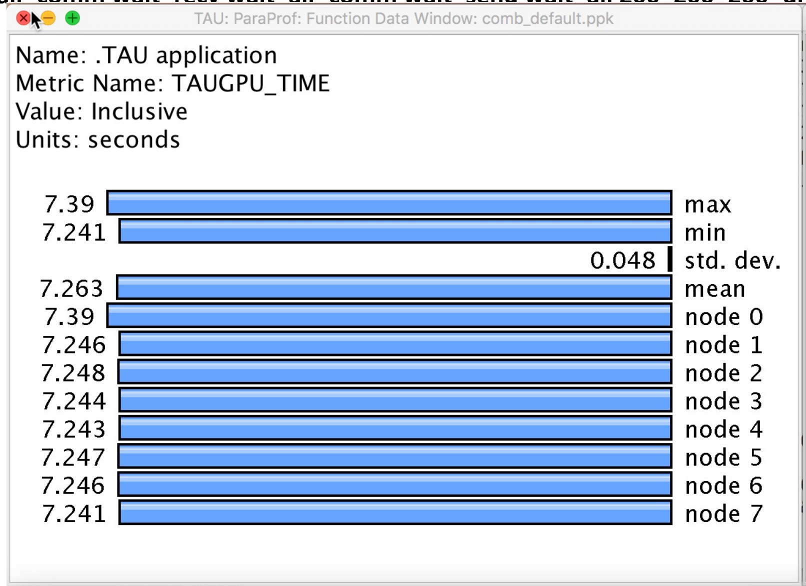
TAU: Extending Control Variables on a Per-Communicator Basis

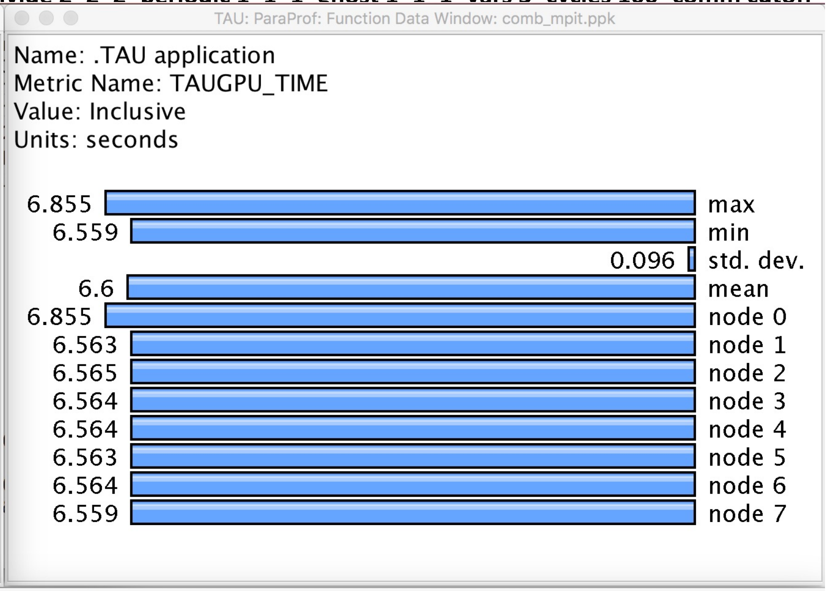
- **Based on named communicators (MPI_Comm_set_name) in an application, TAU allows a user to specify triples to set MPI_T cvars for each communicator:**
  - Communicator name
  - MPI_T CVAR name
  - MPI_T CVAR value
    - % ./configure –mpit –mpi –c++=mpicxx –cc=mpicc –fortran=mpif90 …
    - % make install
    - % export TAU_MPI_T_COMM_METRIC_VALUES=<comm, cvar, value>,…
    - % mpirun –np 64 tau_exec –T mpit ./a.out
    - % paraprof



