Tracking Memory Usage in OpenSHMEM Runtimes with the TAU Performance System

Nicholas Chaimov (ParaTools)*, Sameer Shende (ParaTools), Allen Malony (ParaTools), Manjunath Gorentla Venkata (Oak Ridge National Laboratory), and Neena Imam (Oak Ridge National Laboratory)*

OpenSHMEM 2018

22 August 2018

Hanover, MD

ParaTools

Oak Ridge National Laboratory

Outline

- Motivation: per-object memory tracking
- What is TAU?
- Implementation
  - Library wrapping
  - Context events
  - Allocation classes
  - Profiling vs Tracing
  - Data analysis
- Preliminary results
- Future work and conclusions

Motivation

- Evaluate scalability of OpenSHMEM runtimes in terms of runtime memory usage as the number of PEs increases.
  - By keeping arrays of sizes proportional to the number of PEs, an OpenSHMEM implementation may be limited in its scalability to millions of PEs.
- Extend TAU to track memory allocations within OpenSHMEM runtimes.
  - Trigger atomic events with a value of memory usage from each PE.
  - Trigger **separate** events according to the **data type** of the allocated objects, allowing determination of scaling behavior for different runtime object types.
- Postprocess data to chart memory usage by object type as number of PEs grows.

The 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)
- Integrates with application frameworks

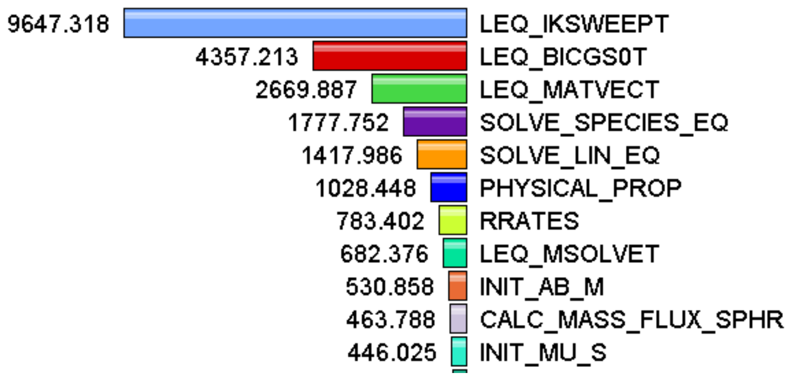
TAU Supports All HPC Platforms

C/C++ CUDA UPC Python GPI Fortran OpenACC Java MPI pthreads Intel MIC OpenMP Intel GNU PGI Cray Sun LLVM MinGW AIX Linux Windows BlueGene Fujitsu ARM Android MPC OpenSHMEM

Insert yours here

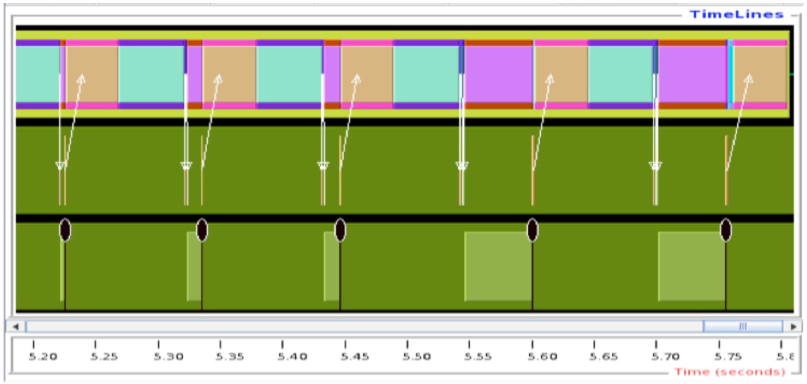
Measurement Approaches

Profiling



Shows **how much** time was spent in each routine

Tracing



Shows **when** events take place on a timeline

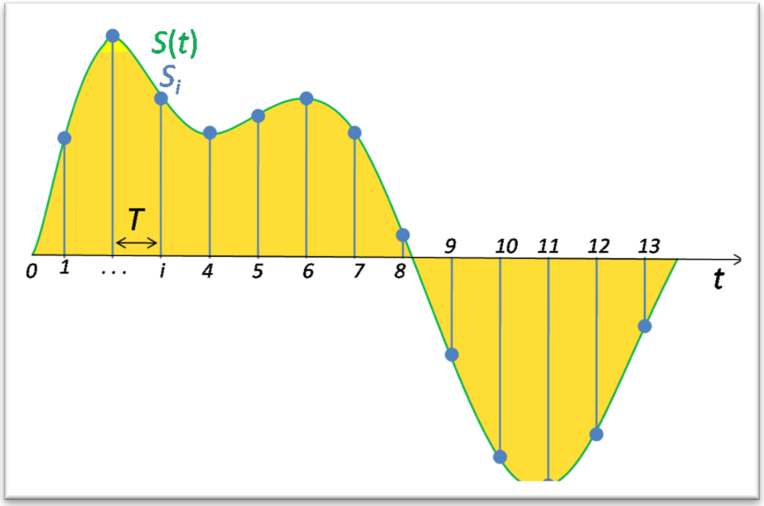
Performance Data Measurement

Direct via Probes

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

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

Indirect via Sampling



- No code modification
- Minimal effort
- Relies on debug symbols (**-g** option)

Questions TAU Can Answer

- **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, *vector instructions*?
- *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 **time spent waiting for collectives**?
- How does the application **scale**?

Atomic 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 (total, samples, min/max/mean/std. deviation statistics)
  - Context Events
    - Atomic event + context (disaggregated according to timer stack when event triggered)

Interval and Atomic Events in TAU

xterm

```
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.187        1,105           1          44    1105659 int main(int, char **) C
 93.2        1,030        1,030           1           0    1030654 MPI_Init()
  5.9        0.879           65          40         320       1637 void func(int, int) C
  4.6           51           51          40           0       1277 MPI_Barrier()
  1.2           13           13         120           0        111 MPI_Recv()
  0.8            9            9           1           0       9328 MPI_Finalize()
  0.0        0.137        0.137         120           0          1 MPI_Send()
  0.0        0.086        0.086          40           0          2 MPI_Bcast()
  0.0        0.002        0.002           1           0          2 MPI_Comm_size()
  0.0        0.001        0.001           1           0          1 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
       365  5.138E+04      44.39   3.09E+04  1.234E+04  Heap Memory Used (KB) : Entry
       365  5.138E+04       2064  3.115E+04   1.21E+04  Heap Memory Used (KB) : Exit
        40         40         40         40          0  Message size for broadcast
---------------------------------------------------------------------------------------
                                                                      27,1          1%
```

