Introducing MPI and UCX Statistics to VI-HPS Tools

Shuki Zanyovka

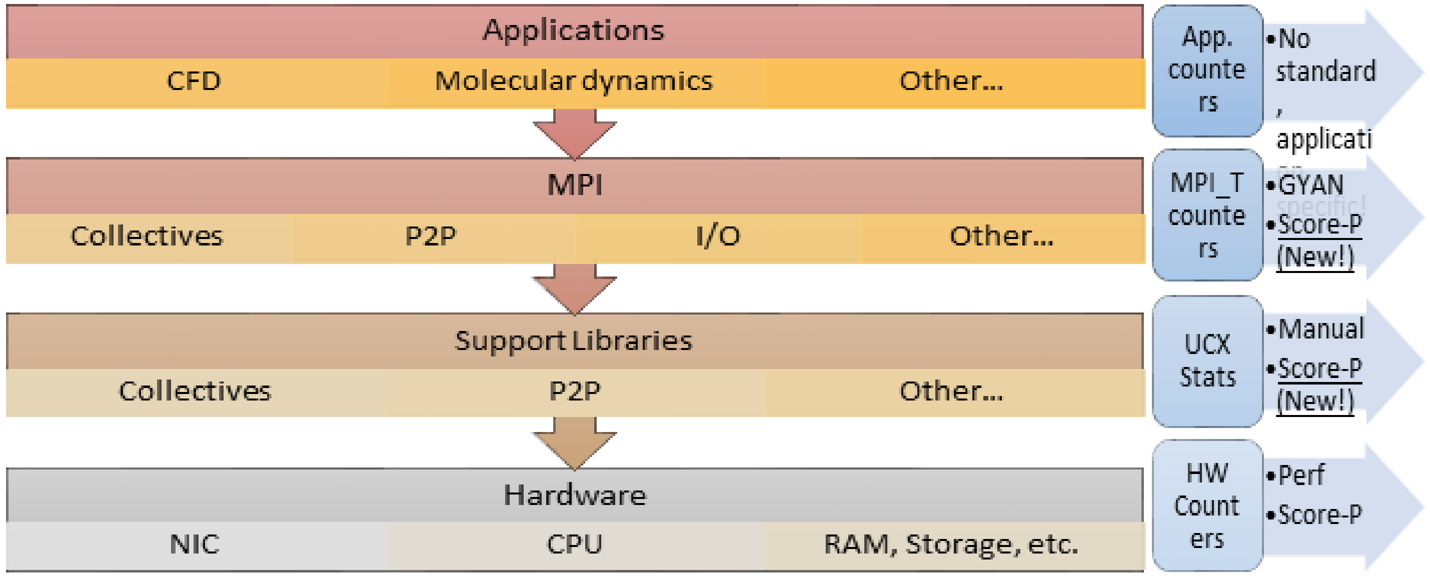
December 2020

HUAWEI | TEL AVIV RESEARCH CENTER

Outline

- Introducing Score-P and the VI-HPS profiling tools
- Case Study 1: LAMMPS (Molecular Dynamics Simulation)
- Case Study 2: WRF (Weather Research and Forecasting)
- Introducing the new Score-P UCX and MPI plugins
- Using Score-P in brief
- Challenges developing the plugins and temporary solutions

Applications / Libraries and Profiling Tools



Introduction to VI-HPS HPC applications profiling toolset

- VI-HPS stands for “Virtual Institute - High Productivity Supercomputing”
- The VI-HPS is a set of multiple profiling and tracing tools for HPC applications
  - Developed by multiple universities across Europe and the US
- Most of the tools and documentation (+ training slides) can be found on https://www.vi-hps.org/

Score-P: Profiling vs. Tracing

- Score-P is the main tool for data acquisition (provided as part of VI-HPS toolset)

- It supports the following modes of operation,
  - **Profiling** of MPI applications to provide a final aggregated (reduce-sum from all MPI processes) value of **each metric**
  - **Tracing** (data acquisition process) of MPI applications via instrumentation (adding a prolog / epilog to each function + sampling)

- Different metrics can be collected in order to properly understand,
  - How an application scales on a cluster setup of N nodes?
  - Get hints regarding what can be done in order to accelerate the application
  - Using these hints, one can try to accelerate the application by modifying either a library (for example, MPI or UCX) or update the application itself

Challenges of MPI Applications Profiling / Tracing #1

- The data acquisition process of an application shouldn’t change its timing of execution (shouldn’t slow it down)
- More processes (and nodes) ➔ **More data collection**
- More metrics (counters) ➔ **More data collection**
- More data ➔ **Writing to storage might slow down the application**
- Which metrics are interesting to measure?
- Post-run analysis of the trace data

HUAWEI | TEL AVIV RESEARCH CENTER

Challenges of MPI Applications Profiling / Tracing #2

- Let’s say we have an application with the following profiling report,
    - 20% network utilization
    - 30% storage utilization
    - 30% CPU time spent on function A
    - 40% CPU time spent on function B
    - Rest of the CPU time spent on multiple functions

- **What can we say as SW engineers (What can be improved)?**

Challenges of MPI Applications Profiling / Tracing #3

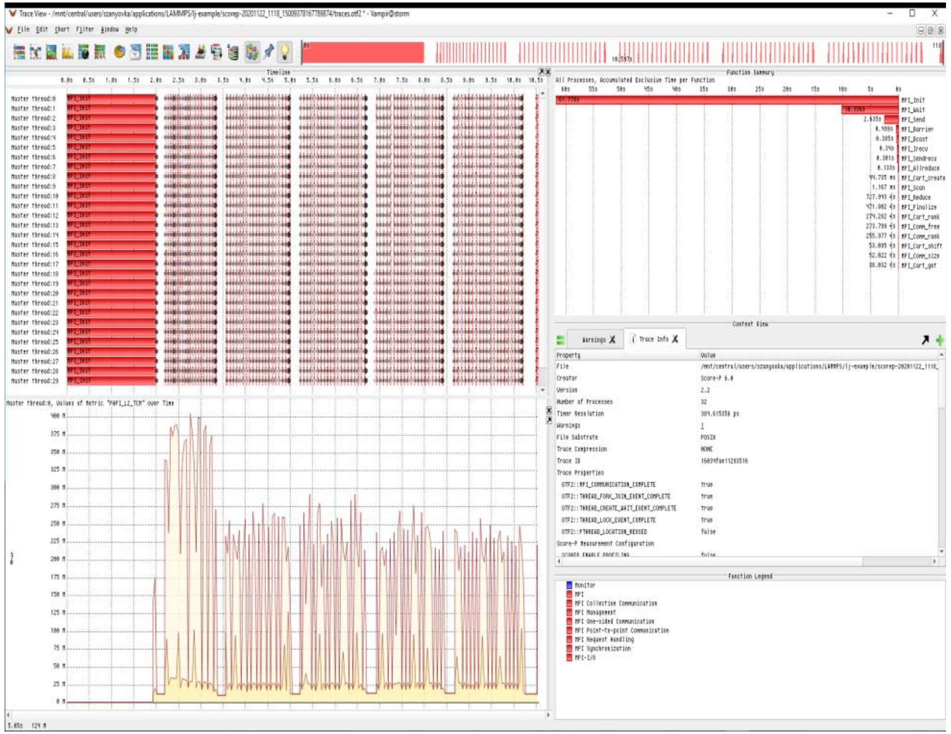
- **Basically, not so much !!!**

- So what kind of measurements might indicate bottlenecks?
  - Spike in network utilization
  - Spike in storage utilization
  - High L1/L2 or L3 cache miss rate ➔**Larger than 20% might be an indicator for a bottleneck**

- To investigate further, an iterative process can be applied,
  - Provide more profiling data to application experts (Scientists who build the applications)
  - Investigating if the data makes sense ➔ If not, we might have room for improvement

Case Study 1: LAMMPS (Molecular Dynamics Simulation)

- This is an example trace visualization in Vampir
- On the right side, we can see the functions that took the most time
- On the left side, we see the trace visualization per process and a certain metric (PAPI L2 cache miss counter)


Trace View
File Edit Chart Filter Window Help
Timeline
Master thread:0, Values of metric "PAPI_L2_TCM" over Time
Function Summary
All Processes, Accumulated Exclusive Time per Function
Context View
Warnings
Trace Info
Function Legend

