Performance Evaluation using the TAU Performance System

Sameer Shende
*University of Oregon*

August 17, 2016, 11am

Ohio Supercomputing Center, The Ohio State University
Columbus, OH
Acknowledgments

• Hari Subramoni, OSU
• Khaled Hamidouche
• D.K. Panda, OSU
• Allen D. Malony, UO
• Auréle Maheo, UO
• Srinivasan Ramesh, UO
• Wyatt Spear, UO
• Soren Rasmussen, UO
• Kevin Huck, UO
TAU Performance System®

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

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

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

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

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

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

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

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

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

• **How can I tune MPI for better performance**? What performance and control does MVAPICH2 export to observe and control its performance?
MVAPICH2

- High performance open-source MPI for InfiniBand, 10-40Gig/iWARP, and RoCE
- MVAPICH (MPI-1), MVAPICH2 (MPI-2.2 and MPI-3.0), Available since 2002
- MPI, OpenSHMEM, UPC, CAF or Hybrid (MPI + PGAS) Applications
- OpenSHMEM Calls CAF Calls UPC MPI Calls Calls
- Unified MVAPICH2-X Runtime InfiniBand, RoCE, iWARP
- Unified communication runtime for MPI, UPC, OpenSHMEM, CAF Available with MVAPICH2-X 1.9 (2012) onwards!
- http://mvapich.cse.ohio-state.edu
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
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
Non-Interactive Mode

- Performance data collected by TAU
  - Support for PVARs and CVARs
  - Setting CVARs to control MVAPICH2
  - Studying performance data in TAU’s ParaProf profile browser
Enhancing MPI_T Support

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

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

- Several new MPI_T based PVARs added to MVAPICH2
  - mv2_vbuf_max_use, mv2_total_vbuf_memory etc
- Enhanced TAU with support for tracking of MPI_T PVARs and CVARs for uninstrumented applications
  - ParaProf, TAU’s visualization front end, enhanced with support for displaying PVARs and CVARs
  - TAU provides tau_exec, a tool to transparently instrument MPI routines
    - Uninstrumented:
      - % mpirun –np 1024 ./a.out
    - Instrumented:
      - % mpirun –np 1024 tau_exec [options] ./a.out
      - % paraprof
## PVARs Exposed by MVAPICH2

