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](http://mug.mvapich.cse.ohio-state.edu)

Slides:
[http://tau.uoregon.edu/TAU_MUG21.pdf](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
• **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?
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

```
int foo()
{
    int a;
    a = a + 1;
    bar();
    a = a + 1;
    return a;
}
```
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?

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

O(KB) → O(TB)
Sampling

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

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

void foo(int i)
{
    if (i > 0)
        foo(i - 1);
}```
Measurements 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
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
- Opapi 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 --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:

```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
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 performance
- 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 \textit{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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU MEMBGS PROTECT BELOW</td>
<td>off</td>
</tr>
<tr>
<td>TAU MEMBGS PROTECT FREE</td>
<td>off</td>
</tr>
<tr>
<td>TAU MPI T ENABLE USER_TUNING_POLICY</td>
<td>off</td>
</tr>
<tr>
<td>TAU OPENMP RUNTIME</td>
<td>on</td>
</tr>
<tr>
<td>TAU OPENMP RUNTIME EVENTS</td>
<td>on</td>
</tr>
<tr>
<td>TAU OPENMP RUNTIME STATES</td>
<td>off</td>
</tr>
<tr>
<td>TAU OUTPUT CUDA CSV</td>
<td>off</td>
</tr>
<tr>
<td>TAU PAPI MUXPLEXING</td>
<td>off</td>
</tr>
<tr>
<td>TAU PROFILE</td>
<td>on</td>
</tr>
<tr>
<td>TAU PROFILE FORMAT</td>
<td>profile</td>
</tr>
<tr>
<td>TAU RECOMMENDATION PHASE_ALLOCATE</td>
<td>MPI T RECOMMEND SHARP USAGE: No performance benefit foreseen with SHaP usage</td>
</tr>
<tr>
<td>TAU RECOMMENDATION PHASE DEALLOCATE</td>
<td>MPI T RECOMMEND SHARP USAGE: You could see potential improvement in performance by enabling MV2 ENABLE SHaP in MVAPICH version 2.3a and above</td>
</tr>
<tr>
<td>TAU RECOMMENDATION PHASE DRIVER</td>
<td>MPI T RECOMMEND SHARP USAGE: You could see potential improvement in performance by enabling MV2 ENABLE SHaP in MVAPICH version 2.3a and above</td>
</tr>
<tr>
<td>TAU RECOMMENDATION PHASE INT</td>
<td>MPI T RECOMMEND SHARP USAGE: No performance benefit foreseen with SHaP usage</td>
</tr>
<tr>
<td>TAU RECOMMENDATION PHASE PROFILE</td>
<td>MPI T RECOMMEND SHARP USAGE: You could see potential improvement in performance by enabling MV2 ENABLE SHaP in MVAPICH version 2.3a and above</td>
</tr>
<tr>
<td>TAU REGION ADDRESSES</td>
<td>off</td>
</tr>
<tr>
<td>TAU SAMPLING</td>
<td>off</td>
</tr>
<tr>
<td>TAU SHOW MEMORY FUNCTIONS</td>
<td>off</td>
</tr>
<tr>
<td>TAU SIGNALS GDB</td>
<td>off</td>
</tr>
<tr>
<td>TAU THORLITE</td>
<td>on</td>
</tr>
<tr>
<td>TAU THORLITE NUMCALLS</td>
<td>100000</td>
</tr>
<tr>
<td>TAU THORLITE PERCALL</td>
<td>10</td>
</tr>
<tr>
<td>TAU TRACe</td>
<td>off</td>
</tr>
<tr>
<td>TAU TRACe FORMAT</td>
<td>tau</td>
</tr>
<tr>
<td>TAU TRACK CUDA CDP</td>
<td>off</td>
</tr>
<tr>
<td>TAU TRACK CUDA ENV</td>
<td>off</td>
</tr>
<tr>
<td>TAU TRACK CUDA INSTRUCTIONS</td>
<td>off</td>
</tr>
<tr>
<td>TAU TRACK CUDA SASS</td>
<td>off</td>
</tr>
<tr>
<td>TAU TRACK HEADROOM</td>
<td>off</td>
</tr>
<tr>
<td>TAU TRACK HEAP</td>
<td>off</td>
</tr>
<tr>
<td>TAU TRACK IO PARAMS</td>
<td>off</td>
</tr>
<tr>
<td>TAU TRACK MEMORY FOOTPRINT</td>
<td>off</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>TraitField</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mpi_T PVAR0: mem_allocated</td>
<td>Current level of allocated memory within the MPI library</td>
</tr>
<tr>
<td>Mpi_T PVAR10: mv2_num_2level_comm_success</td>
<td>Number of successful 2-level comm creations</td>
</tr>
<tr>
<td>Mpi_T PVAR11: mv2_num SHMEM_coll_calls</td>
<td>Number of times MV2 shared-memory collective calls were invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR12: mpt_progress_poll</td>
<td>CH3 RDMA progress engine polling count</td>
</tr>
<tr>
<td>Mpi_T PVAR13: mvp_read_progress_poll</td>
<td>CH3 SMP read progress engine polling count</td>
</tr>
<tr>
<td>Mpi_T PVAR14: mv2_num READ_progress_poll</td>
<td>CH3 SMP read progress engine polling count</td>
</tr>
<tr>
<td>Mpi_T PVAR15: mvp_read_progress_poll_success</td>
<td>CH3 SMP read progress engine polling success count</td>
</tr>
<tr>
<td>Mpi_T PVAR16: mv2_num WRITE_progress_poll</td>
<td>CH3 SMP write progress engine polling count</td>
</tr>
<tr>
<td>Mpi_T PVAR17: rdma_ud_retransmissions</td>
<td>CH3 RDMA UD retransmission count</td>
</tr>
<tr>
<td>Mpi_T PVAR18: mv2_coll_bcast_bimodal</td>
<td>Number of times MV2 bimodal bcast algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR19: mvp_write_progress_poll</td>