COMB LLNL App MPI_T Tuning for COMB_MPI_CART_COMM

```
bash-4.2$
    TAU_MPI_T_COMM_METRIC_VALUES=COMB_MPI_CART_COMM,MPIR_CVAR_GPUDIRECT_LIMIT,2097152,COMB_MPI_CART_COMM,MPIR_CVAR_USE_GPUDIRECT_RECEIVE_LIMIT,2097152,
    COMB_MPI_CART_COMM,MPIR_CVAR_CUDA_IPC_THRESHOLD,16384 MV2_USE_CUDA=1 mpirun -np 8 tau_exec -ebs -T mvapich2,mpit,cuda9,cupti,communicators,gnu -cupti ./comb -comm
    post_recv wait_all -comm post_send wait_all -comm wait_recv wait_all -comm wait_send wait_all 200 200 200 -divide 2 2 2 -periodic 1 1 1 -ghost 1 1 1 -vars 3 -cycles 100 -comm cutoff
    250 -omp_threads 1
Started rank 0 of 8
Node lassen710
Compiler COMB_COMPILER
Cuda compiler COMB_CUDA_COMPILER
GPU 0 visible undefined
Not built with openmp, ignoring -omp_threads 1.
Cart coords     0   0   0
Message policy cutoff 250
Post Recv using wait_all method
Post Send using wait_all method
Wait Recv using wait_all method
Wait Send using wait_all method
Num cycles      100
Num vars        3
ghost_widths    1   1   1
sizes           200   200   200
divisions       2   2   2
periodic        1   1   1
division map
map             0   0   0
map             100   100   100
map             200   200   200
Starting test memcpy seq dst Host src Host
Starting test Comm mock Mesh seq Host Buffers seq Host seq Host
Starting test Comm mpi Mesh seq Host Buffers seq Host seq Host
```



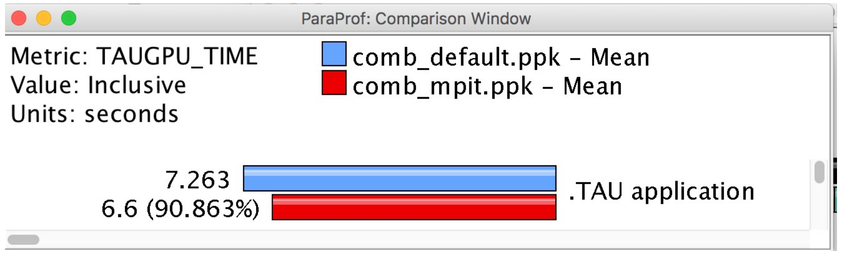


Metadata for n,c,t 0,0,0

Name	Value
TAU_MPI_T_COMM_METRIC_VALUES	COMB_MPI_CART_COMM,MPIR_CVAR_GPUDIRECT_LIMIT,2097152,COMB_MPI_CART_COMM,MPIR_CVAR...