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

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

% export TAU_CALLPATH_DEPTH=0
% export TAU_TRACK_HEAP=1

Atomic Events, Context Events

xterm

```
--------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
--------------------------------------------------------------------------------------
100.0        0.253        1,106           1          44    1106701 int main(int, char **) C
 93.2        1,031        1,031           1           0    1031311 MPI_Init()
  6.0            1           66          40         320       1650 void func(int, int) C
  5.7           63           63          40           0       1588 MPI_Barrier()
  0.8            9            9           1           0       9119 MPI_Finalize()
  0.1            1            1         120           0         10 MPI_Recv()
  0.0        0.141        0.141         120           0          1 MPI_Send()
  0.0        0.085        0.085          40           0          2 MPI_Bcast()
  0.0        0.001        0.001           1           0          1 MPI_Comm_size()
  0.0            0            0           1           0          0 MPI_Comm_rank()
--------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
--------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
--------------------------------------------------------------------------------------
        40         40         40         40          0  Message size for broadcast
       365  5.139E+04      44.39  3.091E+04  1.234E+04  Heap Memory Used (KB) : Entry
        40  5.139E+04       3097  3.114E+04  1.227E+04  Heap Memory Used (KB) : Entry : MPI_Barrier()
        40  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : MPI_Bcast()
         1       2067       2067       2067          0  Heap Memory Used (KB) : Entry : MPI_Comm_rank()
         1       2066       2066       2066          0  Heap Memory Used (KB) : Entry : MPI_Comm_size()
         1  5.139E+04  5.139E+04  5.139E+04  0.0006905  Heap Memory Used (KB) : Entry : MPI_Finalize()
         1      57.56      57.56      57.56          0  Heap Memory Used (KB) : Entry : MPI_Init()
       120  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : MPI_Recv()
       120  5.139E+04  1.129E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : MPI_Send()
         1      44.39      44.39      44.39          0  Heap Memory Used (KB) : Entry : int main(int, char **) C
        40  5.036E+04       2068  3.011E+04  1.227E+04  Heap Memory Used (KB) : Entry : void func(int, int) C
                                                                             4,9            1%
```

Atomic events

Context events =atomic event + executing context

% export TAU_CALLPATH_DEPTH=1

Controls depth of executing context shown in profiles

% export TAU_TRACK_HEAP=1

Context Events (Default)

```
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.357        1,114           1          44    1114040 int main(int, char **) C
 92.6        1,031        1,031           1           0    1031066 MPI_Init()
  6.7           72           74          40         320       1865 void func(int, int) C
  0.7            8            8           1           0       8002 MPI_Finalize()
  0.1            1            1         120           0         12 MPI_Recv()
  0.1        0.608        0.608          40           0         15 MPI_Barrier()
  0.0        0.136        0.136         120           0          1 MPI_Send()
  0.0        0.095        0.095          40           0          2 MPI_Bcast()
  0.0        0.001        0.001           1           0          1 MPI_Comm_size()
  0.0            0            0           1           0          0 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
       365  5.139E+04      44.39  3.091E+04  1.234E+04  Heap Memory Used (KB) : Entry
         1      44.39      44.39      44.39          0  Heap Memory Used (KB) : Entry : int main(int, char **) C
         1       2068       2068       2068          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Comm_rank()
         1       2066       2066       2066          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Comm_size()
         1  5.139E+04  5.139E+04  5.139E+04          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Finalize()
         1      57.58      57.58      57.58          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Init()
        40  5.036E+04       2069  3.011E+04  1.228E+04  Heap Memory Used (KB) : Entry : int main(int, char **) C => void func(int, int) C
        40  5.139E+04       3098  3.114E+04  1.227E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Barrier()
        40  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Bcast()
       120  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Recv()
       120  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Send()
       365  5.139E+04       2065  3.116E+04   1.21E+04  Heap Memory Used (KB) : Exit
```

% export TAU_CALLPATH_DEPTH=2
% export TAU_TRACK_HEAP=1

Context event =atomic event + executing context

Library Wrapping (malloc)

- TAU provides wrapper libraries around *malloc*, *free*, et al. which replace the system version
- Each wrapper
  - Starts a timer
  - Records a context event indicating the size and source line of the allocation or deallocation
  - Call the underlying system version of the function
  - Stop the timer
- Loaded into unmodified application with LD_PRELOAD or through linker script at link time

Library Wrapping (OpenSHMEM)

- During configure, TAU
  - parses shmem.h
  - generates a wrapper library for each function defined in shmem.h
    - For each function, the wrapper
      - Starts a timer
      - For communications, records a context event indicating the size, source, destination and source code line of the communication.
- Parsing header works around differences between SHMEM implementations and versions.
- Loaded into unmodified application with LD_PRELOAD or through linker script at link time

```
extern void  __real_shmem_get128(void * a1, const void * a2, size_t a3, int a4) ;
extern void  __wrap_shmem_get128(void * a1, const void * a2, size_t a3, int a4)  {

  TAU_PROFILE_TIMER(t,"void shmem_get128(void *, const void *, size_t, int) C", "",
TAU_USER);
  TAU_PROFILE_START(t);
  TAU_TRACE_SENDMSG_REMOTE(TAU_SHMEM_TAGID_NEXT, Tau_get_node(), 16*a3, a4);
  __real_shmem_get128(a1, a2, a3, a4);
  TAU_TRACE_RECVMSG(TAU_SHMEM_TAGID, a4, 16*a3);
  TAU_PROFILE_STOP(t);
}
```

Malloc + OpenSHMEM wrapping

- TAU's existing memory tracking support gives us heap usage overall and per function, but does not tell us *where* in the runtime or *of what data type* the allocation belongs to.

TAU: ParaProf: Mean Context Events - /Users/nchaimov/Dropbox/ParaTools/OpenSHMEM/gups-profile-8

Name △	Total	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
▶ .TAU application						
Decrease in Heap Memory (KB)	0.007	0.188	0.036	0.036	0.036	0
Heap Allocate	59,894,454.571	74,735.352	4,194,304	1	801.421	36,159.707
Heap Free	28,335,073.63	45,649.867	4,194,304	1	620.704	29,656.05
Heap Memory Used (KB)	1,761,526,448.384	125,252.594	30,931.747	0.078	14,063.792	8,228.491
Heap Memory Used (KB) at Entry	8,143,537,888.971	263,345	30,931.747	134.328	30,923.457	159.489
Heap Memory Used (KB) at Exit	8,143,599,224.529	263,345	30,931.747	27,577.029	30,923.69	128.898
Increase in Heap Memory (KB)	61,335.594	5	30,668.419	0.152	12,267.119	13,794.88