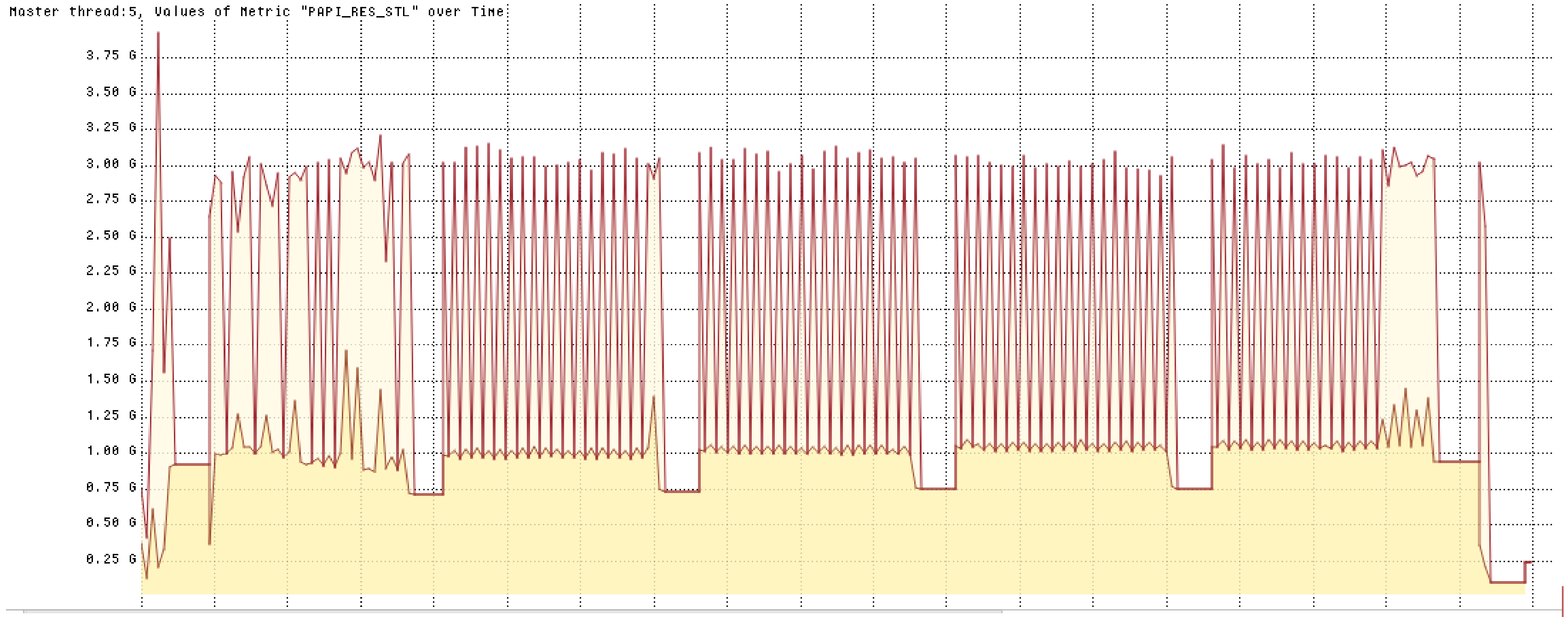
Function	Time
MPI_Init	61.778s
MPI_Wait	[illegible]
MPI_Send	2.635s
MPI_Barrier	0.499s
MPI_Bcast	0.385s
MPI_Irecv	0.34s
MPI_Sendrecv	0.301s
MPI_Allreduce	0.133s
MPI_Cart_create	94.735 ms
MPI_Scan	1.167 ms
MPI_Reduce	727.943 µs
MPI_Finalize	471.082 µs
MPI_Cart_rank	274.282 µs
MPI_Comm_free	273.799 µs
MPI_Comm_rank	255.977 µs
MPI_Cart_shift	53.095 µs
MPI_Comm_size	52.022 µs
MPI_Cart_get	38.032 µs

Property	Value
File	/mnt/central/users/szanyovka/applications/LAMMPS/lj-example/scorep-20201122_1118_
Creator	Score-P 6.0
Version	2.2
Number of Processes	32
Timer Resolution	384.615356 ps
Warnings	1
File Substrate	POSIX
Trace Compression	NONE
Trace ID	16034fae11293516
Trace Properties	
OTF2::MPI_COMMUNICATION_COMPLETE	true
OTF2::THREAD_FORK_JOIN_EVENT_COMPLETE	true
OTF2::THREAD_CREATE_WAIT_EVENT_COMPLETE	true
OTF2::THREAD_LOCK_EVENT_COMPLETE	true
OTF2::PTHREAD_LOCATION_REUSED	false
Score-P Measurement Configuration	
SCOREP_ENABLE_PROFILING	false

🟦	Monitor
🟥	MPI
🟥	MPI Collective Communication
🟥	MPI Management
🟥	MPI One-sided Communication
🟥	MPI Point-to-point Communication
🟥	MPI Request Handling
🟥	MPI Synchronization
🟥	MPI-I/O

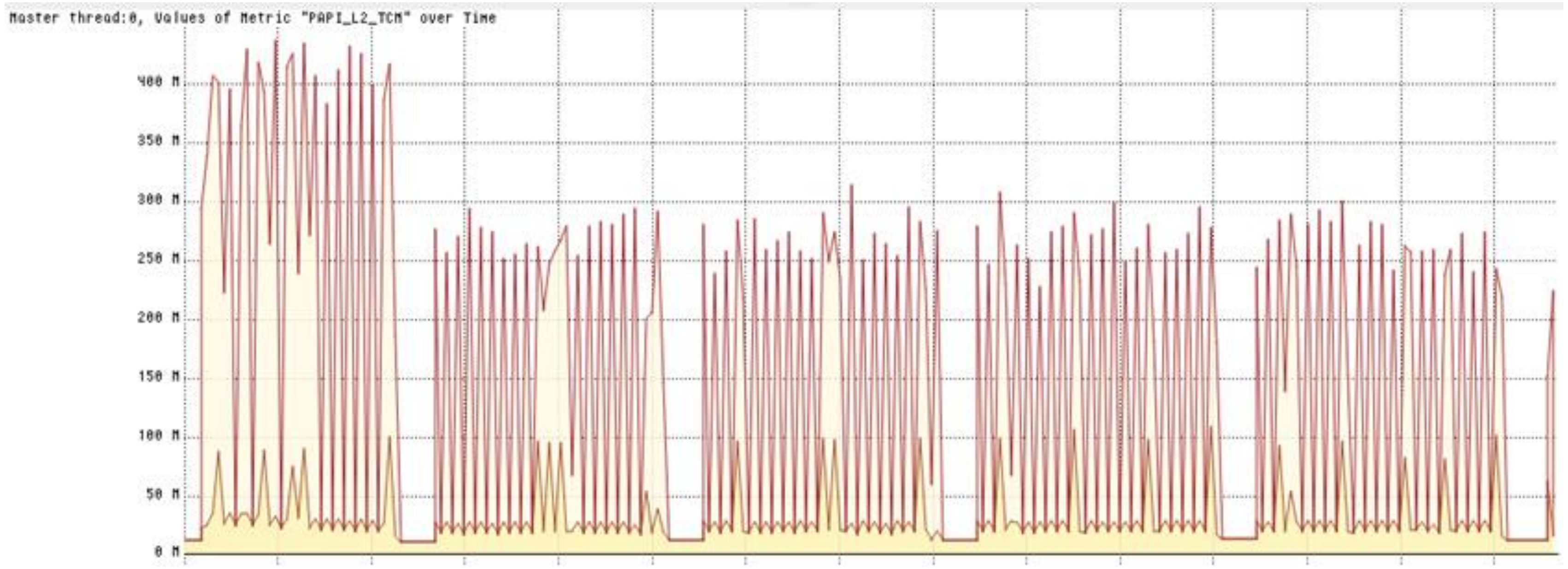
Case Study 1: LAMMPS Cycles Stalled [PAPI] #1

- PAPI_RES_STL counts the number of cycles the processor is **stalled on a resource**



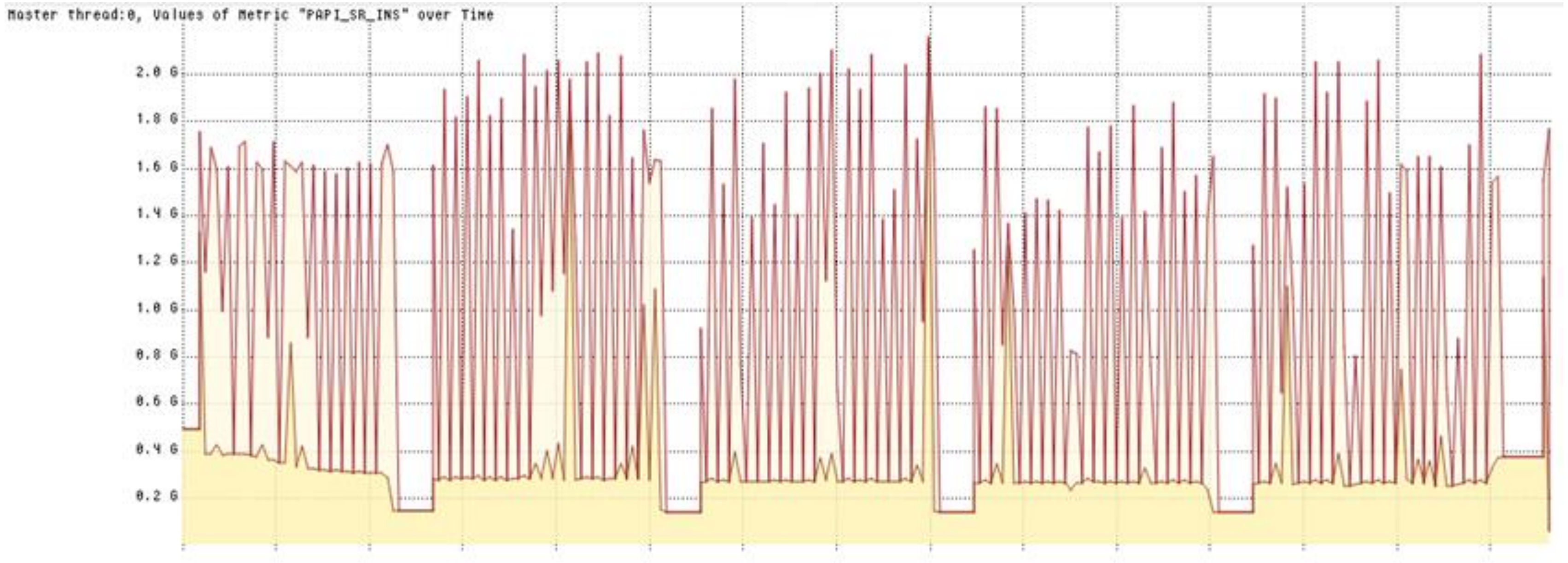
Case Study 1: LAMMPS L2-cache miss rate [PAPI] #2

