Performance Engineering using MVAPICH and TAU

Sameer Shende, Srinivasan Ramesh, Allen D. Malony, Wyatt Spear, Kevin Huck
University of Oregon

MVAPICH User’s Group (MUG) Meeting
12:00pm – 12:30pm, Tuesday, August 20, 2019
Ohio Supercomputing Center, Columbus, Ohio
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
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 Instrumentation Approach

Supports both direct and indirect performance observation

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

Support for user-defined events

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

Event types

• Interval events (begin/end events)
  • Measures exclusive & inclusive durations between events
  • Metrics monotonically increase

• Atomic events (trigger with data value)
  • Used to capture performance data state
  • Shows extent of variation of triggered values (min/max/mean)

Code events

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

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

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

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

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

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

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

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

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

• Introduction

• The MPI Tools Interfaces and Benefits

• Integrating TAU and MVAPICH2 with MPI_T
Overview of the MVAPICH2 Project
High Performance open-source MPI Library for InfiniBand, Omni-Path, Ethernet/iWARP, and RDMA over Converged Ethernet (RoCE)

• MVAPICH (MPI-1), MVAPICH2 (MPI-2.2 and MPI-3.1), Started in 2001, First version available in 2002
• MVAPICH2-X (MPI + PGAS), Available since 2011
• Support for GPGPUs (MVAPICH2-GDR) and MIC (MVAPICH2-MIC), Available since 2014
• Support for Virtualization (MVAPICH2-Virt), Available since 2015
• Support for Energy-Awareness (MVAPICH2-EA), Available since 2015
• Support for InfiniBand Network Analysis and Monitoring (OSU INAM) since 2015
• Used by more than 3,025 organizations in 89 countries
• More than 562,000 (> 0.5 million) downloads from the OSU site directly
• Empowering many TOP500 clusters (Nov ‘18 ranking)
  • 3rd ranked 10,649,640-core cluster (Sunway TaihuLight) at NSC, Wuxi, China
  • 5th, 448,448 cores (Frontera) at TACC
  • 8th, 391,680 cores (ABCI) in Japan
  • 15th, 570,020 cores (Neurion) in S. Korea and many others
• Available with software stacks of many vendors and Linux Distros (RedHat, SuSE, and OpenHPC)
• http://mvapich.cse.ohio-state.edu

Empowering Top500 systems for over a decade

Partner in TACC Frontera System
MVAPICH2 and TAU

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

- Takes a “black box” view of the MPI library
MPI_T usage semantics

```
int MPI_T_init_thread(int required, int *provided);
int MPI_T_cvar_get_num(int *num_cvar);
int MPI_T_cvar_get_info(int cvar_index, char *name, int *name_len, int *verbosity,
                        MPI_Datatype *datatype, MPI_T_enum *enumtype,
                        char *desc, int *desc_len, int *bind, int *scope);
int MPI_T_pvar_session_create(MPI_T_pvar_session *session);
int MPI_T_pvar_handle_alloc(MPI_T_pvar_session *session, int pvar_index,
                            void *obj_handle, MPI_T_pvar_handle *handle, int *count);
int MPI_T_pvar_start(MPI_T_pvar_session *session, MPI_T_pvar_handle *handle);
int MPI_T_pvar_read(MPI_T_pvar_session *session, MPI_T_pvar_handle *handle,
                    void *buf);
int MPI_T_pvar_reset(MPI_T_pvar_session *session, MPI_T_pvar_handle *handle);
int MPI_T_pvar_handle_free(MPI_T_pvar_session *session, MPI_T_pvar_handle *handle);
int MPI_T_pvar_session_free(MPI_T_pvar_session *session);
int MPI_T_finalize(void);
```
MPI_T support with MVAPICH2