Default

With MPI_T CVARs



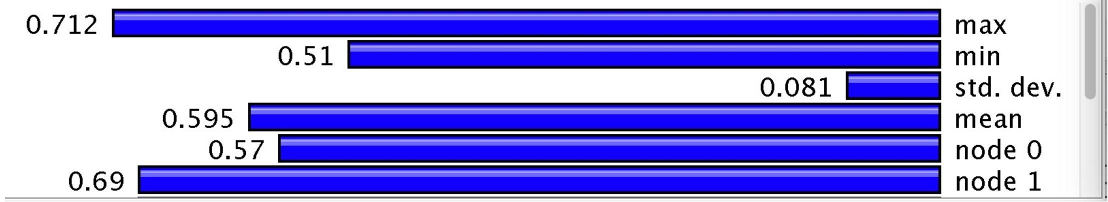
COMB Profile

TAU: ParaProf: Statistics for: node 0 - comb_mpit.ppk

Name △	Exclusive TAUGP...	Inclusive TAUGP...	Calls	Child Calls
.TAU application	3.114	6.855	1	6,806
[CONTEXT] .TAU application	0	3.09	103	0
[SAMPLE] COMB::detail::reset_1::operator()(int, int, int, int) const [{/usr/global/tools/tau/tr	0.57	0.57	19	0
[SAMPLE] COMB::detail::set_1::operator()(int, int, int, int) const [{/usr/global/tools/tau/trai	0.42	0.42	14	0
[SAMPLE] COMB::detail::set_copy::operator()(int, int) const [{/usr/global/tools/tau/training	0.06	0.06	2	0
[SAMPLE] COMB::detail::set_copy::operator()(int, int) const [{/usr/global/tools/tau/training	0.45	0.45	15	0
[SAMPLE] COMB::detail::set_n1::operator()(int, int) const [{/usr/global/tools/tau/training/a	0.06	0.06	2	0
[SAMPLE] __nv_hdl_wrapper_t<false, false, __nv_dl_tag<void (*)(CommContext<mock_pol	0.03	0.03	1	0
[SAMPLE] syscall [{/usr/lib64/libc-2.17.so} {0}]	0.03	0.03	1	0
[SAMPLE] void detail::copy_idxr_idxr<double const, detail::indexer_list_idx, double, detail:	0.03	0.03	1	0
[SUMMARY] void COMB::do_cycles<mock_pol, seq_pol, seq_pol, seq_pol>(CommContext<	0.36	0.36	12	0
[SUMMARY] void COMB::do_cycles<mock_pol, seq_pol, seq_pol, seq_pol>(CommContext<	0.33	0.33	11	0
[SUMMARY] void COMB::do_cycles<mpi_pol, seq_pol, seq_pol, seq_pol>(CommContext<n	0.39	0.39	13	0
[SUMMARY] void COMB::do_cycles<mpi_pol, seq_pol, seq_pol, seq_pol>(CommContext<n	0.36	0.36	12	0
MPI_Barrier()	0.292	0.292	8	0
MPI_Barrier() [<comm> = <COMB_MPI_CART_COMM>]	0.292	0.292	8	0

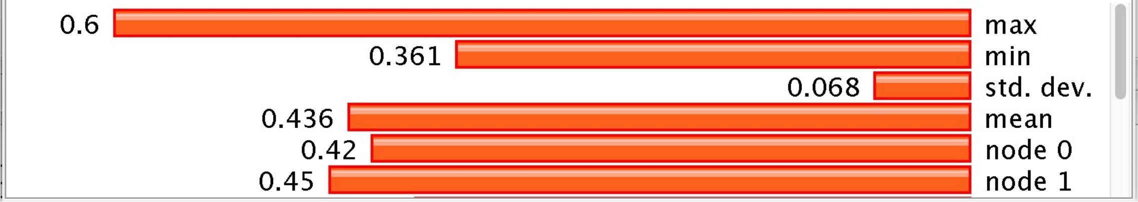
TAU: ParaProf: Function Data Window: comb_mpit.ppk

Name: .TAU application => [CONTEXT] .TAU application => [SAMPLE] COMB::detail::reset_1::operator()(int, int, int, int) const [{/usr/global/tools/tau/training/apps/COMB_LLNL/Comb/include/comb.hpp} {121}]
Metric Name: TAUGPU_TIME
Value: Exclusive
Units: seconds



TAU: ParaProf: Function Data Window: comb_mpit.ppk

Name: .TAU application => [CONTEXT] .TAU application => [SAMPLE] COMB::detail::set_1::operator()(int, int, int, int) const [{/usr/global/tools/tau/training/apps/COMB_LLNL/Comb/include/comb.hpp} {90}]
Metric Name: TAUGPU_TIME
Value: Exclusive
Units: seconds