- L2-cache miss counter (**peak = 250M at steady state**)
- We see a peak of 400M during the first cycle as expected



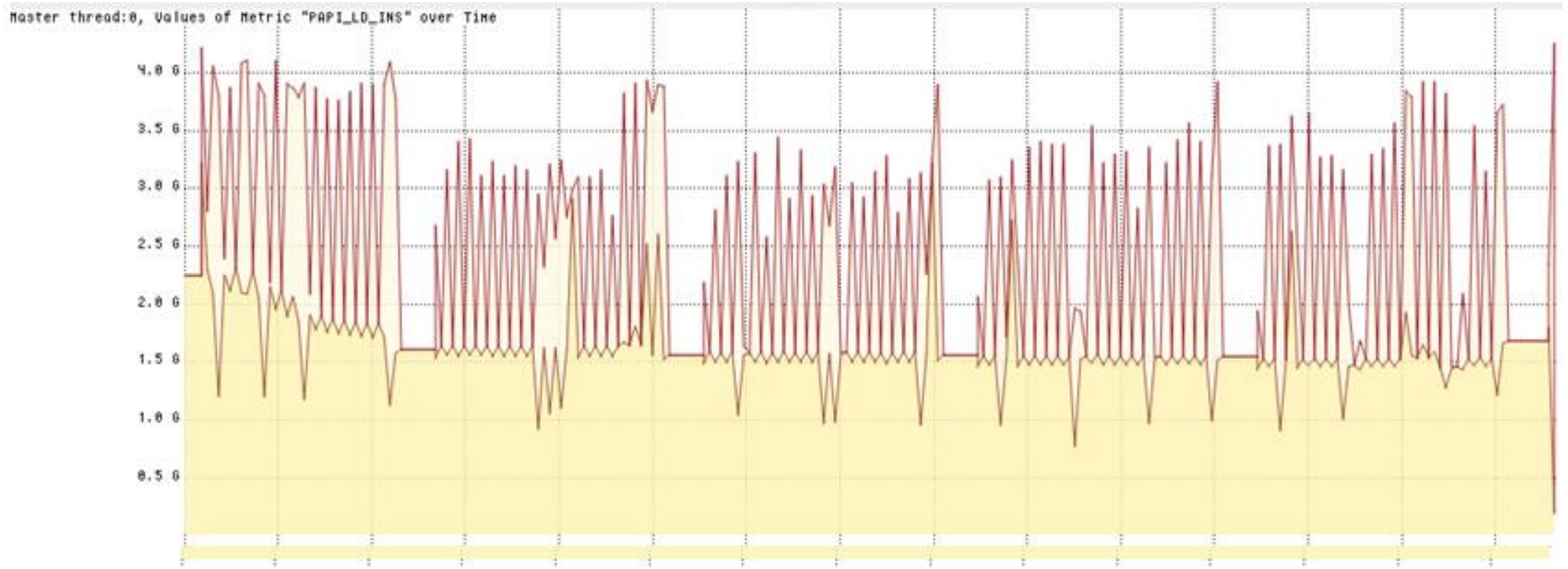
Case Study 1: LAMMPS L2-cache miss rate [PAPI] #3

- **Store** instructions over time (**peak = 1.8G**)



Case Study 1: LAMMPS L2-cache miss rate [PAPI] #4

- **Load** instructions over time (**peak = 3.5G**)

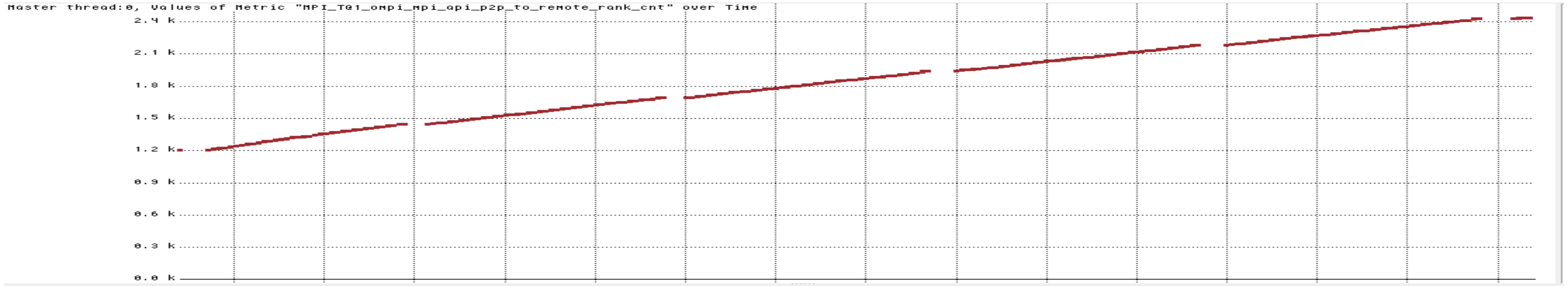


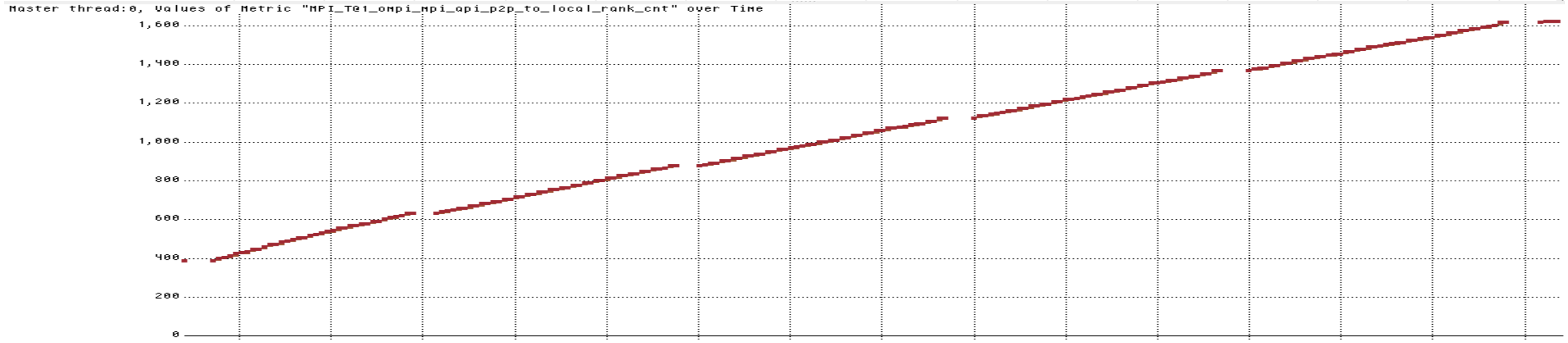
Case Study 1: LAMMPS

- We used the “lj” example with 4 million atoms configuration in this measurement
- It would seem like the cache miss rate of the LAMMPS application in the configuration we ran was not bad: **(~250M / (3.5G + 1.8G)) ~= 4.7%**
- Let’s take a look at some new metrics introduced via the MPI_T interface in H-MPI,
  - Allreduce / Reduce: **Count the number of times the same buffer was used in recv/send buffer in 2 consecutive calls** (if we find such cases, we can reduce the latency of the WQE (Work Queue Element) and CQE (Completion Queue Element) creation by pre-allocating it so UCG can take this into account)
  - Number of times P2P send was used with **remote** rank (i.e. a different node)
  - Number of times P2P send was used with **local** rank (i.e. the same node ➔ **Via shmem**)