<td>CH3 SMP write progress engine polling count</td>
</tr>
<tr>
<td>Mpi_T PVAR20: mv2(MPI)scatter_double_all</td>
<td>Number of times MV2 scatter double allgather bcast algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR21: mv2 scatter dst bcast algorithm invocation count</td>
<td>Maximum level of memory ever allocated within the MPI library</td>
</tr>
<tr>
<td>Mpi_T PVAR22: mv2 scatter bcast_arm</td>
<td>Number of times MV2 scatter bcast algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR23: mv2 scatter bcast arm</td>
<td>Number of times MV2 scatter bcast arm algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR24: mv2 scatter bcast arm</td>
<td>Number of times MV2 scatter bcast arm algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR25: mv2 scatter bcast arm</td>
<td>Number of times MV2 scatter bcast arm algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR26: mv2 scatter bcast arm</td>
<td>Number of times MV2 scatter bcast arm algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR27: mv2 scatter bcast arm</td>
<td>Number of times MV2 scatter bcast arm algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR28: mv2 scatter bcast arm</td>
<td>Number of times MV2 scatter bcast arm algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR29: mv2 scatter bcast arm</td>
<td>Number of times MV2 scatter bcast arm algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR30: mv2 scatter bcast arm</td>
<td>Number of times MV2 scatter bcast arm algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR31: mv2 scatter bcast arm</td>
<td>Number of times MV2 scatter bcast arm algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR32: mpt_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR33: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR34: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR35: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR36: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR37: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR38: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR39: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR40: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR41: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR42: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR43: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR44: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR45: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR46: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR47: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR48: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR49: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR50: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>Mpi_T PVAR51: mv2_md_copy</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
</tbody>
</table>
### CVARs Exposed by MVAPICH2

<table>
<thead>
<tr>
<th>CVAR Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CMA</strong></td>
<td>Message Copy and Assemble. This CVAR controls whether the message copy and assemble functionality is enabled. It is used to enable or disable the MCA (Message Collection and Assemble) feature.</td>
</tr>
<tr>
<td><strong>CMA_NC_ADDR</strong></td>
<td>Non-collaborative message copy and assemble. This CVAR controls whether non-collaborative message copy and assemble is enabled. It is used to enable or disable the non-collaborative MCA feature.</td>
</tr>
<tr>
<td><strong>CMA_PCB</strong></td>
<td>Message Control Block. This CVAR controls the maximum size of the message control block.</td>
</tr>
<tr>
<td><strong>CMA_PCBBUFFERS</strong></td>
<td>Message Control Block Buffers. This CVAR controls the number of message control block buffers.</td>
</tr>
<tr>
<td><strong>CMA_PCB_SIZE</strong></td>
<td>Message Control Block Size. This CVAR controls the size of the message control block.</td>
</tr>
<tr>
<td><strong>CMA_QOS</strong></td>
<td>Quality of Service. This CVAR controls the quality of service for the message copy and assemble functionality.</td>
</tr>
<tr>
<td><strong>CMA_QOS_BUFFERS</strong></td>
<td>Quality of Service Buffers. This CVAR controls the number of quality of service buffers.</td>
</tr>
<tr>
<td><strong>CMA_QOS_SIZE</strong></td>
<td>Quality of Service Size. This CVAR controls the size of the quality of service buffer.</td>
</tr>
<tr>
<td><strong>CMA_TCA</strong></td>
<td>Traffic Class Association. This CVAR controls the traffic class association functionality.</td>
</tr>
<tr>
<td><strong>CMA_TCA_BUFFERS</strong></td>
<td>Traffic Class Association Buffers. This CVAR controls the number of traffic class association buffers.</td>
</tr>
<tr>
<td><strong>CMA_TCA_SIZE</strong></td>
<td>Traffic Class Association Size. This CVAR controls the size of the traffic class association buffer.</td>