<table>
<thead>
<tr>
<th>TrialField</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_T PVAR[0]: mem_allocated</td>
<td>Current level of allocated memory within the MPI library</td>
</tr>
<tr>
<td>MPI_T PVAR[10]: mv2_num_2level_comm_success</td>
<td>Number of successful 2-level comm creations</td>
</tr>
<tr>
<td>MPI_T PVAR[11]: mv2_num_shmem_coll_calls</td>
<td>Number of times MV2 shared-memory collective calls were invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[12]: mpi_progress_poll</td>
<td>CH3 RDMA progress engine polling count</td>
</tr>
<tr>
<td>MPI_T PVAR[13]: mv2_smp_read_progress_poll</td>
<td>CH3 SMP read progress engine polling count</td>
</tr>
<tr>
<td>MPI_T PVAR[14]: mv2_smp_write_progress_poll</td>
<td>CH3 SMP write progress engine polling count</td>
</tr>
<tr>
<td>MPI_T PVAR[15]: mv2_smp_read_progress_poll_success</td>
<td>Unsuccessful CH3 SMP read progress engine polling count</td>
</tr>
<tr>
<td>MPI_T PVAR[16]: mv2_smp_write_progress_poll_success</td>
<td>Unsuccessful CH3 SMP write progress engine polling count</td>
</tr>
<tr>
<td>MPI_T PVAR[17]: rdma_ud_retransmissions</td>
<td>CH3 RDMA UD retransmission count</td>
</tr>
<tr>
<td>MPI_T PVAR[18]: mv2_coll_bcast_bimomial</td>
<td>Number of times MV2 bimomial bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[19]: mv2_coll_bcast_scatter_doubling_all</td>
<td>Number of times MV2 scatter+double allgather bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[20]: mv2_coll_bcast_scatter_ring_allgather</td>
<td>Number of times MV2 scatter+ring allgather bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[21]: mv2_coll_bcast_scatter_ring_allgather</td>
<td>Number of times MV2 scatter+ring allgather shmem bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[22]: mv2_coll_bcast_shmem</td>
<td>Number of times MV2 shmem bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[23]: mv2_coll_bcast_knomial_internode</td>
<td>Number of times MV2 knomial internode bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[24]: mv2_coll_bcast_normal_internode</td>
<td>Number of times MV2 knomial intranode bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[25]: mv2_coll_bcast_mcst_internode��</td>
<td>Number of times MV2 mcst internode bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[26]: mv2_coll_bcast_pipelined</td>
<td>Number of times MV2 pipelined bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[27]: mv2_coll_alltoall_inplace</td>
<td>Number of times MV2 in-place alltoall algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[28]: mv2_coll_alltoall_bruck</td>
<td>Number of times MV2 brucks alltoall algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[29]: mv2_coll_alltoall_rd</td>
<td>Number of times MV2 recursive-doubling alltoall algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[30]: mv2_coll_alltoall_sd</td>
<td>Number of times MV2 scatter-destination alltoall algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[31]: mv2_coll_alltoall_pw</td>
<td>Number of times MV2 pairwise alltoall algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[32]: mpi_alltoallv_mv2_pw</td>
<td>Number of times MV2 pairwise alltoallv algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[33]: mv2_coll_allreduce_shm_rd</td>
<td>Number of times MV2 shm rd allreduce algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[34]: mv2_coll_allreduce_shm_rs</td>
<td>Number of times MV2 shm rs allreduce algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[35]: mv2_coll_allreduce_shm_intra��</td>
<td>Number of times MV2 shm intra allreduce algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[36]: mv2_coll_allreduce_p2p��</td>
<td>Number of times MV2 p2p allreduce algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[37]: mv2_coll_allreduce_2lv</td>
<td>Number of times MV2 two-level allreduce algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[38]: mv2_coll_allreduce_shmem</td>
<td>Number of times MV2 shmem allreduce algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[39]: mv2_coll_allreduce_mcst</td>
<td>Number of times MV2 multicast-based allreduce algorithm was invoked</td>
</tr>
<tr>
<td>MPI_T PVAR[40]: num_maloc_calls</td>
<td>Number of MPI图书馆 maloc calls</td>
</tr>
<tr>
<td>MPI_T PVAR[41]: mv2_reg_cache_hits</td>
<td>Number of registration cache hits</td>
</tr>
<tr>
<td>MPI_T PVAR[42]: mv2_reg_cache_misses</td>
<td>Number of registration cache misses</td>
</tr>
<tr>
<td>MPI_T PVAR[43]: mv2_vbuf_allocated_array</td>
<td>Number of VBUFs allocated</td>
</tr>
<tr>
<td>MPI_T PVAR[44]: mv2_vbuf_freed</td>
<td>Number of VBUFs freed</td>
</tr>
<tr>
<td>MPI_T PVAR[45]: mv2_ud_vbuf_allocated</td>
<td>Number of UD VBUFs allocated</td>
</tr>
<tr>
<td>MPI_T PVAR[46]: mv2_ud_vbuf_freed</td>
<td>Number of UD VBUFs freed</td>
</tr>
<tr>
<td>MPI_T PVAR[47]: mv2_vbuf_free_attempts</td>
<td>Number of times we attempted to free VBUFs</td>
</tr>
<tr>
<td>MPI_T PVAR[48]: mv2_vbuf_free_attempt_success_time</td>
<td>Average time for number of times we successfully freed VBUFs</td>
</tr>
<tr>
<td>MPI_T PVAR[49]: mv2_vbuf_free_attempt_success_time</td>
<td>Average time for number of times we successfully freed VBUFs</td>
</tr>
<tr>
<td>MPI_T PVAR[50]: num_memalign_calls</td>
<td>Number of MPI图书馆 memalign calls</td>
</tr>
<tr>
<td>MPI_T PVAR[51]: mv2_vbuf_allocate_time</td>
<td>Average time for number of times we allocated VBUFs</td>
</tr>
<tr>
<td>MPI_T PVAR[52]: mv2_vbufAllocate_time</td>
<td>Average time for number of times we allocated VBUFs</td>
</tr>
</tbody>
</table>

**Performance Evaluation using the TAU Performance System**
### CVARs Exposed by MVAPICH2

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Local Time</strong></td>
<td>2016-06-16T11:04:07-07</td>
</tr>
</tbody>
</table>
Using MVAPICH2 and TAU

- To set CVARs or read PVARs using TAU for an uninstrumented binary:
  