Case Study 1: LAMMPS – MPI P2P to local vs. remote

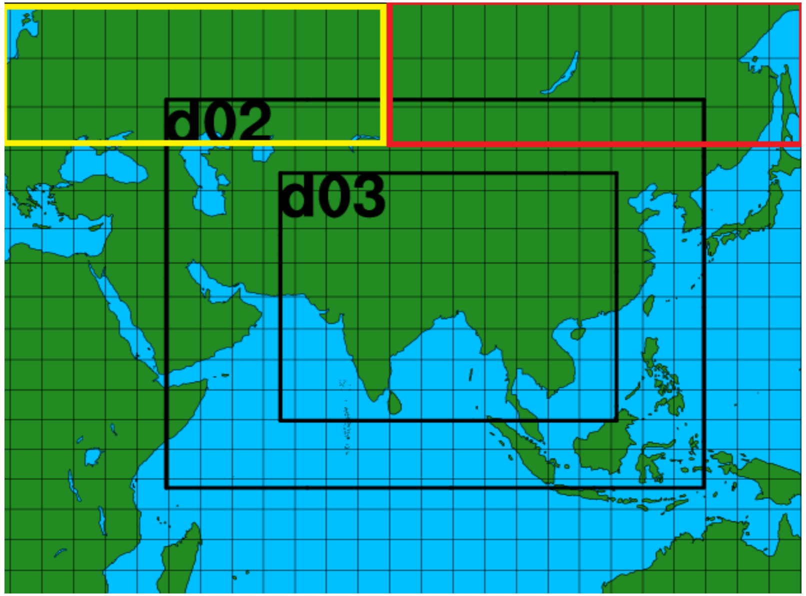
- **We see 1.2K P2P sends to remote ranks vs. 1.2K to local ranks**

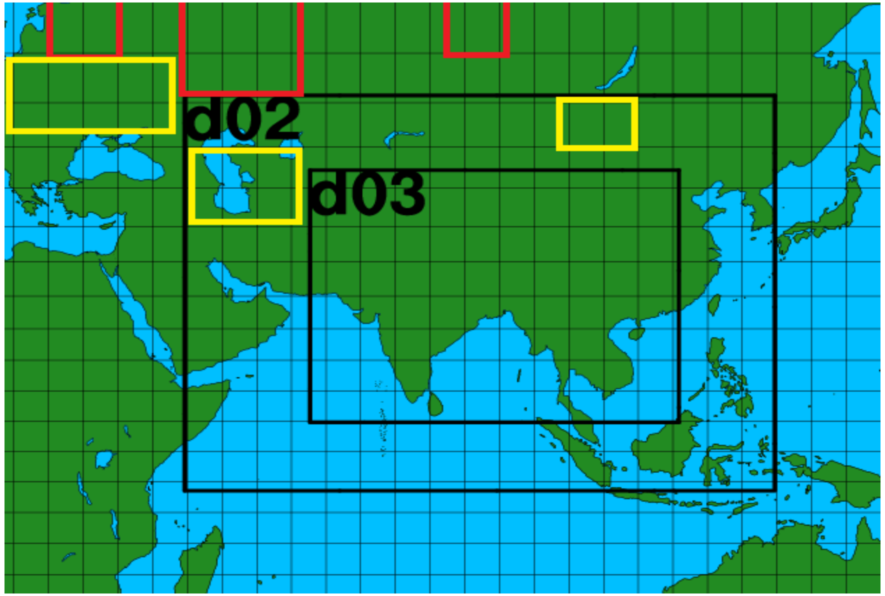




P2P to Remote vs. Local rank

- P2P to **remote** larger than P2P to **local** ➔ **Might be an indication for:**
- **Balanced** vs. **Unbalanced** organization of the neighbors workload

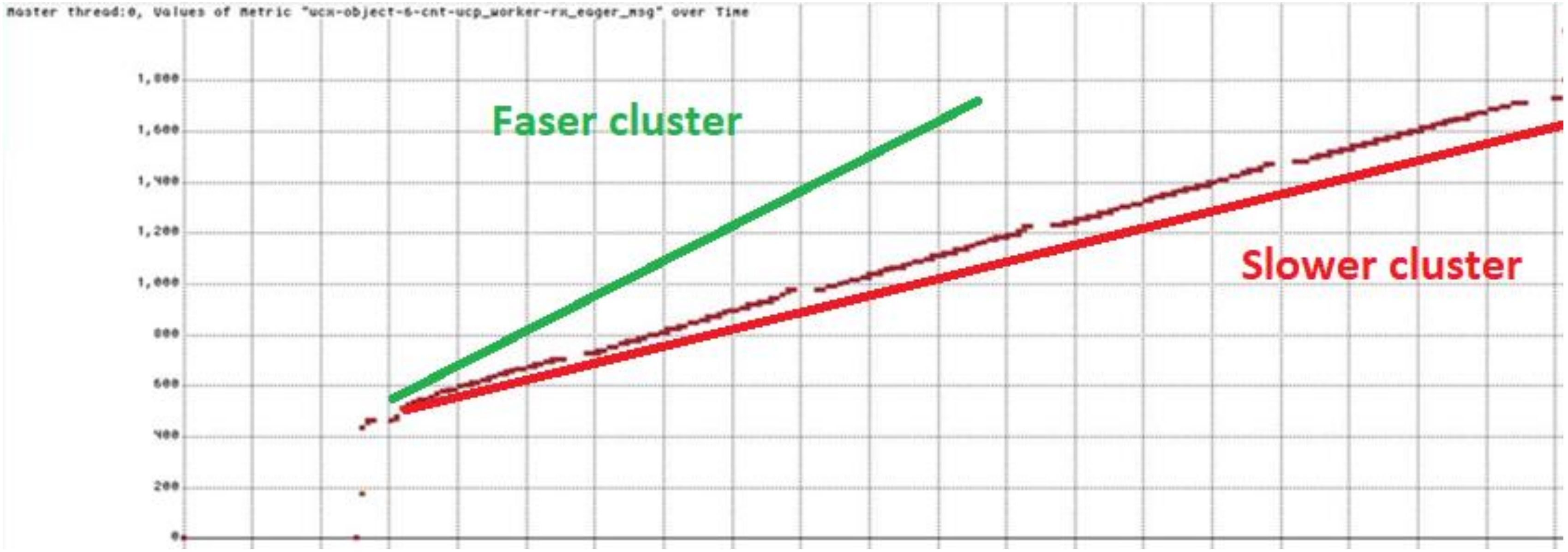




TEL AVIV RESEARCH CENTER

Case Study 1: LAMMPS - UCX counters [Rx eager msg]

- We can see which communication interface was used
- A stable slope will indicate a stable flow of data between processes



Case Study 1: LAMMPS – Analysis #1

- We learn from the new metrics introduced the following,
- In our measurements, we see more P2P sends to **remote** ranks rather than **local** ranks,
  - This might give us a hint that we can re-organize the MPI ranks so that neighbors reside on the same node as much as possible
  - In general, we would like to **minimize** P2P sends to **remote** addresses
- We also learn from the large number of **consecutive** calls to MPI_Allreduce() with the same send and recv buffers that it would make sense to try optimizing the WQE / CQE function calls of the application **to indicate to UCG that the same buffer is used**

Case Study 1: LAMMPS – Analysis #2

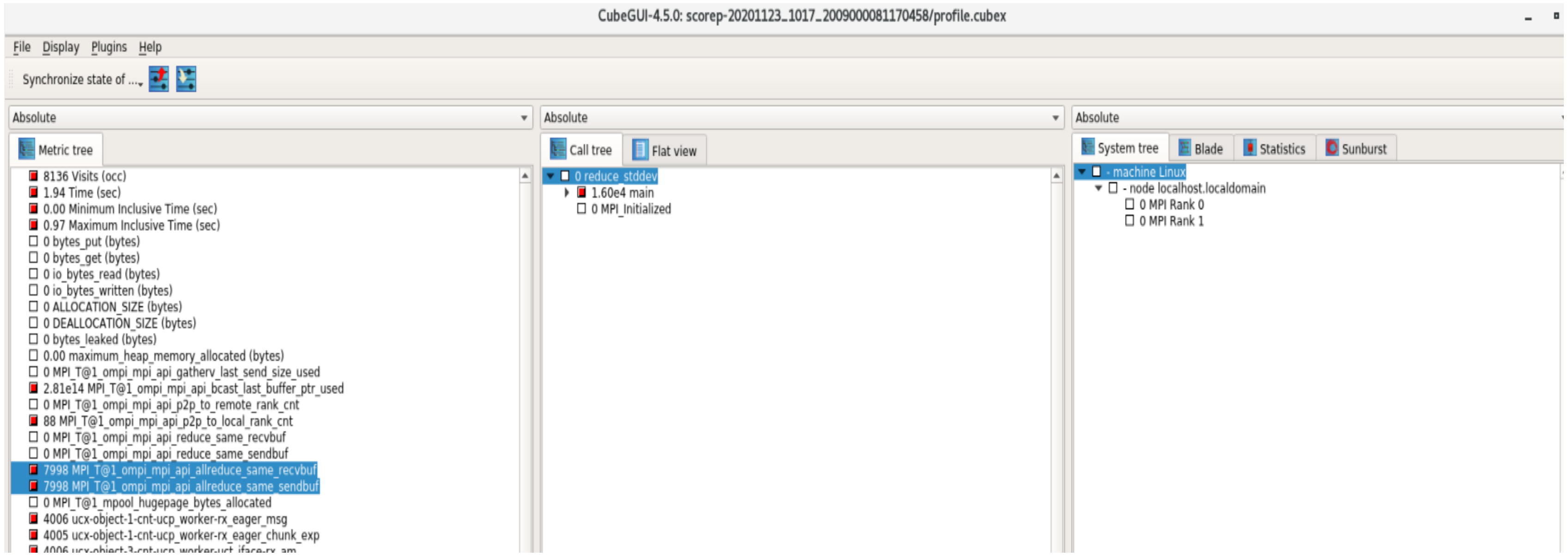
- Besides this, all UCX statistics counters can be sampled into the trace using the Huawei Score-P UCX plugin,
  - UCX counters are sampled at a low rate and a filter can be applied to them using the **UCX_STATS_FILTER environment variable**:
    export UCX_STATS_FILTER="rx_am*,bytes_short,bytes_bcopy,bytes_zcopy,rx*,tx*"