</tr>
<tr>
<td><strong>CMA_TCA_QOS</strong></td>
<td>Traffic Class Association Quality of Service. This CVAR controls the quality of service for the traffic class association functionality.</td>
</tr>
<tr>
<td><strong>CMA_TCA_QOS_BUFFERS</strong></td>
<td>Traffic Class Association Quality of Service Buffers. This CVAR controls the number of traffic class association quality of service buffers.</td>
</tr>
<tr>
<td><strong>CMA_TCA_QOS_SIZE</strong></td>
<td>Traffic Class Association Quality of Service Size. This CVAR controls the size of the traffic class association quality of service buffer.</td>
</tr>
</tbody>
</table>

This list includes some of the most commonly used CVARs for controlling message copy and assemble functionality in MVAPICH2. Each CVAR has a specific purpose and can be adjusted to optimize performance for different workloads.
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

<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_malloc_calls (Number of MPIT_malloc 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_free_calls (Number of MPIT_free 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_memalign_calls (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_free_calls (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
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

Default

With MPI_T CVARs
COMB Profile

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] COMB::detail::reset_1::operator() (int, int, int, int) const
[/usr/global/tools/tau/trai
0.57 0.57 19 0
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] COMB::detail::set_1::operator() (int, int, int, int) const
[/usr/global/tools/tau/trai
0.42 0.42 14 0
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] COMB::detail::set_copy::operator() (int, int, int) const
[/usr/global/tools/tau/trai
0.06 0.06 2 0
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] COMB::detail::set_n1::operator() (int, int) const
[/usr/global/tools/tau/trai
0.06 0.06 2 0
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] __nv_hdl_wrapr_t::false, false, __nv_dl_tag <void (*) (CommContext, mock_pol
0.03 0.03 1 0
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] syscall [/usr/lib64/libc-2.17.so] {0}
0.03 0.03 1 0
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] void detail::copy_idxr_idxr <double const, detail::indexer_list_idx, double, detail
0.03 0.03 1 0
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] void COMB::do_cycles <mock_pol, seq_pol, seq_pol, seq_pol> (CommContext
0.36 0.36 12 0
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] void COMB::do_cycles <mock_pol, seq_pol, seq_pol, seq_pol> (CommContext
0.33 0.33 11 0
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] void COMB::do_cycles <mpi_pol, seq_pol, seq_pol, seq_pol> (CommContext
0.39 0.39 13 0
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] void COMB::do_cycles <mpi_pol, seq_pol, seq_pol, seq_pol> (CommContext
0.36 0.36 12 0
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] void MPL_Barrier()
0.292 0.292 8 0
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] void MPL_Barrier() [ <comm> = <COMB_MPI_CART_COMM> ]
0.292 0.292 8 0
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] COMB::detail::setup::operator() (int, int, int, int) const
[/usr/global/tools/tau/trai
0.436 0.42 0.45 0.068
[CONTEXT]. TAU application

Name: .TAU application => [CONTEXT]. TAU application => [SAMPLE] COMB::detail::setup::operator() (int, int, int, int) const
[/usr/global/tools/tau/trai
0.436 0.42 0.45 0.068
[CONTEXT]. TAU application

Pan University of Oregon
## CVARs Exposed by MVAPICH2

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI Processor Name</td>
<td>lassen710</td>
</tr>
<tr>
<td>MPRI_CVAR_CUDA_IPC_THRESHOLD</td>
<td>16384</td>
</tr>
<tr>
<td>MPRI_CVAR_GPUDIRECT_LIMIT</td>
<td>2097152</td>
</tr>
<tr>
<td>MPRI_CVAR_GPUDIRECT_RECEIVE_LIMIT</td>
<td></td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_ABORT_ON_LEAKED_HANDLES</td>
<td>If true, MPI will call MPI_Finalize if any MPI object handles ha...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_ALLGATHER_PIPELINE_MSG_SIZE</td>
<td>The smallest message size that will be used for the pipelined, large-mes...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_ALLGATHER_COLLECTIVE_ALGORITHM</td>
<td>This CVAR selects proper collective algorithm for allgather operation. For MPI Allgather and MPI Allgatherv, the long message algorithm will be...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_ALLGATHER_LONG.Msg_SIZE</td>
<td>For MPI Allgather and MPI Allgatherv, the short message algorithm will be...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_ALLGATHER_SHORT.Msg_SIZE</td>