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

<table>
<thead>
<tr>
<th>Memory Usage:</th>
<th>InfiniBand N/W:</th>
<th>Pt-to-pt messages:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- current level</td>
<td>- #control packets</td>
<td>- unexpected queue length</td>
</tr>
<tr>
<td>- maximum watermark</td>
<td>- #out-of-order packets</td>
<td>- unexp. match attempts</td>
</tr>
<tr>
<td>Registration cache:</td>
<td>Shared-memory:</td>
<td>Collective ops:</td>
</tr>
<tr>
<td>- hits</td>
<td>- limic/ CMA</td>
<td>- comm. creation</td>
</tr>
<tr>
<td>- misses</td>
<td>- buffer pool size &amp; usage</td>
<td>- #algorithm invocations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Bcast – 8; Gather – 10]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
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
enter/exit events are “interval” events

application-wide performance data

per thread performance

per process performance

(in shared memory)
**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 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
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 MEBDES PROTECT BELOW</td>
<td>off</td>
</tr>
<tr>
<td>TAU MEBDES PROTECT FREE</td>
<td>off</td>
</tr>
<tr>
<td>TAU MPIT_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_MULTIPLYING</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 SHARP 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 MPIT_ENABLE_SHARP 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 MPIT_ENABLE_SHARP in MVAPICH version 2.3a and above</td>
</tr>
<tr>
<td>TAU RECOMMENDATION PHASE_INIT</td>
<td>MPI T RECOMMEND SHARP USAGE: No performance benefit foreseen with SHARP 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 MPIT_ENABLE_SHARP 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_QDB</td>
<td>off</td>
</tr>
<tr>
<td>TAU THROTTLE</td>
<td>on</td>
</tr>
<tr>
<td>TAU THROTTLE NUMCALLS</td>
<td>1.00000</td>
</tr>
<tr>
<td>TAU THROTTLE PERCALL</td>
<td>1.0</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 un instrumented 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>File</th>
<th>Options</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td>Standard Applications</td>
<td>Default Exp</td>
</tr>
<tr>
<td></td>
<td>Default Exp</td>
<td>liushen.ppk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default (pdbc.h2/home)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI PVAR10</td>
<td>mem_allocated: Current level of allocated memory within the MPI library</td>
</tr>
<tr>
<td>MPI PVAR11</td>
<td>mv2_num_2level_comm_success: Number of successful 2-level comm creations</td>
</tr>
<tr>
<td>MPI PVAR12</td>
<td>mv2_num_shmem_colls: Number of times MV2 shared-memory collective calls were invoked</td>
</tr>
<tr>
<td>MPI PVAR13</td>
<td>mpt_progress_poll: CH3 RDMA engine polling count.</td>
</tr>
<tr>
<td>MPI PVAR14</td>
<td>mv2_smp_read_progress_poll: CH3 SMP read progress engine polling count.</td>
</tr>
<tr>
<td>MPI PVAR15</td>
<td>mv2_smp_write_progress_poll: CH3 SMP write progress engine polling count.</td>
</tr>
<tr>
<td>MPI PVAR16</td>
<td>mv2_smp_read_progress_poll_success: Successful CH3 SMP read progress engine polling count.</td>
</tr>
<tr>
<td>MPI PVAR17</td>
<td>mv2_smp_write_progress_poll_success: Successful CH3 SMP write progress engine polling count.</td>
</tr>
<tr>
<td>MPI PVAR18</td>
<td>rdma_ud_retransmissions: CH3 RDMA UD retransmission count.</td>
</tr>
<tr>
<td>MPI PVAR19</td>