Case Study 1: LAMMPS – Analysis #3

- Sampling the UCX statistics using the plugin can assist in understanding,
  - The communication pattern of the application
  - Might show a degradation in the rate of packets sent which could indicate a **bottleneck in the system**
  - For example, in the rx_eager counter show in previous slides we see a rather stable rate
  - The amount of data sent over the network vs. to the local node

Allreduce with the same recv/send buf

- An example of the profiling data showing that MPI_Allreduce() was called multiple times with the same send and recv buffers



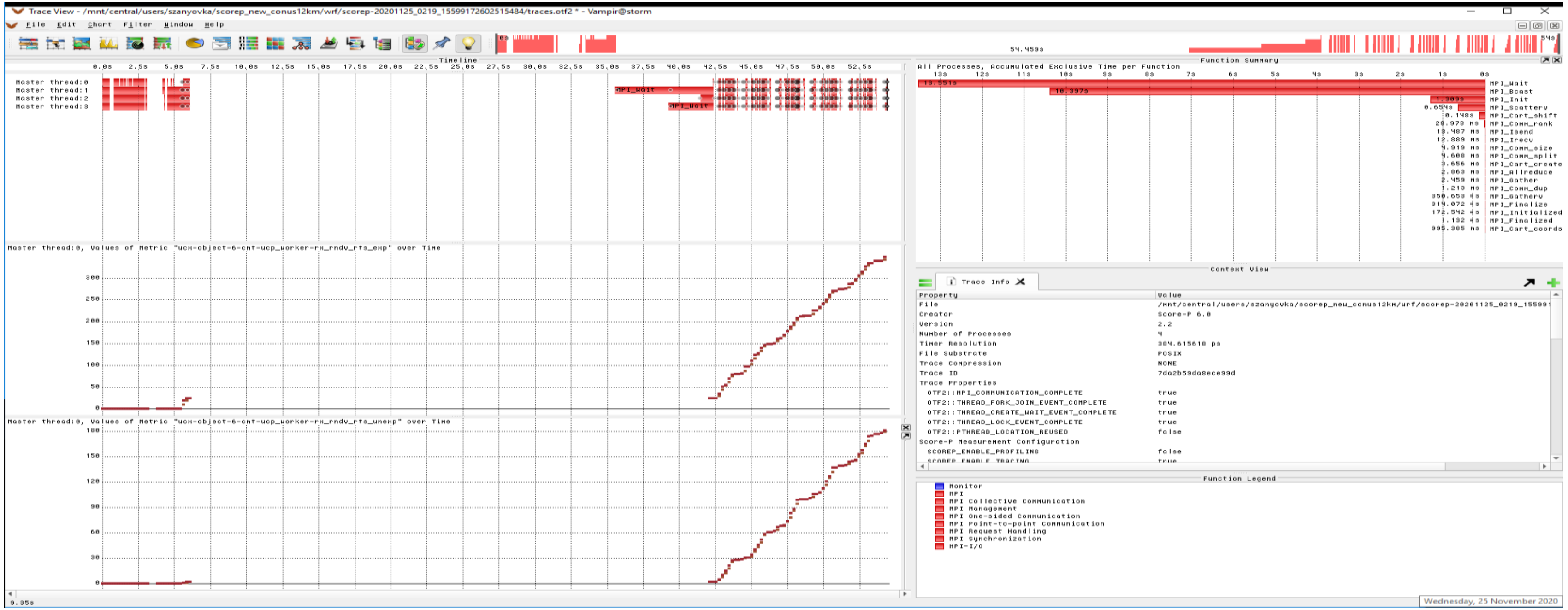
```
8136 Visits (occ)
1.94 Time (sec)
0.00 Minimum Inclusive Time (sec)
0.97 Maximum Inclusive Time (sec)
0 bytes_put (bytes)
0 bytes_get (bytes)
0 io_bytes_read (bytes)
0 io_bytes_written (bytes)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
0 bytes_leaked (bytes)
0.00 maximum_heap_memory_allocated (bytes)
0 MPI_T@1_ompi_mpi_api_gatherv_last_send_size_used
2.81e14 MPI_T@1_ompi_mpi_api_bcast_last_buffer_ptr_used
0 MPI_T@1_ompi_mpi_api_p2p_to_remote_rank_cnt
88 MPI_T@1_ompi_mpi_api_p2p_to_local_rank_cnt
0 MPI_T@1_ompi_mpi_api_reduce_same_recvbuf
0 MPI_T@1_ompi_mpi_api_reduce_same_sendbuf
7998 MPI_T@1_ompi_mpi_api_allreduce_same_recvbuf
7998 MPI_T@1_ompi_mpi_api_allreduce_same_sendbuf
0 MPI_T@1_mpool_hugepage_bytes_allocated
4006 ucx-object-1-cnt-ucp_worker-rx_eager_msg
4005 ucx-object-1-cnt-ucp_worker-rx_eager_chunk_exp
```

```
0 reduce_stddev
  1.60e4 main
  0 MPI_Initialized
```

```
- machine Linux
  - node localhost.localdomain
    0 MPI Rank 0
    0 MPI Rank 1
```

Case Study 2: WRF (Weather Research and Forecasting)

- Rx RNDV RTS “expected” vs. “unexpected”


Trace View - /mnt/central/users/szanyovka/scorep_new_conus12km/wrf/scorep-20201125_0219_15599172602515484/traces.otf2 * - Vampir@storm
File Edit Chart Filter Window Help
Timeline
Master thread:0
Master thread:1
Master thread:2
Master thread:3
MPI_Wait
Master thread:0, Values of Metric "ucx-object-6-cnt-ucp_worker-rx_rndv_rts_exp" over Time
Master thread:0, Values of Metric "ucx-object-6-cnt-ucp_worker-rx_rndv_rts_unexp" over Time
54.459s
Function Summary
Content View
Trace Info
Function Legend
Wednesday, 25 November 2020

All Processes, Accumulated Exclusive Time per Function

13.551s	MPI_Wait
10.397s	MPI_Bcast
1.309s	MPI_Init
0.654s	MPI_Scatterv
0.148s	MPI_Cart_shift
28.973 ms	MPI_Comm_rank
13.487 ms	MPI_Isend
12.889 ms	MPI_Irecv
4.919 ms	MPI_Comm_size
4.608 ms	MPI_Comm_split
3.656 ms	MPI_Cart_create
2.863 ms	MPI_Allreduce
2.459 ms	MPI_Gather
1.213 ms	MPI_Comm_dup
350.653 µs	MPI_Gatherv
314.072 µs	MPI_Finalize
172.542 µs	MPI_Initialized
1.132 µs	MPI_Finalized
995.385 ns	MPI_Cart_coords

Property	Value
File	/mnt/central/users/szanyovka/scorep_new_conus12km/wrf/scorep-20201125_0219_155991
Creator	Score-P 6.0
Version	2.2
Number of Processes	4
Timer Resolution	384.615618 ps
File Substrate	POSIX
Trace Compression	NONE
Trace ID	7da2b59da8ece99d
Trace Properties	
OTF2::MPI_COMMUNICATION_COMPLETE	true
OTF2::THREAD_FORK_JOIN_EVENT_COMPLETE	true
OTF2::THREAD_CREATE_WAIT_EVENT_COMPLETE	true
OTF2::THREAD_LOCK_EVENT_COMPLETE	true
OTF2::PTHREAD_LOCATION_REUSED	false
Score-P Measurement Configuration	
SCOREP_ENABLE_PROFILING	false
SCOREP_ENABLE_TRACING	true

- 🟦 Monitor
- 🟥 MPI
- 🟥 MPI Collective Communication
- 🟥 MPI Management
- 🟥 MPI One-sided Communication
- 🟥 MPI Point-to-point Communication
- 🟥 MPI Request Handling
- 🟥 MPI Synchronization
- 🟥 MPI-I/O