<td>This CVAR selects proper collective algorithm for allreduce operation. The short message algorithm will be used if the send buffer size is &lt;= th...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_ALLREDUCE_COLLECTIVE_ALGORITHM</td>
<td>This CVAR selects proper collective algorithm for alltoallv operation. The medium message algorithm will be used if the per-destination messa...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_ALLREDUCE_SHORT.Msg_SIZE</td>
<td>max no. of irecs/исends posted at a time in some alltoall algorithms. Set...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_ALLTOALL(Msg_SIZE)</td>
<td>If set to true, MPICH will initiate an additional thread to make asynchrono...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_ALLTOALLColLECTIVE_ALGORITHM</td>
<td>This CVAR selects proper collective algorithm for broadcast operation. Let's define short messages as messages with size &lt; MPRI_CVAR_BCAST,...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_BCAST_COLLECTIVE_ALGORITHM</td>
<td>Let's define short messages as messages with size &lt; MPRI_CVAR_BCAST,...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_BCAST_LONG.Msg_SIZE</td>
<td>Let's define short messages as messages with size &lt; MPRI_CVAR_BCAST,...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_BCAST_SHORT.Msg_SIZE</td>
<td>This cvr controls the message size at which CH3 switches from eager to...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_EAGER_MAX.Msg_SIZE</td>
<td>If true, enable HCOLL collectives If non-NULL, this cvr specifies the IP address that other processes shoul...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_ENABLE_HCOLL</td>
<td>If true, force all processes to operate as though all processes are located...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_NOLocal</td>
<td>If true, odd procs on a node are seen as local to each other, and even pro...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_Odd.Even.CLiques</td>
<td>The MPRI_CVAR_CH3_PORT_RANGE environment variable allows you to s...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_RMA_ACTIVE_REQ_THRESHOLD</td>
<td>Threshold of number of active requests to trigger blocking waiting in op...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_RMA_DELAY_ISSUEING_FOR_PIGGYBACKING</td>
<td>Specify if delay issuing of DMA operations for piggybacking LOCK/UNLOCK...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_RMA_DOMAIN.OP_GLOBAL_POOL_SIZE</td>
<td>Size of the Global DMA operations pool (in number of operations) that st...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_RMA_DOMAIN.OP_PWM_POOL_SIZE</td>
<td>Specify the threshold of data size of a DMA operation which can be piggy...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_RMA_DOMAIN.OP_WIN_POOL_SIZE</td>
<td>Size of the window-private DMA operations pool (in number of operation...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_RMA_POKE_PROGRESS_REQ_THRESHOLD</td>
<td>Threshold at which the DMA implementation attempts to complete reques...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_RMA_SCALABLE_FENCE_PROCESS_NUM</td>
<td>Specify the threshold of switching the algorithm used in FENCE from the ...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_RMA_SLOTS_SIZE</td>
<td>Number of DMA slots during window creation. Each slot contains a linked...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_RMA_TARGET_GLOBAL_POOL_SIZE</td>
<td>Size of the Global DMA targets pool (in number of targets) that stores inf...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_RMA_TARGET_LOCK_DATA_BYTES</td>
<td>Size (in bytes) of available lock data this window can provided. If current ...</td>
</tr>
<tr>
<td>MPI_T_CVAR: MPRI_CVAR_CH3_RMA_TARGET_LOCK_ENTRY_WIN_POOL_SIZE</td>
<td>Size of the window–private DMA lock entries pool (in number of lock entr...</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU_PROFILE</td>
<td>on</td>
</tr>
<tr>
<td>TAU_PROFILE_FORMAT</td>
<td>profile</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_0</td>
<td>gpu1-gpu0</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_1</td>
<td>gpu2-gpu0</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_10</td>
<td>internodelink-nic</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_2</td>
<td>gpu3-gpu0</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_3</td>
<td>gpu2-gpu1</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_4</td>
<td>gpu3-gpu1</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_5</td>
<td>gpu3-gpu2</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_6</td>
<td>cpu-gpu0</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_7</td>
<td>cpu-gpu1</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_8</td>
<td>cpu-gpu2</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_9</td>
<td>cpu-gpu3</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_0</td>