CVARs Exposed by MVAPICH2

Metadata for n,c,t 0,0,0

Name	Value
MPI Processor Name	lassen710
MPIR_CVAR_CUDA_IPC_THRESHOLD	16384
MPIR_CVAR_GPUDIRECT_LIMIT	2097152
MPIR_CVAR_USE_GPUDIRECT_RECEIVE_LIMIT	2097152
MPI_T CVAR: MPIR_CVAR_ABORT_ON_LEAKED_HANDLES	If true, MPI will call MPI_Abort at MPI_Finalize if any MPI object handles ha...
MPI_T CVAR: MPIR_CVAR_ALLGATHERV_PIPELINE_MSG_SIZE	The smallest message size that will be used for the pipelined, large-mes...
MPI_T CVAR: MPIR_CVAR_ALLGATHER_COLLECTIVE_ALGORITHM	This CVAR selects proper collective algorithm for allgather operation.
MPI_T CVAR: MPIR_CVAR_ALLGATHER_LONG_MSG_SIZE	For MPI_Allgather and MPI_Allgatherv, the long message algorithm will be...
MPI_T CVAR: MPIR_CVAR_ALLGATHER_SHORT_MSG_SIZE	For MPI_Allgather and MPI_Allgatherv, the short message algorithm will b...
MPI_T CVAR: MPIR_CVAR_ALLREDUCE_COLLECTIVE_ALGORITHM	This CVAR selects proper collective algorithm for allreduce operation.
MPI_T CVAR: MPIR_CVAR_ALLREDUCE_SHORT_MSG_SIZE	the short message algorithm will be used if the send buffer size is <= th...
MPI_T CVAR: MPIR_CVAR_ALLTOALLV_COLLECTIVE_ALGORITHM	This CVAR selects proper collective algorithm for alltoallv operation.
MPI_T CVAR: MPIR_CVAR_ALLTOALL_COLLECTIVE_ALGORITHM	This CVAR selects proper collective algorithm for alltoall operation.
MPI_T CVAR: MPIR_CVAR_ALLTOALL_MEDIUM_MSG_SIZE	the medium message algorithm will be used if the per-destination messa...
MPI_T CVAR: MPIR_CVAR_ALLTOALL_SHORT_MSG_SIZE	the short message algorithm will be used if the per-destination message...
MPI_T CVAR: MPIR_CVAR_ALLTOALL_THROTTLE	max no. of irecvs/isends posted at a time in some alltoall algorithms. Set...
MPI_T CVAR: MPIR_CVAR_ASYNC_PROGRESS	If set to true, MPICH will initiate an additional thread to make asynchrono...
MPI_T CVAR: MPIR_CVAR_BCAST_COLLECTIVE_ALGORITHM	This CVAR selects proper collective algorithm for broadcast operation.
MPI_T CVAR: MPIR_CVAR_BCAST_LONG_MSG_SIZE	Let's define short messages as messages with size < MPIR_CVAR_BCAST_...
MPI_T CVAR: MPIR_CVAR_BCAST_MIN_PROCS	Let's define short messages as messages with size < MPIR_CVAR_BCAST_...
MPI_T CVAR: MPIR_CVAR_BCAST_SHORT_MSG_SIZE	Let's define short messages as messages with size < MPIR_CVAR_BCAST_...
MPI_T CVAR: MPIR_CVAR_CH3_EAGER_MAX_MSG_SIZE	This cvar controls the message size at which CH3 switches from eager to...
MPI_T CVAR: MPIR_CVAR_CH3_ENABLE_HCOLL	If true, enable HCOLL collectives.
MPI_T CVAR: MPIR_CVAR_CH3_INTERFACE_HOSTNAME	If non-NULL, this cvar specifies the IP address that other processes shoul...
MPI_T CVAR: MPIR_CVAR_CH3_NOLOCAL	If true, force all processes to operate as though all processes are located...
MPI_T CVAR: MPIR_CVAR_CH3_ODD_EVEN_CLIQUES	If true, odd procs on a node are seen as local to each other, and even pr...
MPI_T CVAR: MPIR_CVAR_CH3_PORT_RANGE	The MPIR_CVAR_CH3_PORT_RANGE environment variable allows you to s...
MPI_T CVAR: MPIR_CVAR_CH3_RMA_ACTIVE_REQ_THRESHOLD	Threshold of number of active requests to trigger blocking waiting in op...
MPI_T CVAR: MPIR_CVAR_CH3_RMA_DELAY_ISSUING_FOR_PIGGYBACKING	Specify if delay issuing of RMA operations for piggybacking LOCK/UNLOC...
MPI_T CVAR: MPIR_CVAR_CH3_RMA_OP_GLOBAL_POOL_SIZE	Size of the Global RMA operations pool (in number of operations) that st...
MPI_T CVAR: MPIR_CVAR_CH3_RMA_OP_PIGGYBACK_LOCK_DATA_SIZE	Specify the threshold of data size of a RMA operation which can be piggy...
MPI_T CVAR: MPIR_CVAR_CH3_RMA_OP_WIN_POOL_SIZE	Size of the window-private RMA operations pool (in number of operation...
MPI_T CVAR: MPIR_CVAR_CH3_RMA_POKE_PROGRESS_REQ_THRESHOLD	Threshold at which the RMA implementation attempts to complete reque...
MPI_T CVAR: MPIR_CVAR_CH3_RMA_SCALABLE_FENCE_PROCESS_NUM	Specify the threshold of switching the algorithm used in FENCE from the ...
MPI_T CVAR: MPIR_CVAR_CH3_RMA_SLOTS_SIZE	Number of RMA slots during window creation. Each slot contains a linked...
MPI_T CVAR: MPIR_CVAR_CH3_RMA_TARGET_GLOBAL_POOL_SIZE	Size of the Global RMA targets pool (in number of targets) that stores inf...
MPI_T CVAR: MPIR_CVAR_CH3_RMA_TARGET_LOCK_DATA_BYTES	Size (in bytes) of available lock data this window can provided. If current ...
MPI_T CVAR: MPIR_CVAR_CH3_RMA_TARGET_LOCK_ENTRY_WIN_POOL_SIZE	Size of the window-private RMA lock entries pool (in number of lock entr...