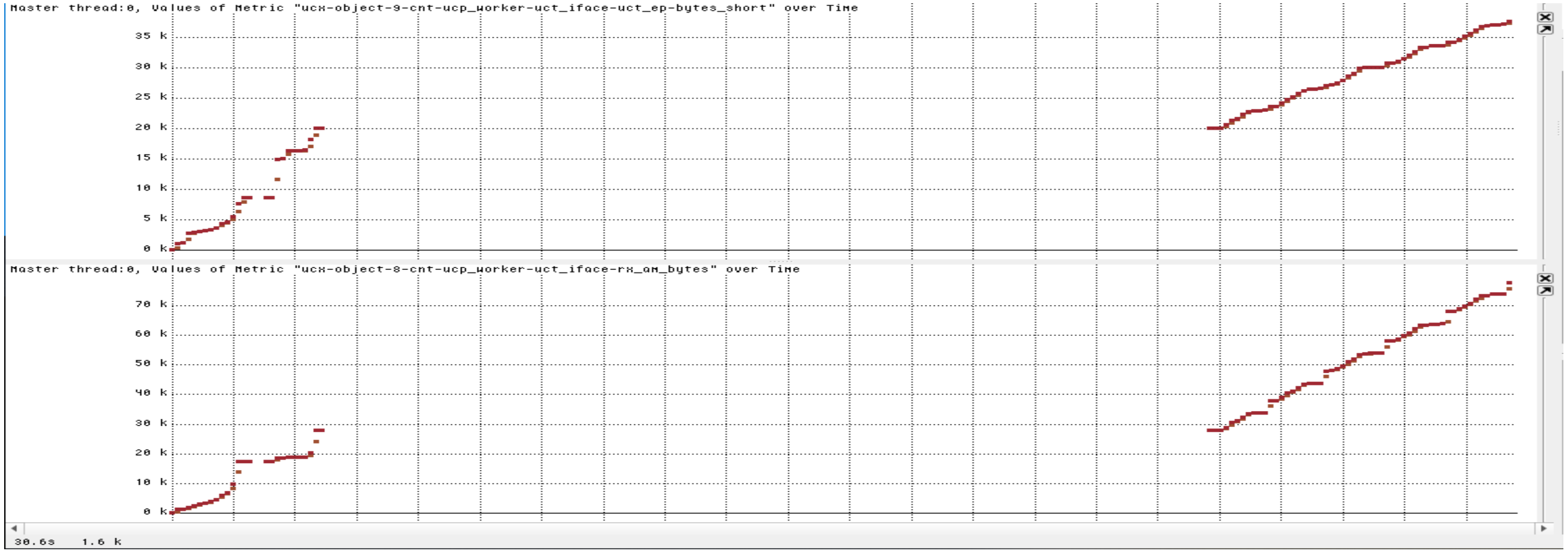
Case Study 2: WRF – Analysis

- This is a short run of **~52 seconds** on a single node of WRF with **4 processes**
- The 2 counters in the previous slide show,
  - how many RNDV RTS messages were “expected” or “unexpected”
  - If the message was “**expected**”: It can indicate that the sender of the message is slower than the receiver
  - If the message was “**unexpected**”: It can indicate that the sender of the message is faster than the receiver (and the receiver wasn’t ready before the message arrived)

➔ **We might have some room for improvement according to these counters**

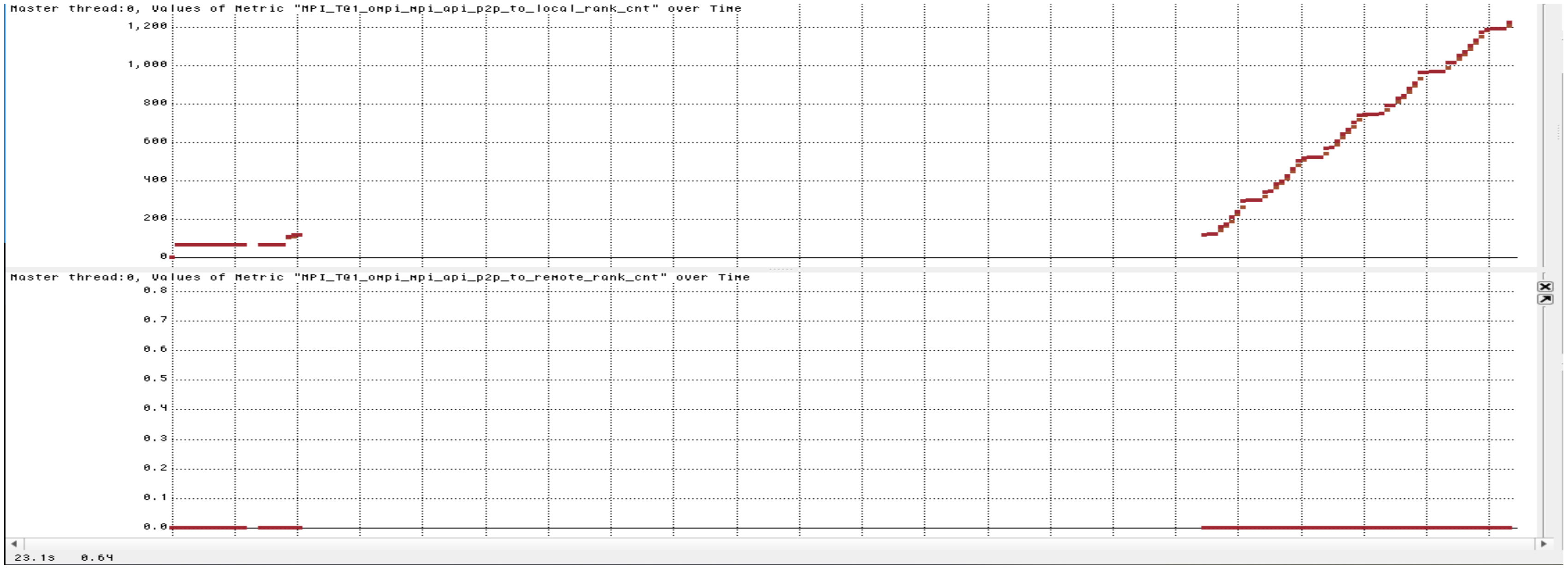
Case Study 2: WRF (EP bytes short / Rx AM bytes)

- Below, we see some communication pattern of the application
- These plots include the initialization sequence of the application



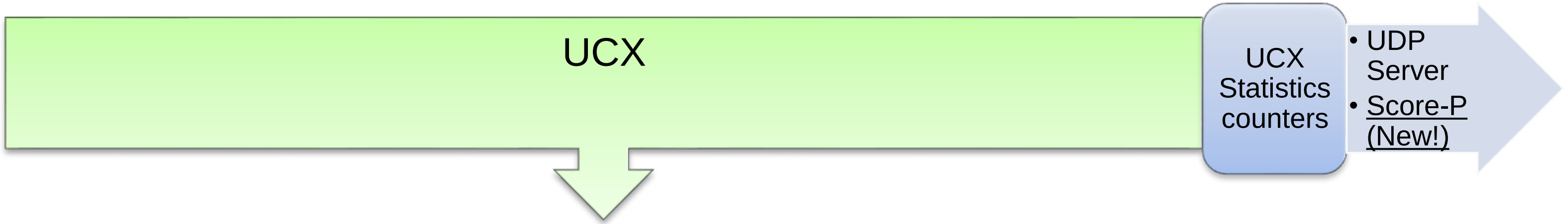
Case Study 2: WRF P2P to Local vs. Remote rank

- P2P to Local vs. Remote ranks
- Run on a single node ➔ Expected resuts (**P2P message to remote == 0**)



The new Score-P UCX Plugin

- The Score-P UCX plugin allows adding UCX statistics trace data
- The plugin has been designed such that it does not affect the application runtime significantly,
  - The **runtime overhead** is controllable (via sampling rate), **~10%** when adding 5-10 UCX metrics and trace size is around also controllable (**~1-50M per 60 seconds per process** of run in current configuration for 10 metrics)
  - The runtime overhead **grows significantly** when adding many processes
- The plugin uses the UCX statistics interface via UDP server



UCX Statistics – What can be collected? #1

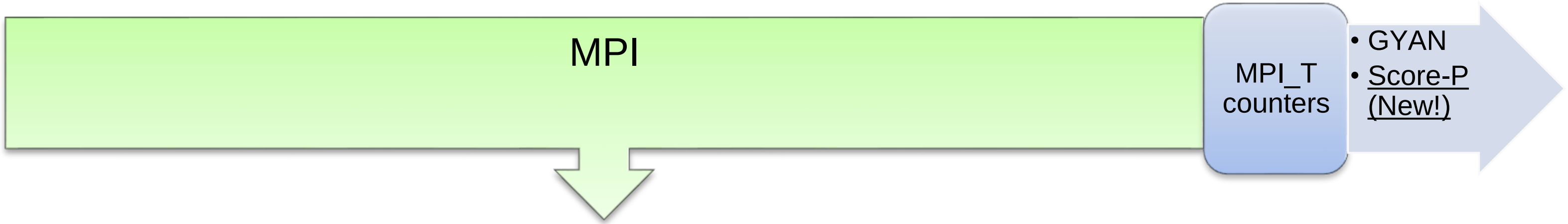
- The following UCX statistics counters can be collected per UCX interface (iface),
  - *rx_am: Number of Rx AM (Active Messages) packets*
  - *rx_am_bytes: Number of Rx AM (Active Messages) bytes*
  - *tx_no_desc: Tx initiated with a NULL descriptor*
  - *flush: How many times the interface flush operation* ***didn’t return state=IN_PROGRESS***
  - *flush_wait: How many times the interface flush operation* ***returned state=IN_PROGRESS***
  - *fence: How many times Fence operation was triggered*
  - *UD interface – rx_drop: Number of dropped Rx packets*