▼ void shmem_init(void) C						
Heap Memory Used (KB) at Entry	135.609	1	135.609	135.609	135.609	0
Heap Allocate	56,441,752.868	74,058.289	4,194,304	1	762.126	34,251.795
Heap Free	28,185,517.874	45,125.633	4,194,304	1	624.601	29,821.428
Memory Error! Allocation of zero bytes	71.961	71.961	1	1	1	0
Heap Memory Used (KB) at Exit	27,577.03	1	27,577.03	27,577.03	27,577.03	0
Increase in Heap Memory (KB)	27,441.421	1	27,441.421	27,441.421	27,441.421	0

Allocation Classes

- New calls in TAU for tracking allocations
  - Track “flat” allocations (no relationships maintained)
    - Tau_track_class_allocation(name, size)
  - Track hierarchical allocations
    - Maintain allocation stack for context
    - Tau_start_class_allocation(name, size, include_in_parent)
    - Tau_stop_class_allocation(name, write_record)
  - Included in profile alongside timing data
  - Option to use context events: show *where* allocations occurred in the runtime
    - Two context stacks: timer stack and allocation stack
    - export TAU_MEM_CONTEXT=1
  - Default weak empty implementation allows enabling and disabling instrumentation at runtime.

```
Tau_start_class_allocation(“a”, 10, 0);   → 10 bytes allocated in object of type A
Tau_start_class_allocation(“b”, 25, 0);   → 25 bytes allocated in object of type B (child of A)
Tau_stop_class_allocation(“b”, 1);
Tau_stop_class_allocation(“a”, 1);
Tau_start_class_allocation(“b”, 10, 0);   → 10 bytes allocated in object of type B (not child)
Tau_stop_class_allocation(“b”, 1);
```

Stored in profile:

```
alloc a         10
alloc b         35
alloc b <= a    25
```

Instrumenting Open MPI

- OpenMPI OPAL object system allows centralized instrumentation of allocations of OPAL objects
  - Insert `Tau_start_class_allocation`, `Tau_stop_class_allocation` into `opal_obj_new` in `opal/class/opal_object.h`
- Tracking child objects requires manual instrumentation at the point of allocation
  - Dynamically-allocated members are allocated outside the constructor
  - Accomplished with dummy allocation regions
    - Reopen allocation region with `Tau_start_class_allocation` as normal.
    - Record child allocations
    - Close parent allocation region with write_record = 0

Tracking Flat Allocations

- Tracking allocations by type requires one line of code inserted into Open MPI runtime
  - *static inline* prevents use of library wrapper

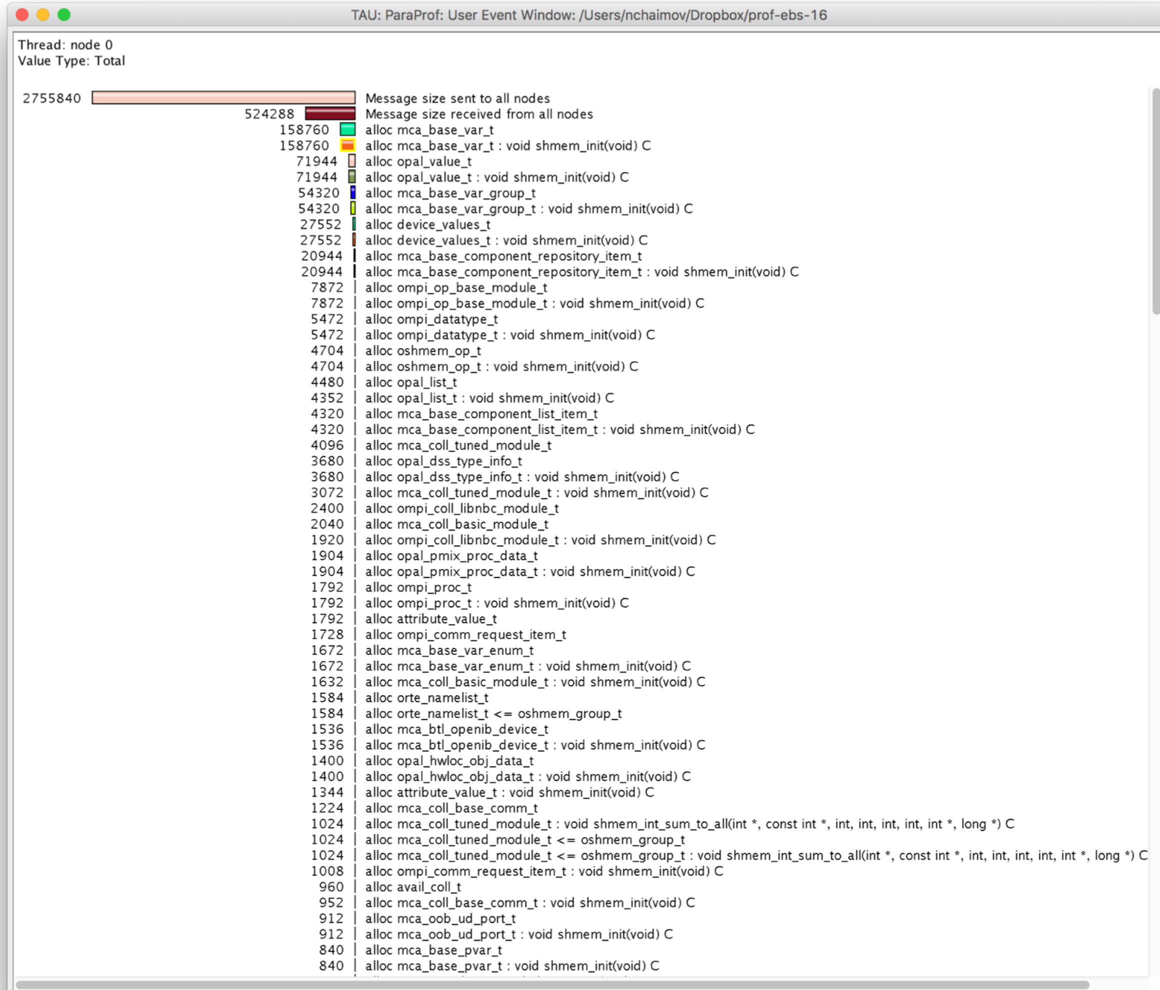
```
static inline opal_object_t *opal_obj_new(opal_class_t * cls)
{
    opal_object_t *object;
    assert(cls->cls_sizeof >= sizeof(opal_object_t));
    Tau_track_class_allocation(cls->cls_name, cls->cls_sizeof);
    [...]
}
```