<td>gpu0-gpu1</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_1</td>
<td>gpu0-gpu2</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_10</td>
<td>nic-internodelink</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_2</td>
<td>gpu0-gpu3</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_3</td>
<td>gpu1-gpu2</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_4</td>
<td>gpu1-gpu3</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_5</td>
<td>gpu2-gpu3</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_6</td>
<td>gpu0-cpu</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_7</td>
<td>gpu1-cpu</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_8</td>
<td>gpu2-cpu</td>
</tr>
<tr>
<td>TAU_RECV_PATH_ID_9</td>
<td>gpu3-cpu</td>
</tr>
</tbody>
</table>
Path Aware Profiling in TAU and MVAPICH2

- Available for download in TAU v2.29.1

<table>
<thead>
<tr>
<th>Name</th>
<th>Exclusive ...</th>
<th>Inclusive ...</th>
<th>Calls</th>
<th>Child ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>main [{/g/g24/shende1/mpit/path_test_3ranks.c} {61,0}]</td>
<td>40.332</td>
<td>42.472</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>MPI_Init()</td>
<td>0.86</td>
<td>0.86</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Send()</td>
<td>0.746</td>
<td>0.746</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>MPI_Send() [ &lt;message send path id&gt; = &lt;1006&gt; ]</td>
<td>0.617</td>
<td>0.617</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>init_accel [{/g/g24/shende1/mpit/path_test_3ranks.c} {42,0}]</td>
<td>0.263</td>
<td>0.263</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MPI_Finalize()</td>
<td>0.254</td>
<td>0.254</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Send() [ &lt;message send path id&gt; = &lt;100600&gt; ]</td>
<td>0.129</td>
<td>0.129</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>.TAU application</td>
<td>0.033</td>
<td>42.505</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MPI_Barrier()</td>
<td>0.017</td>
<td>0.017</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>get_local_rank [{/g/g24/shende1/mpit/path_test_3ranks.c} {26,0}]</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Get_processor_name()</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Comm_rank()</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Comm_size()</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
TAU’s ParaProf 3D Browser
Download TAU from U. Oregon

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
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
- `-optComplInst` Use compiler based instrumentation
- `-optNoComplInst` Do not revert to compiler instrumentation if source instrumentation fails.
- `-optTrackI/O` 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
- 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

<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_FOO_TPRINT</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_TRACK_POWER</td>
<td>0</td>
<td>Tracks power usage by sampling periodically.</td>
</tr>
<tr>
<td>TAU_CALLPATHDEPTH</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>1</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. Throttles 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_CALLSITE</td>
<td>0</td>
<td>Setting to 1 enables callsite profiling that shows where an instrumented function was called. Also compatible with tracing.</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., ENERGY,TIME,P_VIRTUAL_TIME,PAPI_FP_INS,PAPI_NATIVE_&lt;event&gt;:&lt;subevent&gt;)</td>
</tr>
</tbody>
</table>
## 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_TRACE_FORMAT</td>
<td>Default</td>
<td>Setting to “otf2” turns on TAU’s native OTF2 trace generation (configure with --otf=download)</td>
</tr>
<tr>
<td>TAU_EBS_UNWIND</td>
<td>0</td>
<td>Setting to 1 turns on unwinding the callstack during sampling (use with tau_exec --ebs or TAU_SAMPLING=1)</td>
</tr>
<tr>
<td>TAU_EBS_RESOLUTION</td>
<td>line</td>
<td>Setting to “function” or “file” changes the sampling resolution to function or file level respectively.</td>
</tr>
<tr>
<td>TAU_TRACK_LOAD</td>
<td>0</td>
<td>Setting to 1 tracks system load on the node</td>
</tr>
<tr>
<td>TAU_SELECT_FILE</td>
<td>Default</td>
<td>Setting to a file name, enables selective instrumentation based on exclude/include lists specified in the file.</td>
</tr>
<tr>
<td>TAU_OMPT_SUPPORT_LEVEL</td>
<td>basic</td>
<td>Setting to “full” improves resolution of OMPT TR6 regions on threads 1.. N-1. Also, “lowoverhead” option is available.</td>
</tr>
<tr>
<td>TAU_OMPT_RESOLVE_ADDRESS_EAGERLY</td>
<td>1</td>
<td>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.</td>
</tr>
</tbody>
</table>
## Runtime Environment Variables

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
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.”