UCX Statistics – What can be collected? #2

- The following UCX statistics counters can be collected per UCX UCT endpoint (EP),
  - *am: Number of AM (Active Messages) packets sent*
  - *put: Number of PUT requests sent*
  - *get: Number of GET requests sent*
  - *atomic: How many atomic operations were used*
  - *bytes_short: Number of bytes sent via* ***short method***
  - *bytes_bcopy: Number of bytes sent via* ***bcopy method***
  - *bytes_zcopy: Number of bytes sent via* ***zero-copy method***
  - *flush: How many times the interface flush operation* ***didn't return state=IN_PROGRESS***
  - *flush_wait: How many times the interface flush operation* ***returned state=IN_PROGRESS***
  - *fence: How many times Fence operation was triggered*

The new Score-P MPI Plugin

- The Score-P MPI_T plugin allows adding MPI statistics trace data
- The plugin has been designed such that it does not affect the application runtime significantly,
  - The overhead is controllable (via sampling rate), **around ~5%** when adding 10 MPI metrics and trace size is around also controllable (**~1-50M per 60 seconds per process** of run in current configuration for 10 metrics)
  - The runtime overhead **grows significantly** when adding many processes
- The plugin uses the MPI_T interface and was built based on GYAN source code



The new H-MPI counters

- The following new counters were added to H-MPI,
  - MPI_Reduce() – Number of **consecutive calls** with the same send_buf or the same recv_buf
  - MPI_Allreduce() – Number of **consecutive calls** with the same send_buf or the same recv_buf
  - The number of times a P2P send was initiated to a local rank
  - The number of times a P2P send was initiated to a remote rank (not on the same node)

Score-P (Data Acquisition Tool) – How to use? #1

- Profiling an HPC (MPI) application using Score-P requires using scorep as a preprocessor before gcc (same applies to Fortran applications) during build time,
  - In practice, the user needs to update the Makefile of the application to set,

    *GCC=**scorep** mpicc*
    For example,
    $ cat makefile
    EXECS=reduce_avg reduce_stddev
    *MPICC?=**scorep** mpicc*

Score-P (Data Acquisition Tool) – How to use? #2

- Following rebuild, execute the MPI application as follows (this is without a plugin),

```
# Enable tracing
export SCOREP_ENABLE_TRACING=true
export SCOREP_ENABLE_PROFILING=true
export SCOREP_TOTAL_MEMORY=1000M
make clean && make && mpirun -n 2 reduce_stddev 1000
```

Score-P (Data Acquisition Tool) – How to use? #2

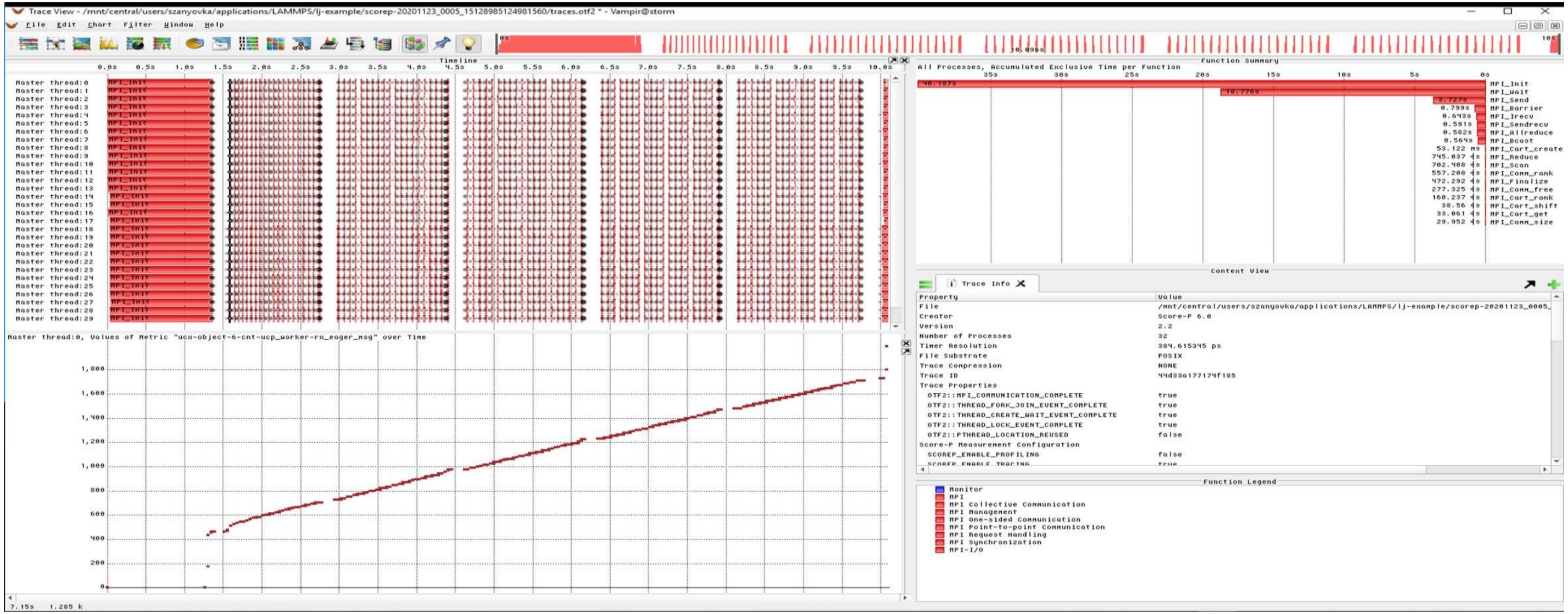
- Following the execution of the application built with the Score-P preprocessor,
  the **scorep-*** directory will appear,

```
$ ll
drwxr-xr-x. 3 shukiz shukiz     115 Aug  2 23:38 scorep-20200802_2338_921672935011334
```

- The scorep-* directory contains the profiling and trace data
- In order to analyze this data there are multiple tools provided in the VI-HPS toolset
- Some of the different tools will be described in the next slides

Main Tools: Vampir Trace Visualization Tool

- Vampir is an advanced trace data visualization and analysis tool,


Trace View - /mnt/central/users/szanyovka/applications/LAMMPS/lj-example/scorep-20201123_0005_15128985124981560/traces.otf2 * - Vampir@storm

File Edit Chart Filter Window Help
Timeline
Master thread:0 … Master thread:29
MPI_Init
Master thread:0, Values of Metric "ucx-object-6-cnt-ucp_worker-rx_eager_msg" over Time
7.15s 1.285 k

Function Summary

All Processes, Accumulated Exclusive Time per Function

Time	Function
40.107s	MPI_Init
18.776s	MPI_Wait
3.727s	MPI_Send
0.799s	MPI_Barrier
0.643s	MPI_Irecv
0.591s	MPI_Sendrecv
0.582s	MPI_Allreduce
0.564s	MPI_Bcast
53.122 ms	MPI_Cart_create
745.037 µs	MPI_Reduce
702.408 µs	MPI_Scan
557.208 µs	MPI_Comm_rank
472.292 µs	MPI_Finalize
277.325 µs	MPI_Comm_free
160.237 µs	MPI_Cart_rank
38.56 µs	MPI_Cart_shift
33.061 µs	MPI_Cart_get
29.952 µs	MPI_Comm_size