<td>mv2_coll_bcast_bimodal: Number of times MV2 bimodalicast algorithm was invoked</td>
</tr>
<tr>
<td>MPI PVAR20</td>
<td>mv2_coll_bcast_scatter-doubling_all: Number of times MV2 scatter-doubling allgather algorithm was invoked</td>
</tr>
<tr>
<td>MPI PVAR21</td>
<td>mv2_coll_bcast_scatter_ring_all: Maximum level of memory ever allocated within the MPI library</td>
</tr>
<tr>
<td>MPI PVAR22</td>
<td>mv2_coll_bcast_scatter_ring_algorithm: Number of times MV2 scatter-ring allgather algorithm was invoked</td>
</tr>
<tr>
<td>MPI PVAR23</td>
<td>mv2_coll_bcast_scatter_ring_algorithm: Number of times MV2 scatter-ring allgather algorithm was invoked</td>
</tr>
<tr>
<td>MPI PVAR24</td>
<td>mv2_coll_bcast_kromial_internode: Number of times MV2 kromial internodeicast algorithm was invoked</td>
</tr>
<tr>
<td>MPI PVAR25</td>
<td>mv2_coll_bcast_mcast_internode: Number of times MV2 mcast internodeicast algorithm was invoked</td>
</tr>
<tr>
<td>MPI PVAR26</td>
<td>mv2_coll_bcast_detailed: Number of times MV2 pipeline broadcast algorithm was invoked</td>
</tr>
<tr>
<td>MPI PVAR27</td>
<td>mv2_coll_alltoall_inplace: Number of times MV2 in-place alltoall algorithm was invoked</td>
</tr>
<tr>
<td>MPI PVAR28</td>
<td>mv2_coll_alltoall_bruck: Number of times MV2 bruck alltoall algorithm was invoked</td>
</tr>
<tr>
<td>MPI PVAR29</td>
<td>mv2_coll_alltoall_bruck: Number of times MV2 recursive doubling alltoall algorithm was invoked</td>
</tr>
<tr>
<td>MPI PVAR30</td>
<td>mpt_malocso_calls: Number of MPT malocso calls.</td>
</tr>
<tr>
<td>MPI PVAR31</td>
<td>mpt_malocso_alltoall: Number of times MPT alltoall algorithm was invoked</td>
</tr>
<tr>
<td>MPI PVAR32</td>
<td>mpt_malocso_pairwise Alltoall algorithm was invoked.</td>
</tr>
<tr>
<td>MPI PVAR33</td>
<td>mpt_malocso_pairwise Alltoall algorithm was invoked.</td>
</tr>
<tr>
<td>MPI PVAR34</td>
<td>mpt_malocso_pairwise Alltoall algorithm was invoked.</td>
</tr>
<tr>
<td>MPI PVAR35</td>
<td>mpt_malocso_pairwise Alltoall algorithm was invoked.</td>
</tr>
<tr>
<td>MPI PVAR36</td>
<td>mpt_malocso_pairwise Alltoall algorithm was invoked.</td>
</tr>
<tr>
<td>MPI PVAR37</td>
<td>mpt_malocso_pairwise Alltoall algorithm was invoked.</td>
</tr>
<tr>
<td>MPI PVAR38</td>
<td>mpt_malocso_pairwise Alltoall algorithm was invoked.</td>
</tr>
<tr>
<td>MPI PVAR39</td>
<td>mpt_malocso_pairwise Alltoall algorithm was invoked.</td>
</tr>
<tr>
<td>MPI PVAR40</td>
<td>num_colls: Number of MPT colls.</td>
</tr>
<tr>
<td>MPI PVAR41</td>
<td>reg_cache_hits: Number of registration cache hits.</td>
</tr>
<tr>
<td>MPI PVAR42</td>
<td>reg_cache_misses: Number of registration cache misses.</td>
</tr>
<tr>
<td>MPI PVAR43</td>
<td>vbuf_free_attempts: Number of VBUFs allocated.</td>
</tr>
<tr>
<td>MPI PVAR44</td>
<td>vbuf_free_attempts: Number of VBUFs freed.</td>
</tr>
<tr>
<td>MPI PVAR45</td>
<td>vbuf_free_attempts: Number of VBUFs allocated.</td>
</tr>
<tr>
<td>MPI PVAR46</td>
<td>vbuf_free_attempts: Number of VBUFs freed.</td>
</tr>
<tr>
<td>MPI PVAR47</td>
<td>vbuf_free_attempts: Number of VBUFs allocated.</td>
</tr>
<tr>
<td>MPI PVAR48</td>
<td>vbuf_free_attempts: Number of VBUFs freed.</td>
</tr>
<tr>
<td>MPI PVAR49</td>
<td>vbuf_free_attempts: Number of VBUFs allocated.</td>
</tr>
<tr>
<td>MPI PVAR50</td>
<td>vbuf_free_attempts: Number of VBUFs freed.</td>
</tr>
<tr>
<td>CVARs Exposed by MVAPICH2</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Local Time</strong></td>
<td><strong>TriField</strong></td>
</tr>
<tr>
<td>2016-06-16T10:11:04-07:00</td>
<td>35</td>
</tr>
</tbody>
</table>