  ```
  % export TAU_TRACK_MPI_T_PVARS=1
  % export TAU_MPI_T_CVAR_METRICS=
      MPIR_CVAR_VBUF_POOL_REduced_VALUE[1],
      MPIR_CVAR_IBA_EAGER_THRESHOLD
  % export TAU_MPI_T_CVAR_VALUES=32,64000
  % export PATH=/path/to/tau/x86_64/bin:$PATH
  % mpirun -np 1024 tau_exec -T mvapich2 ./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_calloc_calls (Number of MPIT_calloc 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_calls (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_calls (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_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

Performance Evaluation using the TAU Performance System
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
Examples
Simplifying the use of TAU!

Uninstrumented code:

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

With TAU:

- % mpirun -np 16 tau_exec ./a.out
- % paraprof
- For more Information at the statement level:
  - % mpirun -np 16 tau_exec --ebs ./a.out (or use TAU_SAMPLING=1)
- To rewrite the binary to instrument individual functions (using MAQAO):
  - % tau_rewrite a.out a.inst; mpirun -np 16 ./a.inst (beta)
- % pprof -a | more
- % paraprof (GUI)
TAU for Heterogeneous Measurement

Multiple performance perspectives
Integrate Host-GPU support in TAU measurement framework

• Enable use of each measurement approach
• Include use of PAPI and CUPTI
• Provide profiling and tracing support

Tutorial

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

Uninstrumented execution

• % mpirun -np 256 ./a.out

Track GPU operations

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

Track MPI performance

• % mpirun -np 256 tau_exec ./a.out

Track OpenMP, I/O, and MPI performance (MPI enabled by default)

• % mpirun -np 256 tau_exec –ompt -io ./a.out

Track memory operations

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

Use event based sampling (compile with –g)

• % mpirun –np 256 tau_exec –ebs ./a.out
• Also –ebs_source=<PAPI_COUNTER> -ebs_period=<overflow_count>
Using TAU

TAU supports several measurement and thread options

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

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

To instrument source code automatically using PDT

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

% export TAU_MAKEFILE=$TAU/Makefile.tau-papi-mpi-pdt
% export TAU_OPTIONS='"-optVerbose ..."' (see tau_compiler.sh)
% export PATH=$TAUDIR/x86_64/bin:$PATH

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

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

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

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

% ls $TAU/Makefile.*
Makefile.tau-mpi-pdt
Makefile.tau-papi-mpi-pdt
Makefile.tau-beacon-papi-mpi-pdt-mpit
Makefile.tau-icpc-papi-mpi-pdt
Makefile.tau-icpc-papi-ompt-mpi-pdt-openmp
Makefile.tau-mpi-pdt-openmp-opari
Makefile.tau-mpi-pthread-python-pdt
Makefile.tau-papi-mpi-pdt-openmp-opari-scorep
Makefile.tau-papi-mpi-pdt-scorep
Makefile.tau-papi-mpi-pthread-pdt
Makefile.tau-papi-pthread-pdt

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

Makefile.tau-papi-mpi-pdt

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

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

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

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

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

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

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

Application / Library

C / C++ parser

Fortran parser F77/90/95

IL analyzer

Fortran IL analyzer

Program Database Files

DUCTAPE

TAU instrumentor

Automatic source instrumentation
Automatic Source Instrumentation using PDT

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

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

**Before**

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

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

% make
```

**After**

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

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

% export TAU_MAKEFILE=
    $TAU/Makefile.tau-papi-mpi-pdt
% make
```
Selective Instrumentation File

```plaintext
% export TAU_OPTIONS='"-optTauSelectFile=select.tau ..."'
% cat select.tau
BEGIN_INCLUDE_LIST
int main#
int dgemm#
END_INCLUDE_LIST
BEGIN_FILE_INCLUDE_LIST
Main.c
Blas/*.f77
END_FILE_INCLUDE_LIST
# replace include with exclude list
BEGIN_INSTRUMENT_SECTION
loops routine="foo"
loops routine="int main#"
END_INSTRUMENT_SECTION
```
Installing and Configuring TAU

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

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

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

Performance Evaluation using the TAU Performance System
## Compile-Time Options

Optional parameters for the TAU_OPTIONS environment variable:

```bash
% tau_compiler.sh
```

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

Optional parameters for the TAU_OPTIONS environment variable:

% tau_compiler.sh

- optShared
  Use TAU’s shared library (libTAU.so) instead of static library (default)
- optPdtCxxOpts=""
  Options for C++ parser in PDT (cxxparse).
- optPdtF90Parser=""
  Specify a different Fortran parser
- optPdtCleanscapeParser
  Specify the Cleanscape Fortran parser instead of GNU gfparser
- optTau=""
  Specify options to the tau_instrumentor
- optTrackDMAPP
  Enable instrumentation of low-level DMAPP API calls on Cray
- optTrackPthread
  Enable instrumentation of pthread calls

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

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

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

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

• TAU uses Beacon (backplane for event and control notification) to observe the performance of a running application (for e.g., vbuf pool statistics, high water mark of total and vbuf memory usage, message size statistics).
# Other Runtime Environment Variables

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU_TRACE</td>
<td>0</td>
<td>Setting to 1 turns on tracing</td>
</tr>
<tr>
<td>TAU_CALLPATH</td>
<td>0</td>
<td>Setting to 1 turns on callpath profiling</td>
</tr>
<tr>
<td>TAU.Track MEMORY FOOTPRINT</td>
<td>0</td>
<td>Setting to 1 turns on tracking memory usage by sampling periodically the resident set size and high water mark of memory usage</td>
</tr>
<tr>
<td>TAU.Track POWER</td>
<td>0</td>
<td>Tracks instantaneous power usage by sampling periodically.</td>
</tr>
<tr>
<td>TAU_CALLPATH_DEPTH</td>
<td>2</td>
<td>Specifies depth of callpath. Setting to 0 generates no callpath or routine information, setting to 1 generates flat profile and context events have just parent information (e.g., Heap Entry: foo)</td>
</tr>
<tr>
<td>TAU_SAMPLING</td>
<td>0</td>
<td>Setting to 1 enables event-based sampling.</td>
</tr>
<tr>
<td>TAU.Track SIGNALS</td>
<td>0</td>
<td>Setting to 1 generate debugging callstack info when a program crashes</td>
</tr>
<tr>
<td>TAU.COMM_MATRIX</td>
<td>0</td>
<td>Setting to 1 generates communication matrix display using context events</td>
</tr>
<tr>
<td>TAU.THROTTLE</td>
<td>1</td>
<td>Setting to 0 turns off throttling. Enabled by default to remove instrumentation in lightweight routines that are called frequently</td>
</tr>
<tr>
<td>TAU.THROTTLE_NUMCALLS</td>
<td>100000</td>
<td>Specifies the number of calls before testing for throttling</td>
</tr>
<tr>
<td>TAU.THROTTLE_PERCALL</td>
<td>10</td>
<td>Specifies value in microseconds. Throttle a routine if it is called over 100000 times and takes less than 10 usec of inclusive time per call</td>
</tr>
<tr>
<td>TAU.COMPENSATE</td>
<td>0</td>
<td>Setting to 1 enables runtime compensation of instrumentation overhead</td>
</tr>
<tr>
<td>TAU_PROFILE_FORMAT</td>
<td>Profile</td>
<td>Setting to “merged” generates a single file. “snapshot” generates xml format</td>
</tr>
<tr>
<td>TAU_METRICS</td>
<td>TIME</td>
<td>Setting to a comma separated list generates other metrics. (e.g., TIME,ENERGY,PAPI_FP_INS,PAPI_NATIVE_&lt;event&gt;:_&lt;subevent&gt;)</td>
</tr>
</tbody>
</table>

Performance Evaluation using the TAU Performance System
<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU_TRACK_MEMORY_LEAKS</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>
Goal: What is the volume of inter-process communication? Along which calling path?
Evaluating Extent of Vectorization on MIC

Performance Evaluation using the TAU Performance System

% export TAU_MAKEFILE=$TAUROOT/mic_linux/lib/Makefile.tau-papi-mpi-pdt
% export TAU_METRICS=TIME,

PAPI_NATIVE_VPU_ELEMENTS_ACTIVE,PAPI_NATIVE_VPU_INSTRUCTIONS_EXECUTED
ParaProf’s Topology Display Window (BGQ)

Performance Evaluation using the TAU Performance System
ParaProf’s Scalable 3D Visualization (BGQ)

Performance Evaluation using the TAU Performance System

786,432 ranks
ParaProf 3D Profile Browser

Performance Evaluation using the TAU Performance System
Generating a Trace File Using Score-P

Goal: Identify the temporal aspect of performance. What happens in my code at a given time? When? Configure with –scorep=download (new in TAU v2.25.2)

Event trace visualized in Vampir [www.vampir.eu]
Download TAU from U. Oregon

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

Free download, open source, BSD license
Performance Evaluation using the TAU Performance System
Support Acknowledgments

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

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

Department of Defense (DoD)
- PETTT, HPCMP

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

Performance Evaluation using the TAU Performance System