Tracking Hierarchical Allocations

```
static inline opal_object_t *opal_obj_new(opal_class_t * cls
{
    opal_object_t *object;
    assert(cls->cls_sizeof >= sizeof(opal_object_t));

    Tau_start_class_allocation(cls->cls_name, cls->cls_sizeof, 0);

#if OPAL_WANT_MEMCHECKER
    object = (opal_object_t *) calloc(1, cls->cls_sizeof);
#else
    object = (opal_object_t *) malloc(cls->cls_sizeof);
#endif
    if (opal_class_init_epoch != cls->cls_initialized) {
        opal_class_initialize(cls);
    }
    if (NULL != object) {
        object->obj_class = cls;
        object->obj_reference_count = 1;
        opal_obj_run_constructors(object);
    }
    Tau_stop_class_allocation(cls->cls_name, 1);
    return object;
}
```

Allocations during constructors automatically attributed to enclosing allocation region

Memory Allocation Profiles



Memory Allocation Profiles

Sorted By: Number of Samples

Total	NumSamples	Max	Min	Mean	Std. Dev	Name
2755840	131280	8208	8	20.992	326.138	Message size sent to all nodes
524288	65536	8	8	8	0	Message size received from all nodes
158760	945	168	168	168	0	alloc mca_base_var_t
158760	945	168	168	168	0	alloc mca_base_var_t : void shmem_init(void) C
71944	529	136	136	136	0	alloc opal_value_t
71944	529	136	136	136	0	alloc opal_value_t : void shmem_init(void) C
27552	287	96	96	96	0	alloc device_values_t
27552	287	96	96	96	0	alloc device_values_t : void shmem_init(void) C
54320	194	280	280	280	0	alloc mca_base_var_group_t
54320	194	280	280	280	0	alloc mca_base_var_group_t : void shmem_init(void) C
20944	119	176	176	176	0	alloc mca_base_component_repository_item_t
20944	119	176	176	176	0	alloc mca_base_component_repository_item_t : void shmem_init(void) C
4704	98	48	48	48	0	alloc oshmem_op_t
4704	98	48	48	48	0	alloc oshmem_op_t : void shmem_init(void) C
4320	90	48	48	48	0	alloc mca_base_component_list_item_t
4320	90	48	48	48	0	alloc mca_base_component_list_item_t : void shmem_init(void) C
4480	70	64	64	64	0	alloc opal_list_t
4352	68	64	64	64	0	alloc opal_list_t : void shmem_init(void) C
3680	46	80	80	80	0	alloc opal_dss_type_info_t
3680	46	80	80	80	0	alloc opal_dss_type_info_t : void shmem_init(void) C
1400	35	40	40	40	0	alloc opal_hwloc_obj_data_t
1400	35	40	40	40	0	alloc opal_hwloc_obj_data_t : void shmem_init(void) C
1584	33	48	48	48	0	alloc orte_namelist_t
1584	33	48	48	48	0	alloc orte_namelist_t <= oshmem_group_t
1792	28	64	64	64	0	alloc attribute_value_t
1728	24	72	72	72	0	alloc ompi_comm_request_item_t
1344	21	64	64	64	0	alloc attribute_value_t : void shmem_init(void) C
1672	19	88	88	88	0	alloc mca_base_var_enum_t
1672	19	88	88	88	0	alloc mca_base_var_enum_t : void shmem_init(void) C
5472	19	288	288	288	0	alloc ompi_datatype_t
5472	19	288	288	288	0	alloc ompi_datatype_t : void shmem_init(void) C
1904	17	112	112	112	0	alloc opal_pmix_proc_data_t
1904	17	112	112	112	0	alloc opal_pmix_proc_data_t : void shmem_init(void) C
816	17	48	48	48	0	alloc orte_namelist_t : void shmem_init(void) C
816	17	48	48	48	0	alloc orte_namelist_t <= oshmem_group_t : void shmem_init(void) C
1792	16	112	112	112	0	alloc ompi_proc_t
1792	16	112	112	112	0	alloc ompi_proc_t : void shmem_init(void) C
128	16	8	8	8	0	alloc mxm_conn_h
128	16	8	8	8	0	alloc mxm_conn_h : void shmem_init(void) C
128	16	8	8	8	0	alloc mxm_conn_h <= ompi_proc_t
128	16	8	8	8	0	alloc mxm_conn_h <= ompi_proc_t : void shmem_init(void) C
768	16	48	48	48	0	alloc orte_namelist_t : void shmem_int_sum_to_all(int *, const int *, int, int, int, int, int *, long *) C
768	16	48	48	48	0	alloc orte_namelist_t <= oshmem_group_t : void shmem_int_sum_to_all(int *, const int *, int, int, int, int, int *, long *) C

Data Analysis in Jupyter

- We want to use memory tracking for scaling studies.
- ParaProf and PerfExplorer (TAU's visualizers) do not provide the right kind of charts for visualizing the scaling of context events.
- To increase flexibility, we added a parser which generates Pandas dataframes from TAU profile files.
  - Allows use of Jupyter notebooks and the wide array of Python visualization libraries on data collected from TAU.

jupyter gups_results (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Run Code

In [1]:
```
from tau_mem_summarize import TauTrialProfileData
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib as mp
import matplotlib.pyplot as plt
```

In [3]:
```
profiles = []
allocs = []
for num_nodes in [1,2,4,8,16,32,64]:
    dirname = 'gups-nodes-' + str(num_nodes)
    profile = TauTrialProfileData.parse(dirname)
    profiles.append(profile)
    allocs.append(profile.summarize_allocations()[['Total']].rename(columns
```

In [4]:
```
all_allocs = allocs[0].join(allocs[1:]).sort_values(by=64, ascending=False)
```

In [5]:
```
types_of_interest = ['ompi_proc_t', 'ompi_communicator_t', 'mca_coll_base_c
                     'ompi_request_t', 'ompi_predefined_request_t', 'ompi_w
                     'mca_pml_ob1_comm_proc_t', 'mca_pml_comm_t', 'mca_pml_
                     'mca_btl_base_module_t', 'oshmem_group_t', 'oshmem_gro
```

In [6]:
```
allocs_of_interest = all_allocs[all_allocs.index.str.contains('|'.join(type
#allocs_of_interest = all_allocs.transpose() # all
allocs_of_interest.index = allocs_of_interest.index.map(lambda x: x * 16)
allocs_of_interest.rename_axis('Ranks')
```

Out[6]:

Timer	alloc ompi_proc_t	alloc orte_namelist_t <= oshmem_group_t	alloc ompi_proc_t **	alloc ompi_proc_t ** <= oshmem_group_t	alloc ompi_proc_t ** <= oshmem_group_t <= ompi_group_t	al mxm_conr ompi_pro
Ranks						
16	28672.0	25344.0	8.320000e+03	4224.0	2048.0	204
32	114688.0	99840.0	3.302400e+04	16640.0	8192.0	819
64	458752.0	396288.0	1.315840e+05	66048.0	32768.0	3276
128	1835008.0	1579008.0	5.253120e+05	263168.0	131072.0	13107
256	7340032.0	6303744.0	2.099200e+06	1050624.0	524288.0	52428
512	29360128.0	25190400.0	8.392704e+06	4198400.0	2097152.0	209715
1024	117440512.0	100712448.0	3.356262e+07	16785408.0	8388608.0	838860