**CVARs Description**

- **MVP_RCV рын Name**
  - Carneus.m.nic.us:sawr.com

- **MVP_CVAR_ABORT_ON_LEAKED_HANDLES**
  - if true, MPL will call MVP_ABORT at MVP_finalize if any MVP object handles have been leaked. For example...

- **MVP_CVAR_ALLGATHERV_PIPELINE_MSG_SIZE**
  - The smallest message size that will be used for the pipelined, large-message, ring algorithm in the MVP_ALLGATHERV operation.

- **MVP_CVAR_ALLGATHER_LONG_MSG_SIZE**
  - If MVP_ALLGATHER and MVP_ALLGATHERV are used, the long message algorithm will be used if the send buffer size is <= 32768.

- **MVP_CVAR_ALLGATHER_SHORT_MSG_SIZE**
  - For MVP_ALLGATHER and MVP_ALLGATHERV, the short message algorithm will be used if the send buffer size is <= 32768.

- **MVP_CVAR_ALLREDUCE_SHORT_MSG_SIZE**
  - If MVP_ALLREDUCE and MVP_ALLREDUCEV are used, the short message algorithm will be used if the send buffer size is <= 32768.

- **MVP_CVAR_ALLREDUCE_LONG_MSG_SIZE**
  - If MVP_ALLREDUCE and MVP_ALLREDUCEV are used, the long message algorithm will be used if the send buffer size is <= 32768.

- **MVP_CVAR_ALLGATHERV_PIPELINE_MIN_PROCS**
  - The minimum number of processes that will be used if the message size is <= 32768.

- **MVP_CVAR_ALLGATHERV_PIPELINE_MAX_PROCS**
  - The maximum number of processes that will be used if the message size is <= 32768.

- **MVP_CVAR_ALLREDUCE_PIPELINE_MIN_PROCS**
  - The minimum number of processes that will be used if the message size is <= 32768.

- **MVP_CVAR_ALLREDUCE_PIPELINE_MAX_PROCS**
  - The maximum number of processes that will be used if the message size is <= 32768.

- **MVP_CVAR_ASYNC_PROGRESS**
  - If set to true, the MVP operation will execute asynchronously.

- **MVP_CVAR_BCAST_LONG_MSG_SIZE**
  - The smallest message size that will be used for the pipelined, large-message, ring algorithm in the MVP_BCAST operation.

- **MVP_CVAR_BCAST_MIN_PROCS**
  - The minimum number of processes that will be used if the message size is <= 32768.

- **MVP_CVAR_BCAST_SHORT_MSG_SIZE**
  - If MVP_BCAST and MVP_BCAST_EX are used, the short message algorithm will be used if the send buffer size is <= 32768.

- **MVP_CVAR_CH3_EAGER_MSG_MIN_SIZE**
  - This CVAR controls the minimum message size at CH3 switches to eager from rendezvous mode.

- **MVP_CVAR_CH3_EAGER_MSG_MAX_SIZE**
  - If true, enable HCCollective.

- **MVP_CVAR_CH3_INTERFACE_HOSTNAME**
  - If non-null, this CVAR specifies the IP address that other processes should use when connecting to this pr.

- **MVP_CVAR_CH3_NOLOCAL**
  - If true, force all processes to operate as though all processes are located on another node. For example...

- **MVP_CVAR_CH3_0DD_EVEN_CUIQUES**
  - Enable checking of aliasing in collective operations.