Path Aware Profiling in TAU and MVAPICH2

- **To identify the path taken by an MPI message:**
  - GPU memory to GPU memory
  - Unique send and receive path ids captured
- **Configure TAU with -PROFILEPATHS:**
- **Partition the time in MPI pt-to-pt operations:**
  - MPI_Send and MPI_Recv
  - Parameter based profiling identifies paths
- **Path captured as metadata in TAU profiles**
  - PVARs based on CUPTI counters
  - MVAPICH2 exports PVARs to TAU with MPI_T

Metadata for n,c,t 0,0,0

Name	Value
TAU_PROFILE	on
TAU_PROFILE_FORMAT	profile
TAU_RECV_PATH_ID_|_0	gpu1-gpu0
TAU_RECV_PATH_ID_|_1	gpu2-gpu0
TAU_RECV_PATH_ID_|_10	internodelink-nic
TAU_RECV_PATH_ID_|_2	gpu3-gpu0
TAU_RECV_PATH_ID_|_3	gpu2-gpu1
TAU_RECV_PATH_ID_|_4	gpu3-gpu1
TAU_RECV_PATH_ID_|_5	gpu3-gpu2
TAU_RECV_PATH_ID_|_6	cpu-gpu0
TAU_RECV_PATH_ID_|_7	cpu-gpu1
TAU_RECV_PATH_ID_|_8	cpu-gpu2
TAU_RECV_PATH_ID_|_9	cpu-gpu3
TAU_RECYCLE_THREADS	off
TAU_REGION_ADDRESSES	off
TAU_SAMPLING	off
TAU_SEND_PATH_ID_|_0	gpu0-gpu1
TAU_SEND_PATH_ID_|_1	gpu0-gpu2
TAU_SEND_PATH_ID_|_10	nic-internodelink
TAU_SEND_PATH_ID_|_2	gpu0-gpu3
TAU_SEND_PATH_ID_|_3	gpu1-gpu2
TAU_SEND_PATH_ID_|_4	gpu1-gpu3
TAU_SEND_PATH_ID_|_5	gpu2-gpu3
TAU_SEND_PATH_ID_|_6	gpu0-cpu
TAU_SEND_PATH_ID_|_7	gpu1-cpu
TAU_SEND_PATH_ID_|_8	gpu2-cpu
TAU_SEND_PATH_ID_|_9	gpu3-cpu