7 rows × 27 columns

Running an application

- Build your app with the instrumented runtime, then run with

```
<launcher> <launcher args>
tau_exec -T shmem,pdt -shmem -ebs -memory
<app> <app args>
```

- Example: GUPS on University of Oregon Talapas system (Slurm)
  - `srun  tau_exec -T shmem,pdt -shmem -ebs -memory ./gups`

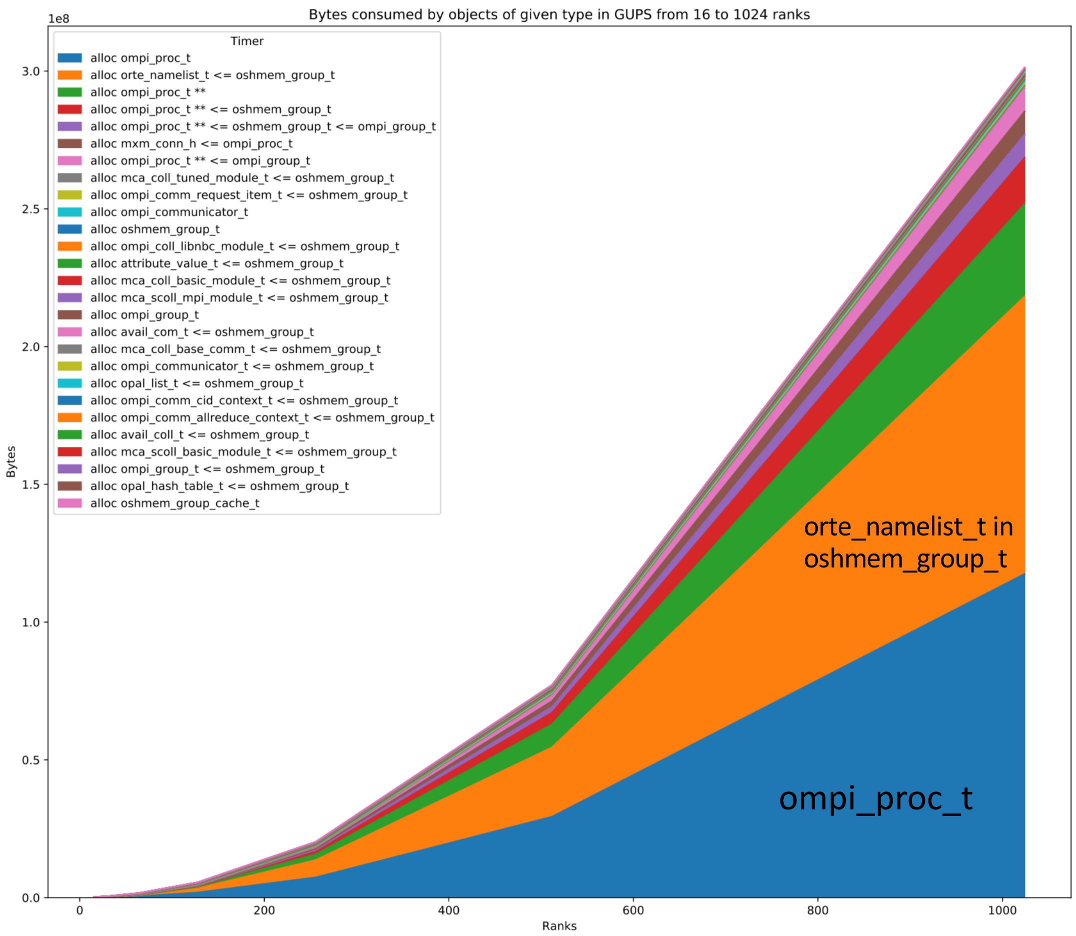
malloc wrapper

Configuration of TAU

SHMEM wrapper

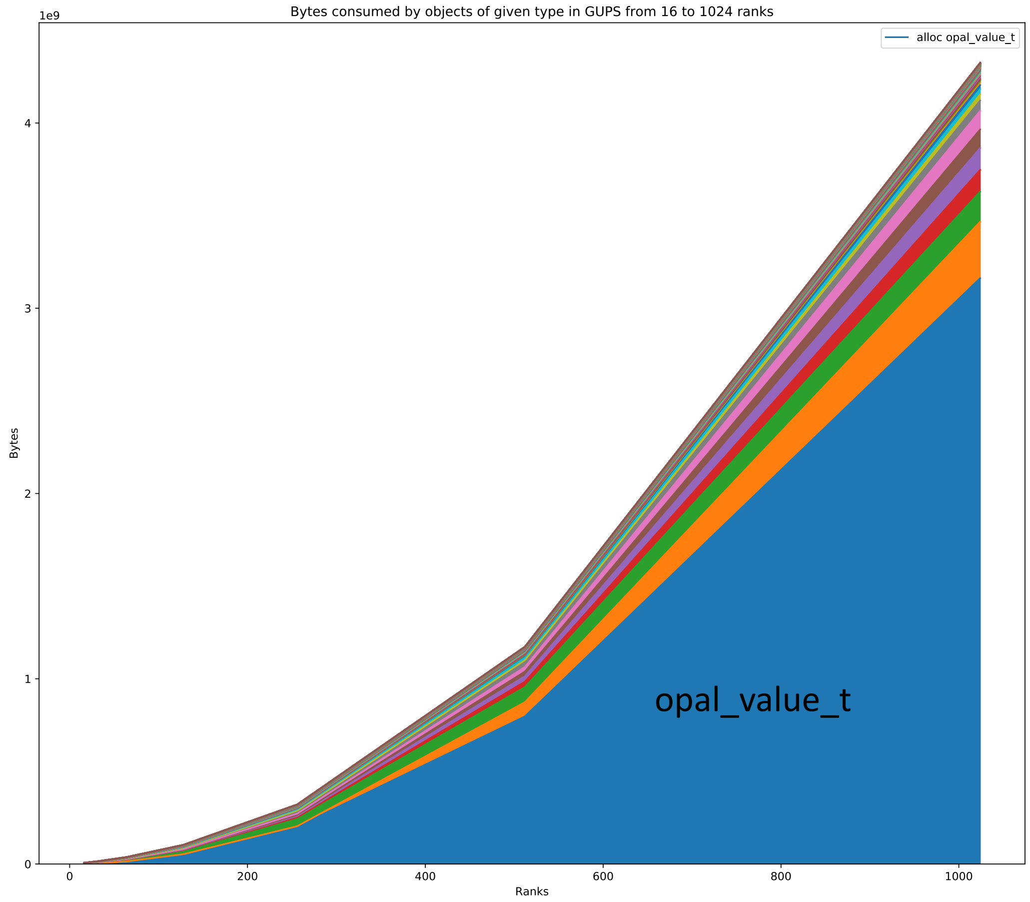
Sampling

Preliminary Results with GUPS



- Scaling GUPS from 16 to 1024 PEs on Oregon Talapas system
- Record allocations of objects of interest and their child allocations
- Object types with largest allocations in runtime at 1024 PEs among selected types
  - ompi_proc_t (117 MB)
  - orte_namelist_t (child of oshmem_group_t) (100 MB)
  - ompi_proc_t list (child of oshmem_group_t) (33.5 MB)

All Types



- Looking at *all* runtime types shows opal_value_t is by far the largest user of memory
- Usages are spread out as children of many other object types.