- **MVP_CVAR_CH3_PORT_RANGE**
  - The MVP_CVAR_CH3_PORT_RANGE environment variable allows you to specify the range of TCP ports...

- **MVP_CVAR_CH3_RMA_RECV_COMPLECTED**
  - Threshold for the number of completed requests the runtime finds before it stops trying to find more co.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Threshold for the number of MPI requests the runtime finds before it stops trying to check more req.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Issue a request for the passive target RMA lock immediately. Default behavior is to defer the lock reqe.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Enable/disable an optimization that merges lock, op, and unlock messages for single-operation passive ta.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Threshold for the number of new requests since the last attempt to complete pending requests. Higher...

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - High water mark for the RMA implementation attempts to complete requests while completing RMA oper.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Disable the optimization of merging lock, op, and unlock messages for single-operation passive ta.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Issue a request for the passive target RMA lock immediately. Default behavior is to defer the lock reqe.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Enable/disable an optimization that merges lock, op, and unlock messages for single-operation passive ta.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Threshold for the number of new requests since the last attempt to complete pending requests. Higher...

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - High water mark for the RMA implementation attempts to complete requests while completing RMA oper.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Enable/disable an optimization that merges lock, op, and unlock messages for single-operation passive ta.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Threshold for the number of new requests since the last attempt to complete pending requests. Higher...

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - High water mark for the RMA implementation attempts to complete requests while completing RMA oper.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Disable the optimization of merging lock, op, and unlock messages for single-operation passive ta.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Issue a request for the passive target RMA lock immediately. Default behavior is to defer the lock reqe.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Enable/disable an optimization that merges lock, op, and unlock messages for single-operation passive ta.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Threshold for the number of new requests since the last attempt to complete pending requests. Higher...

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - High water mark for the RMA implementation attempts to complete requests while completing RMA oper.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Disable the optimization of merging lock, op, and unlock messages for single-operation passive ta.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Issue a request for the passive target RMA lock immediately. Default behavior is to defer the lock reqe.

- **MVP_CVAR_CH3_RMA_RECV_COMPLETED**
  - Enable/disable an optimization that merges lock, op, and unlock messages for single-operation passive ta.
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_alloc (Number of MPITalloc calls)</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>0</td>
<td>1</td>
<td>89</td>
</tr>
<tr>
<td>num_free (Number of MPIT_free calls)</td>
<td>47,801</td>
<td>47,801</td>
<td>47,801</td>
<td>0</td>
<td>1</td>
<td>47,801</td>
</tr>
<tr>
<td>num_malloc (Number of MPIT_malloc calls)</td>
<td>49,258</td>
<td>49,258</td>
<td>49,258</td>
<td>0</td>
<td>1</td>
<td>49,258</td>
</tr>
<tr>
<td>num_memalign (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 (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,OMPI_CVAR_GPUDIRECT_LIMIT,2097152,COMB_MPI_CART_COMM,OMPI_CVAR_USE_GPUDIRECT_RECEIVE_LIMIT,2097152,COMB_MPI_CART_COMM,OMPI_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