Context View

Trace Info

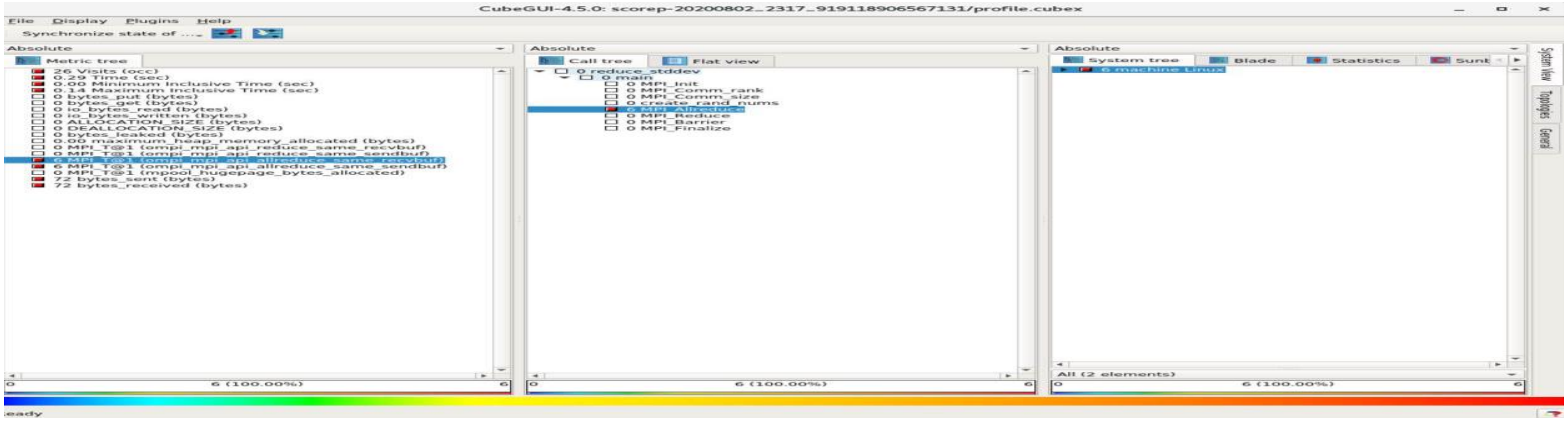
Property	Value
File	/mnt/central/users/szanyovka/applications/LAMMPS/lj-example/scorep-20201123_0005_
Creator	Score-P 6.0
Version	2.2
Number of Processes	32
Timer Resolution	384.615345 ps
File Substrate	POSIX
Trace Compression	NONE
Trace ID	44d33a177174f185
Trace Properties	
OTF2::MPI_COMMUNICATION_COMPLETE	true
OTF2::THREAD_FORK_JOIN_EVENT_COMPLETE	true
OTF2::THREAD_CREATE_WAIT_EVENT_COMPLETE	true
OTF2::THREAD_LOCK_EVENT_COMPLETE	true
OTF2::PTHREAD_LOCATION_REUSED	false
Score-P Measurement Configuration	
SCOREP_ENABLE_PROFILING	false
SCOREP_ENABLE_TRACING	true

Function Legend

- 🟦 Monitor
- 🟥 MPI
- 🟥 MPI Collective Communication
- 🟥 MPI Management
- 🟥 MPI One-sided Communication
- 🟥 MPI Point-to-point Communication
- 🟥 MPI Request Handling
- 🟥 MPI Synchronization
- 🟥 MPI-I/O

Main Tools: Cube Profile & Trace Data Visualization Tool

- Cube is an extendible VI-HPS profiling & trace data visualization and analysis tool
- The input to Cube is the Score-P profiling data
- To execute Cube,
  $ cube scorep-20200803_1508_956100066319885/profile.cubex



CubeGUI-4.5.0: scorep-20200802_2317_919118906567131/profile.cubex

```
26 Visits (occ)
0.29 Time (sec)
0.00 Minimum Inclusive Time (sec)
0.14 Maximum Inclusive Time (sec)
0 bytes_put (bytes)
0 bytes_get (bytes)
0 io_bytes_read (bytes)
0 io_bytes_written (bytes)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
0 bytes_leaked (bytes)
0.00 maximum_heap_memory_allocated (bytes)
0 MPI_T@1 (ompi_mpi_api_reduce_same_recvbuf)
0 MPI_T@1 (ompi_mpi_api_reduce_same_sendbuf)
6 MPI_T@1 (ompi_mpi_api_allreduce_same_recvbuf)
6 MPI_T@1 (ompi_mpi_api_allreduce_same_sendbuf)
0 MPI_T@1 (mpool_hugepage_bytes_allocated)
72 bytes_sent (bytes)
72 bytes_received (bytes)
```

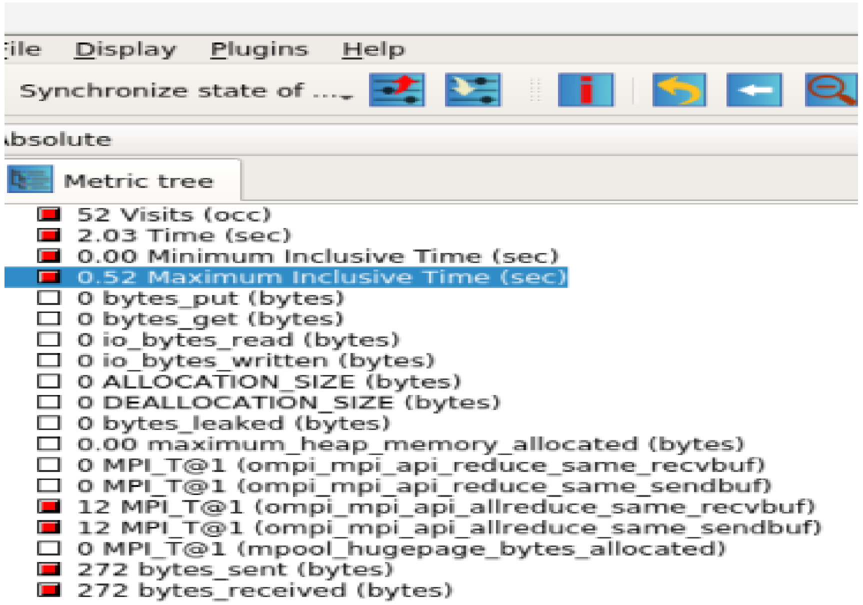
```
0 reduce_stddev
  0 main
    0 MPI_Init
    0 MPI_Comm_rank
    0 MPI_Comm_size
    0 create_rand_nums
    6 MPI_Allreduce
    0 MPI_Reduce
    0 MPI_Barrier
    0 MPI_Finalize
```

```
6 machine Linux
```

```
0          6 (100.00%)          6
All (2 elements)
```

Main Tools: Cube Profile & Trace Data Visualization Tool

- Cube is extendible via plugins that can be developed to analyze the trace and profiling data
- For example, Blade Trace visualization (which can probably be extended),



52 Visits (occ)
2.03 Time (sec)
0.00 Minimum Inclusive Time (sec)
0.52 Maximum Inclusive Time (sec)
0 bytes_put (bytes)
0 bytes_get (bytes)
0 io_bytes_read (bytes)
0 io_bytes_written (bytes)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
0 bytes_leaked (bytes)
0.00 maximum_heap_memory_allocated (bytes)
0 MPI_T@1 (ompi_mpi_api_reduce_same_recvbuf)
0 MPI_T@1 (ompi_mpi_api_reduce_same_sendbuf)
12 MPI_T@1 (ompi_mpi_api_allreduce_same_recvbuf)
12 MPI_T@1 (ompi_mpi_api_allreduce_same_sendbuf)
0 MPI_T@1 (mpool_hugepage_bytes_allocated)
272 bytes_sent (bytes)
272 bytes_received (bytes)

Challenges developing the plugins #1

- **The problem:**
  MPI_Init() is called after Score-P Plugin initialization was called which yields a chicken and egg problem,
  - The plugin is not able to dynamically enumerate the MPI counters before MPI was initialized
  - Therefore, it is not able to supply the metric names and id's to the Score-P framework in time

- **The current solution (temporary workaround),**
  - Add a static table of all supported MPI counters names and their OMPI Id's to the plugin and not use dynamic enumeration at all

Challenges developing the plugins #2

- **The problem:**
  The UCX statistics has a variable number of interfaces (and therefore, a variable number of metrics) per MPI run topology configuration**,**
  - Each number of nodes and cluster configuration can yield a different set of objects
  - Also, a **chicken and egg** problem since UCX counters are initialized only in a later stage of the application initialization sequence

- **The current solution (temporary workaround),**
  - Employ UCX statistics filtering to reduce overhead in trace data and runtime of the application
  - We added N temporary counters (configurable via the metric Score-P plugin environment variable, e.g. UCX@30) **which will later be renamed**
  - Using the get_optional_value() function of the plugin to sample the UCX Statistics – **When it becomes available, the temporary allocated Score-P counters are renamed**

Challenges developing the plugins #3

- **The problem:**
  The UCX statistics is based on triggering snapshot by the user via calling API and was mainly built for profiling at the end of the application but not for collecting trace data
  - The overhead of taking a snapshot of UCX counters is currently high

- **The current solution (temporary workaround),**
  - Apply a low sampling rate to reduce the data acquisition overhead

Collaboration Opportunities

- **Score-P,**
  - Plugin to defer its initialization
  - Data acquisition: In 1 call rather than N calls
  - Plugin to dynamically add new counters to the Score-P framework
- **MPI and UCX,**
  - Introduce new MPI and UCX counters for better HPC applications profiling

HUAWEI | TEL AVIV RESEARCH CENTER

Plugins on github

- **Updated Score-P (for UCX plugin),**
  - https://github.com/shuki-zanyovka/scorep_6.0
  - Under branch: **huawei-ucx-workaround**

- **MPI and UCX plugins,**
  - https://github.com/shuki-zanyovka/scorep_plugin_mpi
  - https://github.com/shuki-zanyovka/scorep_plugin_ucx

HUAWEI

www.huawei.com