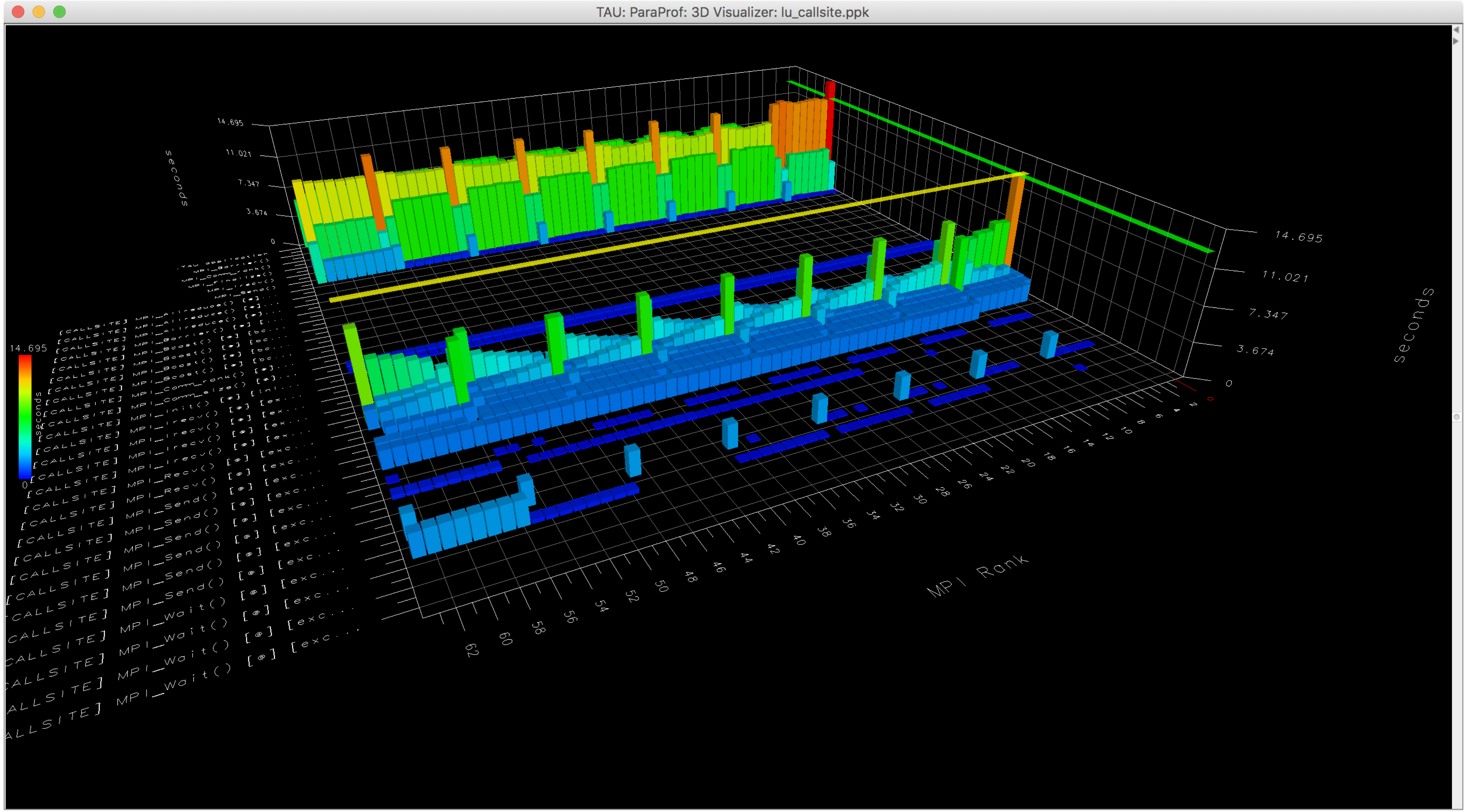
Path Aware Profiling in TAU and MVAPICH2

- **Available for download in TAU v2.29.1**

TAU: ParaProf: Statistics for: node 0 - path_3ranks.ppk

Name	Exclusive ... ▽	Inclusive ...	Calls	Child ...
main [{/g/g24/shende1/mpit/path_test_3ranks.c} {61,0}]	40.332	42.472	1	12
MPI_Init()	0.86	0.86	1	0
MPI_Send()	0.746	0.746	4	2
MPI_Send() [<message send path id> = <1006>]	0.617	0.617	2	0
init_accel [{/g/g24/shende1/mpit/path_test_3ranks.c} {42,0}]	0.263	0.263	1	1
MPI_Finalize()	0.254	0.254	1	0
MPI_Send() [<message send path id> = <100600>]	0.129	0.129	2	0
.TAU application	0.033	42.505	1	1
MPI_Barrier()	0.017	0.017	3	0
get_local_rank [{/g/g24/shende1/mpit/path_test_3ranks.c} {26,0}]	0	0	1	0
MPI_Get_processor_name()	0	0	2	0
MPI_Comm_rank()	0	0	1	0
MPI_Comm_size()	0	0	1	0

TAU's ParaProf 3D Browser



Download TAU from U. Oregon

Tuning and Analysis Utilities

http://tau.uoregon.edu

http://taucommander.com

http://www.hpclinux.com [OVA for VirtualBox]

https://e4s.io [Extreme-scale Scientific Software Stack]

for more information

Free download, open source, BSD license

PRL, OACISS, 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, NVIDIA (Mellanox), AWS, AMD, Broadcom, IBM, Google

NASA

Partners:

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

Reference

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 ; tar xf ext.tgz
- ./configure -bfd=download -pdt=<dir> -papi=<dir> -mpi –pthread –c++=mpicxx –cc=mpicc –fortran=mpif90 –dwarf=download –unwind=download –otf=download –iowrapper –papi=<dir>
- make install

•Using TAU:

- export TAU_MAKEFILE=<taudir>/x86_64/lib/Makefile.tau-<TAGS>
- make CC=tau_cc.sh   CXX=tau_cxx.sh   F90=tau_f90.sh

Compile-Time Options

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

-optVerbose	Turn on verbose debugging messages
-optCompInst	Use compiler based instrumentation
-optNoCompInst	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/gfparse) …

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.