The Ohio State University
## CVARs Exposed by MVAPICH2

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name Processor Name</td>
<td>lassen710</td>
</tr>
<tr>
<td>MPIR_CVAR_CUDA_IPC_THRESHOLD</td>
<td>16384</td>
</tr>
<tr>
<td>MPIR_CVAR_GPUDIRECT_LIMIT</td>
<td>2097152</td>
</tr>
<tr>
<td>MPIR_CVAR_USE_GPUDIRECT_RECEIVE_LIMIT</td>
<td>2097152</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_ABORT_ON_LEAKED_HANDLES</td>
<td>If true, MPI will call MPI_Finalize if any MPI object handles ha...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_ALLGATHER_PIPELINE_MSG_SIZE</td>
<td>The smallest message size that will be used for the pipelined, large-mes...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_ALLGATHER_COLLECTIVE_ALGORITHM</td>
<td>This CVAR selects proper collective algorithm for allgather operation. For MPIT_Allgather and MPIT_Allgatherv, the long message algorithm will be...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_ALLGATHER_LONG_MSG_SIZE</td>
<td>For MPIT_Allgatherv and MPIT_Allgatherv, the short message algorithm will b...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_ALLREDUCE_COLLECTIVE_ALGORITHM</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>MPIT_CVAR: MPIR_CVAR_ALLREDUCE_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>MPIT_CVAR: MPIR_CVAR_ALLTOALL_COLLECTIVE_ALGORITHM</td>
<td>This CVAR selects proper collective algorithms for alltoall operations. The medium message algorithm will be used if the per-destination message...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_ALLTOALL_MEDIUM_MSG_SIZE</td>
<td>The short message algorithm will be used if the per-destination message... max no. of irecs/sends posted at a time in some alltoall algorithms. Set...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_ALLTOALL_THROTTLE</td>
<td>If set to true, MPICH will initiate an additional thread to make asynchrono...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_ASYNC_PROGRESS</td>
<td>This CVAR selects proper collective algorithm for broadcast operations. Let's define short messages as messages with size &lt; MPIR_CVAR_BCAST_MIN_PROCS...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_BCAST_COLLECTIVE_ALGORITHM</td>
<td>Let's define short messages as messages with size &lt; MPIR_CVAR_BCAST_MIN_PROCS...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_BCAST_LONG_MSG_SIZE</td>
<td>Let's define short messages as messages with size &lt; MPIR_CVAR_BCAST_MIN_PROCS...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_BCAST_SHORT_MSG_SIZE</td>
<td>This cvar controls the message size at which Ch3 switches from eager to...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_EAGER_MAX_MSG_SIZE</td>
<td>If true, enable HCOLL collects...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_ENABLE_HCOLL</td>
<td>If true, enable HCOLL collects...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_ENABLE_HCOLL</td>
<td>If true, enable HCOLL collects...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_INTERFACE_HOSTNAME</td>
<td>If true, all processes to operate as though all processes are located...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_NOLOCAL</td>
<td>If true, odd processes on a node are seen as local to each other, and even pr...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_ODD_EVEN_CLiques</td>
<td>The MPIT_CVAR_CH3_PORT_RANGE environment variable allows you to s...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_RMA_ACTIVE_REQ_THRESHOLD</td>
<td>Threshold of number of active requests to trigger blocking waiting in op...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_DELAY_ISSUING_FOR_PIGGYBACKING</td>
<td>Specify if delay issuing of RMA operations for piggybacking...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_RMA_DELAY_ISSUING_FOR_PIGGYBACKING</td>
<td>Size of the Global RMA operations pool (in number of operations) that st...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_RMA_OPS_GLOBAL_POOL_SIZE</td>
<td>Specify the threshold of data size of a RMA operation which can be piggy...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_RMA_OPS_PIGGYBACK_LOCK_DATA_SIZE</td>
<td>Size of the window-private RMA operations pool (in number of operation...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_RMA_PKE_POKE_THRESHOLD</td>
<td>Specify the threshold of switching the algorithm used in FENCE from the...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_RMA_PKE_POKE_RECE_THRESHOLD</td>
<td>Number of RMA slots during window creation. Each slot contains a linked...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_RMA_PKE_POKE_RECE_THRESHOLD</td>
<td>Size of the Global RMA targets pool (in number of targets) that stores inf...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_RMA_SLOTS_SIZE</td>
<td>Size in bytes of available lock data this window can provided. If current ...</td>
</tr>
<tr>
<td>MPIT_CVAR: MPIR_CVAR_CH3_RMA_TARGET_GLOBAL_POOL_SIZE</td>
<td>Size of the window–private RMA lock entries pool (in number of lock entri...</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, Containers for HPC]

for more information

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

www.uoregon.edu
Support Acknowledgements

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

Department of Defense (DoD)
- PETTT, HPCMP

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

NASA

CEA, France

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