Time Series View

- This instrumentation gives us *total allocations*.
  - Sum of all allocations throughout application execution.
  - Does not distinguish between data types whose objects persist for the lifetime of the application and those that, for example
    - are only used during initialization; or,
    - are subject to repeated allocation and deallocation.
- To distinguish these cases, we need to keep traces or phase-based profiles, not context profiles.

OTF2 Trace Format in TAU

jlinford — ssh cori.nersc.gov — 150×14

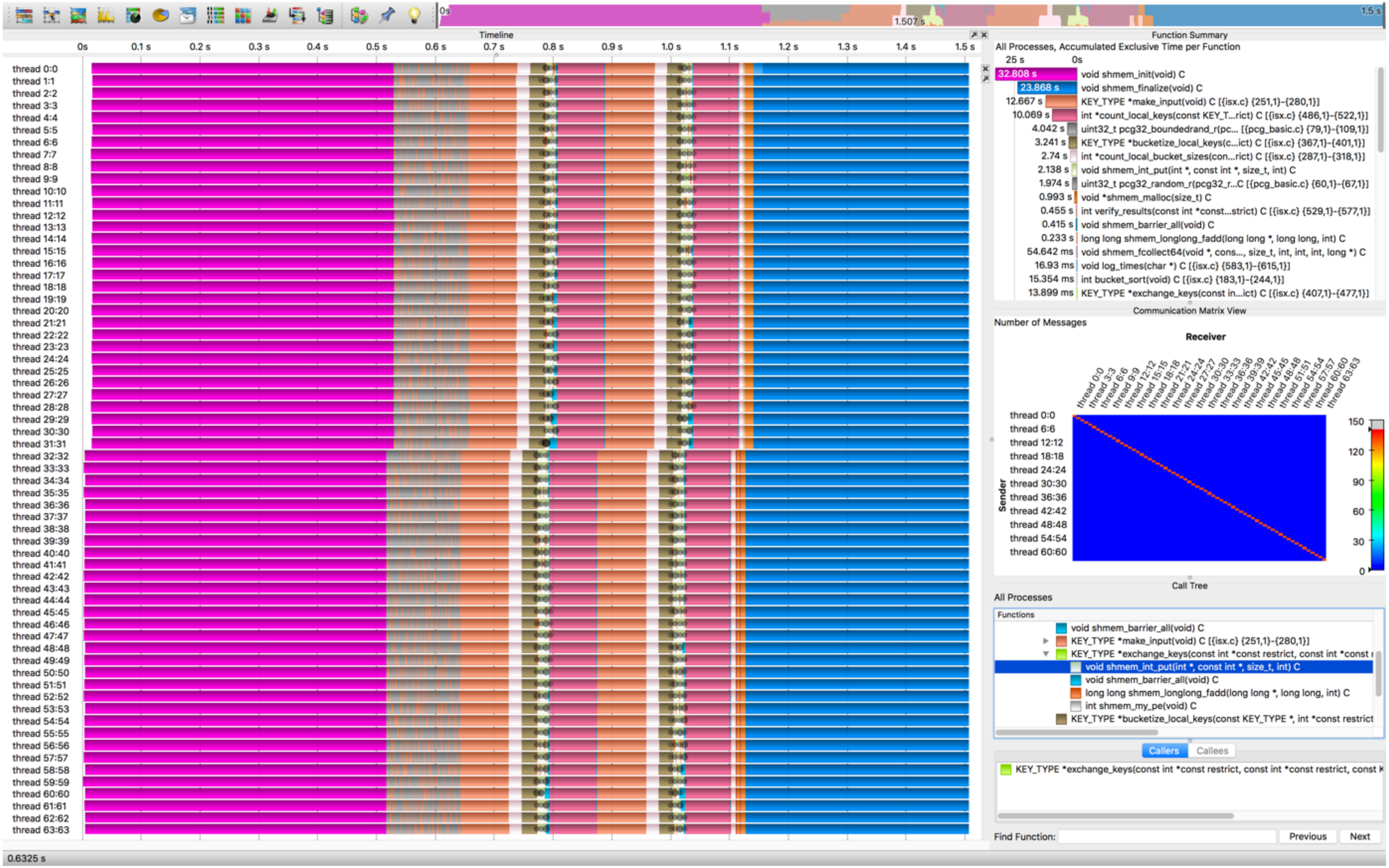
```
[jlinford@nid00011 ~/workspace/ISx/SHMEM :( $ tau meas list
== Measurement Configurations (/global/project/projectdirs/m88/jlinford/ISx/SHMEM/.tau/project.json) ==========================================
```

Name	Profile	Trace	Sample	Source Inst.	Compiler Inst.	OpenMP	CUDA	I/O	MPI	SHMEM
sample	tau	none	Yes	never	never	ignore	No	No	No	Yes
profile	tau	none	No	automatic	never	ignore	No	No	No	Yes
trace	none	otf2	No	automatic	never	ignore	No	No	No	Yes

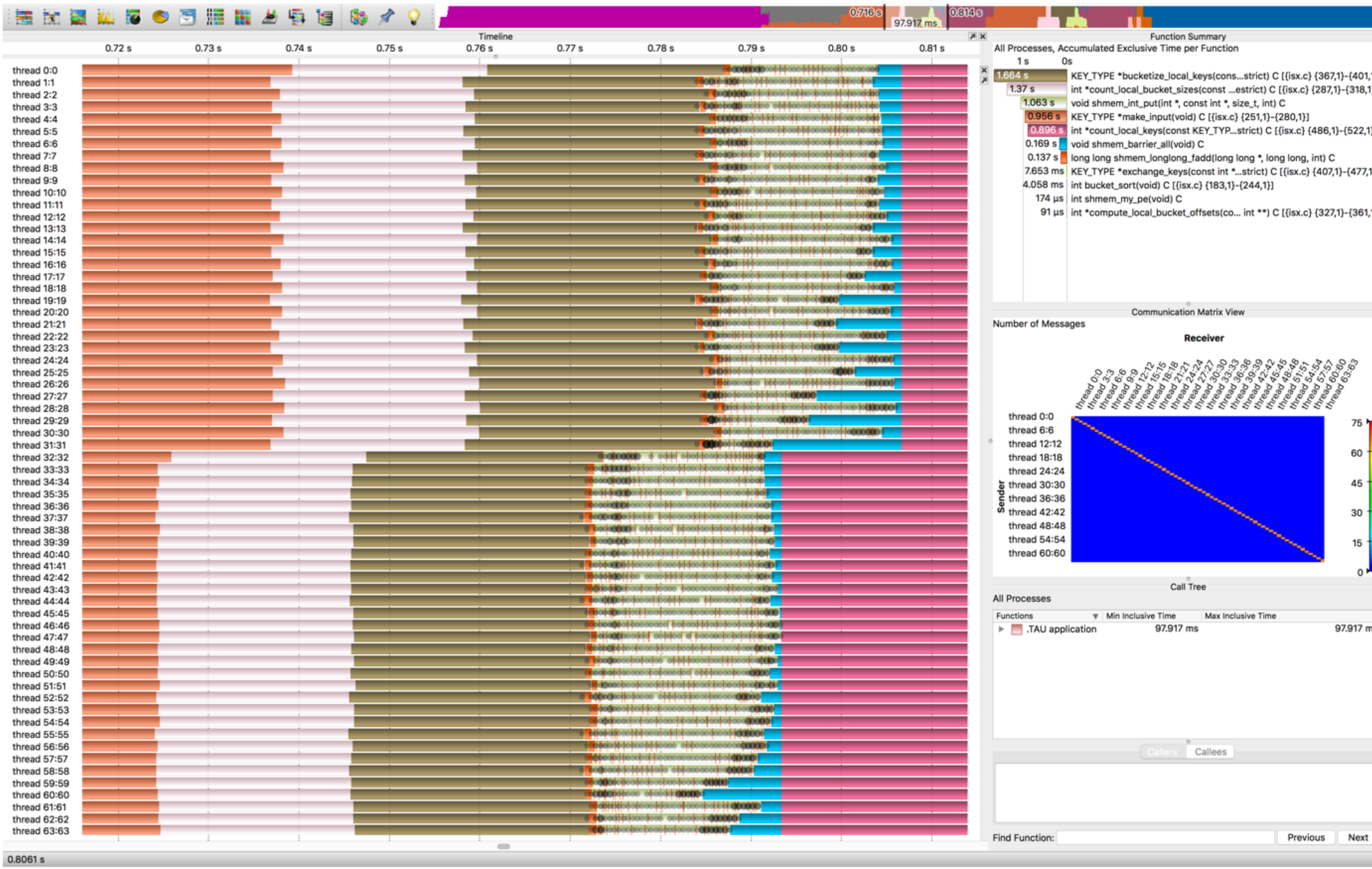
```
jlinford@nid00011 ~/workspace/ISx/SHMEM $
```

- OTF2 dramatically improves on SLOG2:
  - Smaller trace files
  - Richer trace data, e.g. RMA events
  - Better trace visualization (Vampir, Ravel)
- TAU can now generate OTF2 files natively:
  - No Score-P required!
  - Uses OpenSHMEM internally for event reduction
  - Writes context events to OTF2 trace

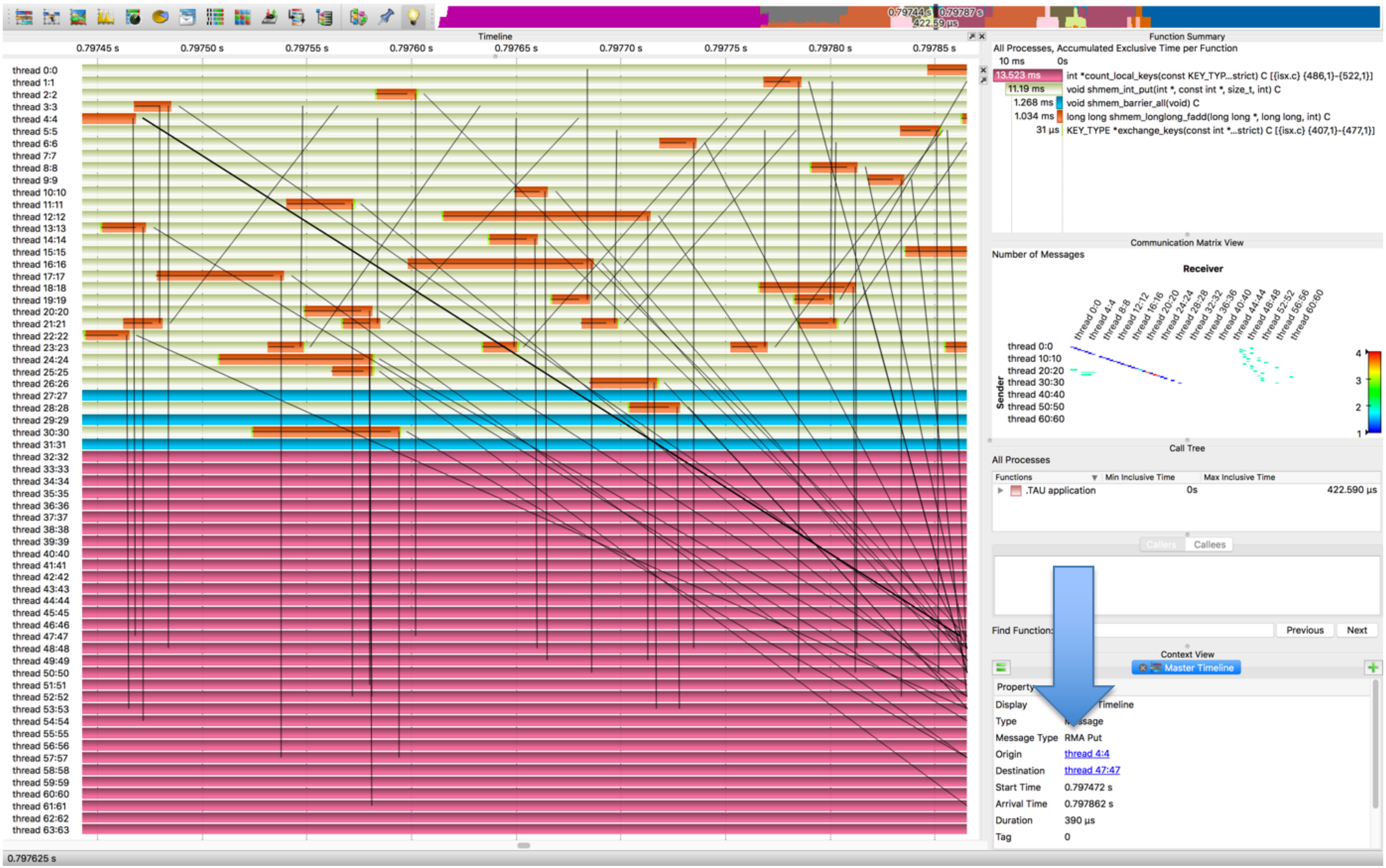
ISx in Vampir



Different Nodes, Different Timelines



Get/Put Recorded as RMA Events

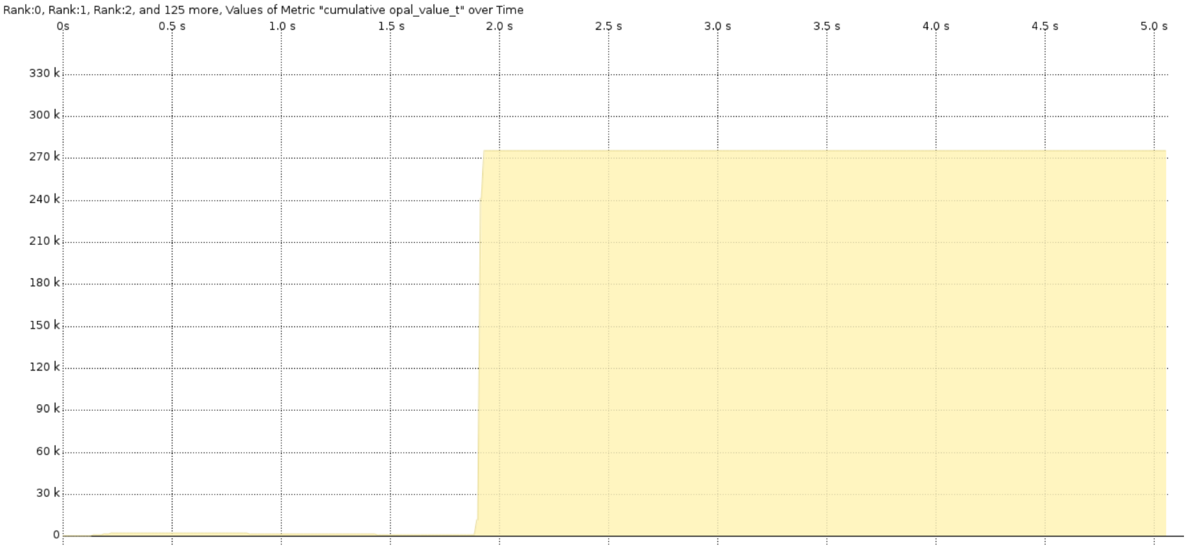


Cumulative Allocation by Class

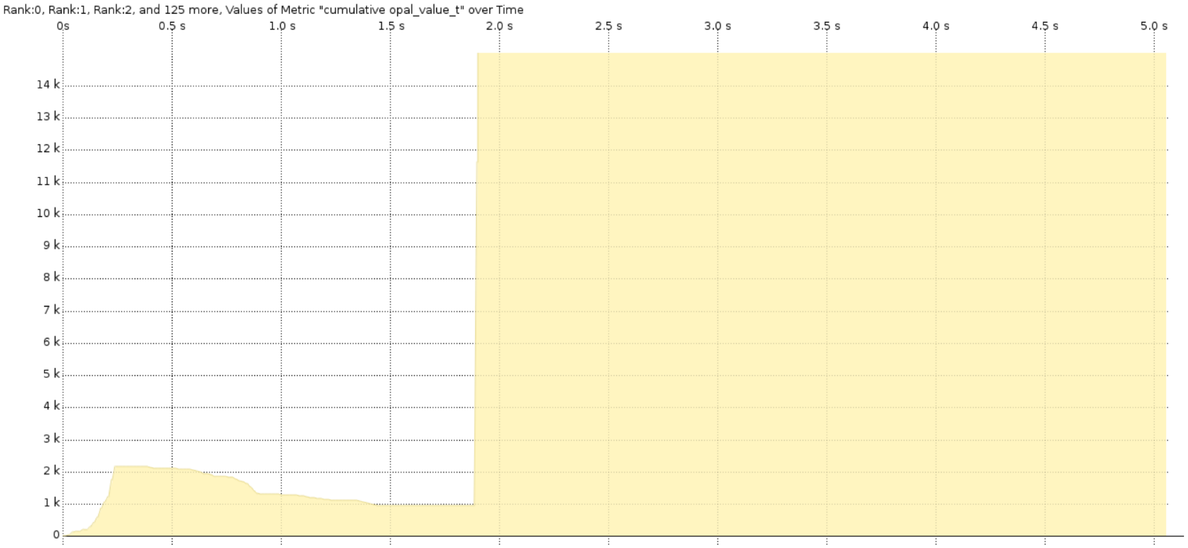
- Instrumentation of Open MPI destructors enables tracking frees by data type as well as mallocs, enabling a cumulative allocations context event.

```
static inline void opal_obj_run_destructors(opal_object_t * object)
{
    opal_destruct_t* cls_destruct;

    assert(NULL != object->obj_class);

    Tau_start_class_deallocation(object->obj_class->cls_name,object->obj_class->cls_sizeof, 0);
    cls_destruct = object->obj_class->cls_destruct_array;
    while( NULL != *cls_destruct ) {
        (*cls_destruct)(object);
        cls_destruct++;
    }
    Tau_stop_class_deallocation(object->obj_class->cls_name, 1;
}
```

Tracing opal_value_t in GUPS



Full view



Zoomed to show fluctuation during initialization

Future Work in Memory Allocation Tracking

- Identification of targets for optimization by sharing across PEs on a node
  - If there are object types which are written to only during initialization, or which have fields written to only during initialization, these might be shared between PEs on the same physical node
  - Identification requires tracking writes to allocations attributed to particular data types.
- Large scaling studies on Summit, etc.

Future Work in OpenSHMEM in TAU

- For SC18 release:
  - Support for OpenSHMEM 1.4 standard, including threading modes and contexts.
  - Support for hybrid CUDA/OpenSHMEM applications.

Conclusions

- With no runtime instrumentation, TAU can capture runtime allocations by *source line of origin*.
- With minimal runtime instrumentation (4 lines of code), TAU can capture runtime allocations by both source line of origin *and* data type of the allocation.
- Now available in TAU v2.27.2:
  - http://tau.uoregon.edu/tau.tgz
- Instrumented runtime at:
  - https://github.com/paratoolsinc/ompi