…

TAU's Runtime Environment Variables

Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_TRACK_MEMORY_FOOTPRINT	0	Setting to 1 turns on tracking memory usage by sampling periodically the resident set size and high water mark of memory usage
TAU_TRACK_POWER	0	Tracks power usage by sampling periodically.
TAU_CALLPATH_DEPTH	2	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)
TAU_SAMPLING	1	Setting to 1 enables event-based sampling.
TAU_TRACK_SIGNALS	0	Setting to 1 generate debugging callstack info when a program crashes
TAU_COMM_MATRIX	0	Setting to 1 generates communication matrix display using context events
TAU_THROTTLE	1	Setting to 0 turns off throttling. Throttles instrumentation in lightweight routines that are called frequently
TAU_THROTTLE_NUMCALLS	100000	Specifies the number of calls before testing for throttling
TAU_THROTTLE_PERCALL	10	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
TAU_CALLSITE	0	Setting to 1 enables callsite profiling that shows where an instrumented function was called. Also compatible with tracing.
TAU_PROFILE_FORMAT	Profile	Setting to "merged" generates a single file. "snapshot" generates xml format
TAU_METRICS	TIME	Setting to a comma separated list generates other metrics. (e.g., ENERGY,TIME,P_VIRTUAL_TIME,PAPI_FP_INS,PAPI_NATIVE_<event>:<subevent>)

Runtime Environment Variables

Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_TRACE_FORMAT	Default	Setting to "otf2" turns on TAU's native OTF2 trace generation (configure with –otf=download)
TAU_EBS_UNWIND	0	Setting to 1 turns on unwinding the callstack during sampling (use with tau_exec –ebs or TAU_SAMPLING=1)
TAU_EBS_RESOLUTION	line	Setting to "function" or "file" changes the sampling resolution to function or file level respectively.
TAU_TRACK_LOAD	0	Setting to 1 tracks system load on the node
TAU_SELECT_FILE	Default	Setting to a file name, enables selective instrumentation based on exclude/include lists specified in the file.
TAU_OMPT_SUPPORT_LEVEL	basic	Setting to "full" improves resolution of OMPT TR6 regions on threads 1.. N-1. Also, "lowoverhead" option is available.
TAU_OMPT_RESOLVE_ADDRESS_EAGERLY	1	Setting to 1 is necessary for event based sampling to resolve addresses with OMPT. Setting to 0 allows the user to do offline address translation.

Runtime Environment Variables

Environment Variable	Default	Description
TAU_TRACK_MEMORY_LEAKS	0	Tracks allocates that were not de-allocated (needs –optMemDbg or tau_exec –memory)
TAU_EBS_SOURCE	TIME	Allows using PAPI hardware counters for periodic interrupts for EBS (e.g., TAU_EBS_SOURCE=PAPI_TOT_INS when TAU_SAMPLING=1)
TAU_EBS_PERIOD	100000	Specifies the overflow count for interrupts
TAU_MEMDBG_ALLOC_MIN/MAX	0	Byte size minimum and maximum subject to bounds checking (used with TAU_MEMDBG_PROTECT_*)
TAU_MEMDBG_OVERHEAD	0	Specifies the number of bytes for TAU's memory overhead for memory debugging.
TAU_MEMDBG_PROTECT_BELOW/ABOVE	0	Setting to 1 enables tracking runtime bounds checking below or above the array bounds (requires –optMemDbg while building or tau_exec –memory)
TAU_MEMDBG_ZERO_MALLOC	0	Setting to 1 enables tracking zero byte allocations as invalid memory allocations.
TAU_MEMDBG_PROTECT_FREE	0	Setting to 1 detects invalid accesses to deallocated memory that should not be referenced until it is reallocated (requires –optMemDbg or tau_exec –memory)
TAU_MEMDBG_ATTEMPT_CONTINUE	0	Setting to 1 allows TAU to record and continue execution when a memory error occurs at runtime.
TAU_MEMDBG_FILL_GAP	Undefined	Initial value for gap bytes
TAU_MEMDBG_ALINGMENT	Sizeof(int)	Byte alignment for memory allocations
TAU_EVENT_THRESHOLD	0.5	Define a threshold value (e.g., .25 is 25%) to trigger marker events for min/max

Acknowledgment

"This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative."