

Performance Tools for Heterogeneous Parallel Systems and Applications

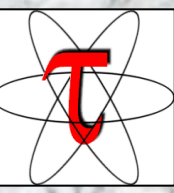
Sameer Shende, Allen D. Malony, Wyatt Spear, Scott Biersdorff, Chee Wai Lee

Performance Research Laboratory

University of Oregon

<http://tau.uoregon.edu/tau.ppt>

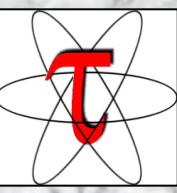




Motivation

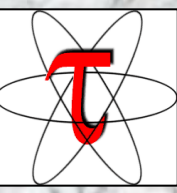
- Heterogeneous parallel systems are highly relevant today
- Heterogeneous hardware technology more accessible
 - Multicore processors (e.g., 4-core, 6-core, 8-core, ...)
 - Manycore (throughput) accelerators (e.g., Tesla, Fermi)
 - High-performance engines (e.g., Cell BE, Larrabee)
 - Special purpose components (e.g., FPGAs)
- Performance is the main driving concern
 - Heterogeneity is an important (the?) path to extreme scale
- Heterogeneous software technology required for performance
 - More sophisticated parallel programming environments
 - Integrated parallel performance tools
 - support heterogeneous performance model and perspectives

Implications for Parallel Performance Tools



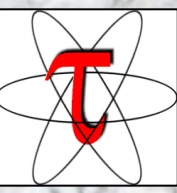
- Current status quo is somewhat comfortable
 - Mostly homogeneous parallel systems and software
 - Shared-memory multithreading – OpenMP
 - Distributed-memory message passing – MPI
- Parallel computational models are relatively stable (simple)
 - Corresponding performance models are relatively tractable
 - Parallel performance tools can keep up and evolve
- Heterogeneity creates richer computational potential
 - Results in greater performance diversity and complexity
- Heterogeneous systems will utilize more sophisticated programming and runtime environments
- Performance tools have to support richer computation models and more versatile performance perspectives

Heterogeneous Performance Views

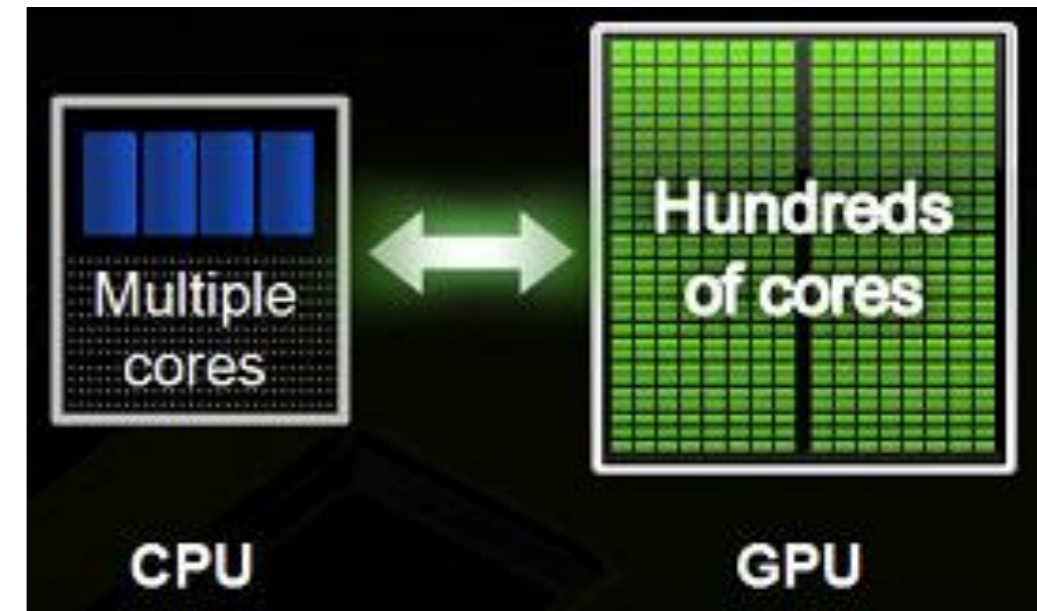


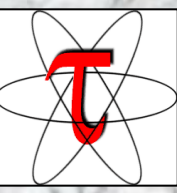
- Want to create performance views that capture heterogeneous concurrency and execution behavior
 - Reflect interactions between heterogeneous components
 - Capture performance semantics relative to computation model
 - Assimilate performance for all execution paths for shared view
- Existing parallel performance tools are CPU(host)-centric
 - Event-based sampling (not appropriate for accelerators)
 - Direct measurement (through instrumentation of events)
- What perspective does the host have of other components?
 - Determines the semantics of the measurement data
 - Determines assumptions about behavior and interactions
- Performance views may have to work with reduced data

Task-based Performance View



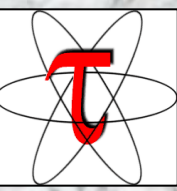
- ❑ Consider the “task” abstraction for GPU accelerator scenario
- ❑ Host regards external execution as a task
 - Tasks operate concurrently with respect to the host
 - Requires support for tracking asynchronous execution
- ❑ Host creates measurement perspective for external task
 - Maintains local and remote performance data
 - Tasks may have limited measurement support
 - May depend on host for performance data I/O
 - Performance data might be received from external task
- ❑ How to create a view of heterogeneous external performance?



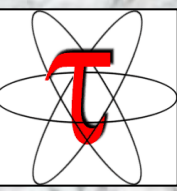


- CUDA enables programming of kernels for GPU acceleration
- GPU acceleration acts as an external tasks
- Performance measurement appears straightforward
- Execution model complicates performance measurement
 - Synchronous and asynchronous operation with respect to host
 - Overlapping of data transfer and kernel execution
 - Multiple GPU devices and multiple streams per device
- Different acceleration kernels used in parallel application
 - Multiple application sections
 - Multiple application threads/processes
 - See performance in context:
 - temporal, spatial, (host) thread/process

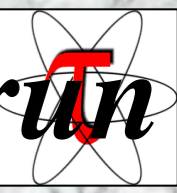
TAU: A Brief Introduction



- ❑ TAU is a performance evaluation tool
- ❑ It supports parallel profiling and tracing
- ❑ Profiling shows you how much (total) time was spent in each routine
- ❑ Tracing shows you *when* the events take place in each process along a timeline
- ❑ TAU uses a package called PDT for automatic instrumentation of the source code
- ❑ Profiling and tracing can measure time as well as hardware performance counters from your CPU
- ❑ TAU can automatically instrument your source code (routines, loops, I/O, memory, phases, etc.)
- ❑ TAU runs on all HPC platforms and it is free (BSD style license)
- ❑ TAU has instrumentation, measurement and analysis tools
 - paraprof is TAU's 3D profile browser
- ❑ To use TAU's automatic source instrumentation, you need to set a couple of environment variables and substitute the name of your compiler with a TAU shell script

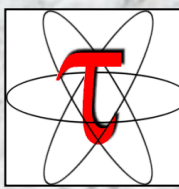


- TAU supports several measurement options (profiling, tracing, profiling with hardware counters, etc.)
 - Each measurement configuration of TAU corresponds to a unique stub makefile and library that is generated when you configure it
 - To instrument source code using PDT
 - Choose an appropriate TAU stub makefile in <arch>/lib:
% **setenv TAU_MAKEFILE \$TAU/Makefile.tau-mpi-pdt**
 - % **setenv TAU_OPTIONS '-optVerbose ...'** (see `tau_compiler.sh -help`)
- And use `tau_f90.sh`, `tau_cxx.sh` or `tau_cc.sh` as Fortran, C++ or C compilers:
- % **mpif90 foo.f90**
- changes to
- % **tau_f90.sh foo.f90**
- Execute application and analyze performance data:
 - % **pprof** (for text based profile display)
 - % **paraprof** (for GUI)

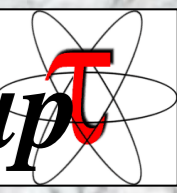


- TAU and DyninstAPI are mature technologies for performance instrumentation, measurement and analysis
- TAU has been a long-time user of DyninstAPI
- Using DyninstAPI's recent binary re-writing capabilities, created a binary re-writer tool for TAU (*tau_run*)
 - Supports TAU's performance instrumentation
 - Works with TAU instrumentation selection
 - files and routines based on exclude/include lists
 - TAU's measurement library (DSO) is loaded by *tau_run*
- Runtime (pre-execution) and binary re-writing are both supported
- Simplifies code instrumentation and tool usage greatly!
- Included on POINT LiveDVD (tau.uoregon.edu/point.iso)

tau_run with NAS PBS

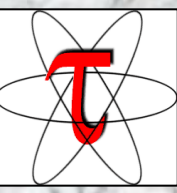


```
livetau@paratools01:~  
/home/livetau% cd ~/tutorial  
/home/livetau/tutorial% # Build an uninstrumented bt NAS Parallel Benchmark  
/home/livetau/tutorial% make bt CLASS=W NPROCS=4  
/home/livetau/tutorial% cd bin  
/home/livetau/tutorial/bin% # Run the instrumented code  
/home/livetau/tutorial/bin% mpirun -np 4 ./bt_W.4  
/home/livetau/tutorial/bin%  
/home/livetau/tutorial/bin% # Instrument the executable using TAU with DyninstAPI  
/home/livetau/tutorial/bin%  
/home/livetau/tutorial/bin% tau_run ./bt_W.4 -o ./bt.i  
/home/livetau/tutorial/bin% rm -rf profile.* MULT*  
/home/livetau/tutorial/bin% mpirun -np 4 ./bt.i  
/home/livetau/tutorial/bin% paraprof  
/home/livetau/tutorial/bin%  
/home/livetau/tutorial/bin% # Choose a different TAU configuration  
/home/livetau/tutorial/bin% ls $TAU/libTAUsh  
libTAUsh-depthlimit-mpi-pdt.so*      libTAUsh-papi-pdt.so*  
libTAUsh-mpi-pdt.so*                libTAUsh-papi-pthread-pdt.so*  
libTAUsh-mpi-pdt-upc.so*            libTAUsh-param-mpi-pdt.so*  
libTAUsh-mpi-python-pdt.so*         libTAUsh-pdt.so*  
libTAUsh-papi-mpi-pdt.so*           libTAUsh-pdt-trace.so*  
libTAUsh-papi-mpi-pdt-upc.so*       libTAUsh-phase-papi-mpi-pdt.so*  
libTAUsh-papi-mpi-pdt-upc-udp.so*   libTAUsh-pthread-pdt.so*  
libTAUsh-papi-mpi-pdt-vampirtrace-trace.so* libTAUsh-python-pdt.so*  
libTAUsh-papi-mpi-python-pdt.so*  
/home/livetau/tutorial/bin%  
/home/livetau/tutorial/bin% tau_run -XrunTAUsh-papi-mpi-pdt-vampirtrace-trace bt_W.4 -o bt.vpt  
/home/livetau/tutorial/bin% setenv VT_METRICS PAPI_FP_INS:PAPI_L1_DCM  
/home/livetau/tutorial/bin% mpirun -np 4 ./bt.vpt  
/home/livetau/tutorial/bin% vampir bt.vpt.otf &
```



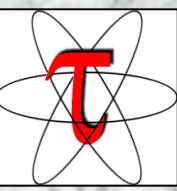
- ❑ Performance evaluation tools such as TAU provide a wealth of options to measure the performance of an application
- ❑ Need to simplify TAU usage to easily evaluate performance properties, including I/O, memory, and communication
- ❑ Designed a new tool (*tau_exec*) that leverages runtime instrumentation by pre-loading measurement libraries
- ❑ Works on dynamic executables (default under Linux)
- ❑ Substitutes I/O, MPI, and memory allocation/deallocation routines with instrumented calls
 - Interval events (e.g., time spent in write())
 - Atomic events (e.g., how much memory was allocated)
- ❑ Measure I/O and memory usage

TAU Execution Command (tau_exec)



- Uninstrumented execution
 - % mpirun -np 256 ./a.out
- Track MPI performance
 - % mpirun -np 256 **tau_exec** ./a.out
- Track I/O and MPI performance (MPI enabled by default)
 - % mpirun -np 256 **tau_exec** **-io** ./a.out
- Track memory operations
 - % setenv TAU_TRACK_MEMORY_LEAKS 1
 - % mpirun -np 256 **tau_exec** **-memory** ./a.out
- Track I/O performance and memory operations
 - % mpirun -np 256 **tau_exec** **-io** **-memory** ./a.out
- **Track GPGPU operations**
 - % mpirun -np 256 **tau_exec** **-cuda** ./a.out

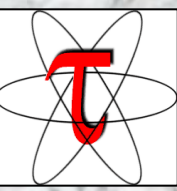
tau_exec: A tool to simplify Memory, I/O evaluation



```
xterm
> cd ~/workshop-point/matmult
> mpif90 matmult.f90 -o matmult
> mpirun -np 4 ./matmult
>
> # To use tau_exec to measure the I/O and memory usage:
> mpirun -np 4 tau_exec -io -memory ./matmult
>
> # To measure memory leaks and get complete callpaths
> setenv TAU_TRACK_MEMORY_LEAKS 1
> setenv TAU_CALLPATH_DEPTH 100
> mpirun -np 4 tau_exec -io -memory ./matmult
> paraprof
> # Right click on a given rank (e.g. "node 2") and choose "Show Context Event
> # Window" and expand the ".TAU Application" node to see the callpath
> # To use a different configuration (e.g., Makefile.tau-papi-mpi-pdt)
> setenv TAU_METRICS TIME:PAPI_FP_INS:PAPI_L1_DCM
> mpirun -np 4 tau_exec -io -memory -T papi,mpi,pdt ./matmult
> # Using tau_exec with DyninstAPI:
> tau_run matmult -o matmult.i
> mpirun -np 4 tau_exec -io -memory ./matmult.i
>
> tau_run -XrunTAUsh-papi-mpi-pdt matmult -o matmult.i
> mpirun -np 4 tau_exec -io -memory -T papi,mpi,pdt ./matmult.i
> paraprof █
```

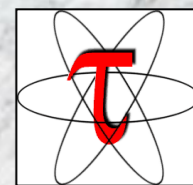
//

Environment Variables in TAU



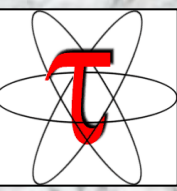
Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_TRACK_MEMORY_LEAKS	0	Setting to 1 turns on leak detection
TAU_TRACK_HEAP or TAU_TRACK_HEADROOM	0	Setting to 1 turns on tracking heap memory/headroom at routine entry & exit using context events (e.g., Heap at Entry: main=>foo=>bar)
TAU_CALLPATH_DEPTH	2	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)
TAU_SYNCHRONIZE_CLOCKS	1	Synchronize clocks across nodes to correct timestamps in traces
TAU_COMM_MATRIX	0	Setting to 1 generates communication matrix display using context events
TAU_THROTTLE	1	Setting to 0 turns off throttling. Enabled by default to remove instrumentation in lightweight routines that are called frequently
TAU_THROTTLE_NUMCALLS	100000	Specifies the number of calls before testing for throttling
TAU_THROTTLE_PERCALL	10	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
TAU_COMPENSATE	0	Setting to 1 enables runtime compensation of instrumentation overhead
TAU_PROFILE_FORMAT	Profile	Setting to "merged" generates a single file. "snapshot" generates xml format
TAU_METRICS	TIME	Setting to a comma separated list generates other metrics. (e.g., TIME:linuxtimers:PAPI_FP_OPS:PAPI_NATIVE_<event>)

Memory Leaks in MPI



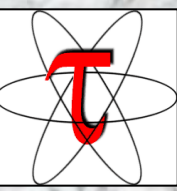
TAU: ParaProf: Context Events for thread: n,c,t, 0,0,0 - samarc_obe_4p_iomem_cp.ppk

Name	Total	MeanValue	NumSamples	MaxValue	MinValue	Std. Dev.
▼ .TAU application						
▼ MPI_Finalize()						
free size	23,901,253	22,719.822	1,052	2,099,200	2	186,920.948
malloc size	5,013,902	65,972.395	76	5,000,000	2	569,732.815
MEMORY LEAK!	5,000,264	500,026.4	10	5,000,000	3	1,499,991.2
▼ read()						
Bytes Read	4	4	1	4	4	0
READ Bandwidth (MB/s) <file="pipe">		0.308	1	0.308	0.308	0
Bytes Read <file="pipe">	4	4	1	4	4	0
READ Bandwidth (MB/s)		0.308	1	0.308	0.308	0
▼ write()						
WRITE Bandwidth (MB/s)		0.635	102	12	0	1.472
Bytes Written <file="/dev/infiniband/rdma_cm">	24	24	1	24	24	0
Bytes Written	1,456	14.275	102	28	4	5.149
WRITE Bandwidth (MB/s) <file="/dev/infiniband/uverbs0">		0.528	97	12	0.089	1.32
Bytes Written <file="pipe">	64	16	4	28	4	12
WRITE Bandwidth (MB/s) <file="/dev/infiniband/rdma_cm">		1.714	1	1.714	1.714	0
Bytes Written <file="/dev/infiniband/uverbs0">	1,368	14.103	97	24	12	4.562
WRITE Bandwidth (MB/s) <file="pipe">		2.967	4	5.6	0	2.644
▼ writev()						
WRITE Bandwidth (MB/s)		4.108	2	7.667	0.549	3.559
Bytes Written	297	148.5	2	230	67	81.5
WRITE Bandwidth (MB/s) <file="socket">		4.108	2	7.667	0.549	3.559
Bytes Written <file="socket">	297	148.5	2	230	67	81.5
▼ readv()						
Bytes Read	112	28	4	36	20	8
READ Bandwidth (MB/s) <file="socket">		25.5	4	36	10	11.079
Bytes Read <file="socket">	112	28	4	36	20	8
READ Bandwidth (MB/s)		25.5	4	36	10	11.079
▼ MPI_Comm_free()						
free size	10,952	195.571	56	1,024	48	255.353
▶ read()						
▶ MPI_Type_free()						
▶ MPI_Init()						
▼ fopen64()						
free size	231,314	263.456	878	568	35	221.272
MEMORY LEAK!	1,105,956	1,868.169	592	7,200	32	3,078.574
malloc size	1,358,286	901.318	1,507	7,200	32	2,087.737
▶ OurMain()						
▶ fclose()						

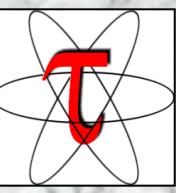


- ❑ Source code
 - Manual (TAU API, TAU component API)
 - Automatic (robust)
 - C, C++, F77/90/95 (Program Database Toolkit (PDT))
 - OpenMP (directive rewriting (Opari), POMP2 spec)
 - Library header wrapping
- ❑ Object code
 - Pre-instrumented libraries (e.g., MPI using PMPI)
 - Statically- and dynamically-linked (with LD_PRELOAD)
- ❑ Executable code
 - Binary and dynamic instrumentation (Dyninst)
 - Virtual machine instrumentation (e.g., Java using JVMPI)
- ❑ TAU compiler scripts to automate instrumentation process

Library wrapping: tau_wrap -r <library.so>



- ❑ How to instrument an external library without source?
 - Source may not be available
 - Library may be too cumbersome to build (with instrumentation)
- ❑ Build a library wrapper tools
 - Used PDT to parse header files
 - Generate new header files with instrumentation files
 - Library loads the original library using the dlopen() call
- ❑ Application is instrumented
- ❑ Add the -I<wrapper> directory to the command line
- ❑ C pre-processor will substitute our headers
 - Redirects references at routine callsite to a wrapper call
 - Wrapper internally calls the original
 - Wrapper has TAU measurement code



HDF5 Library Wrapping

```
[sameer@zorak]$ tau_wrap hdf5.h.pdb hdf5.h -o hdf5.inst.c -f select.tau -g hdf5 -r libhdf5.so; cd wrapper; make
```

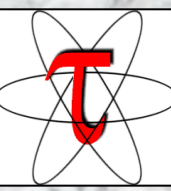
Usage : **tau_wrap** <pdbfile> <sourcefile> [-o <outputfile>] [-r runtime libname] [-g groupname] [-i headerfile] [-c|-c++|-fortran] [-f <instr_req_file>]

- instrumented wrapper library source (hdf5.inst.c)
- instrumentation specification file (select.tau)
- group (hdf5)
- tau_exec loads libhdf5_wrap.so shared library using LD_PRELOAD
- creates the wrapper/ directory

NODE 0;CONTEXT 0;THREAD 0:

%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive Name
				usec/call	
100.0	0.057	1	1	13	1236 .TAU Application
70.8	0.875	0.875	1	0	875 hid_t H5Fcreate()
9.7	0.12	0.12	1	0	120 herr_t H5Fclose()
6.0	0.074	0.074	1	0	74 hid_t H5Dcreate()
3.1	0.038	0.038	1	0	38 herr_t H5Dwrite()
2.6	0.032	0.032	1	0	32 herr_t H5Dclose()
2.1	0.026	0.026	1	0	26 herr_t H5check_version()
0.6	0.008	0.008	1	0	8 hid_t H5Screate_simple()
0.2	0.002	0.002	1	0	2 herr_t H5Tset_order()
0.2	0.002	0.002	1	0	2 hid_t H5Tcopy()
0.1	0.001	0.001	1	0	1 herr_t H5Sclose()
0.1	0.001	0.001	2	0	0 herr_t H5open()
0.0	0	0	1	0	0 herr_t H5Tclose()

TAU and TAUcuda

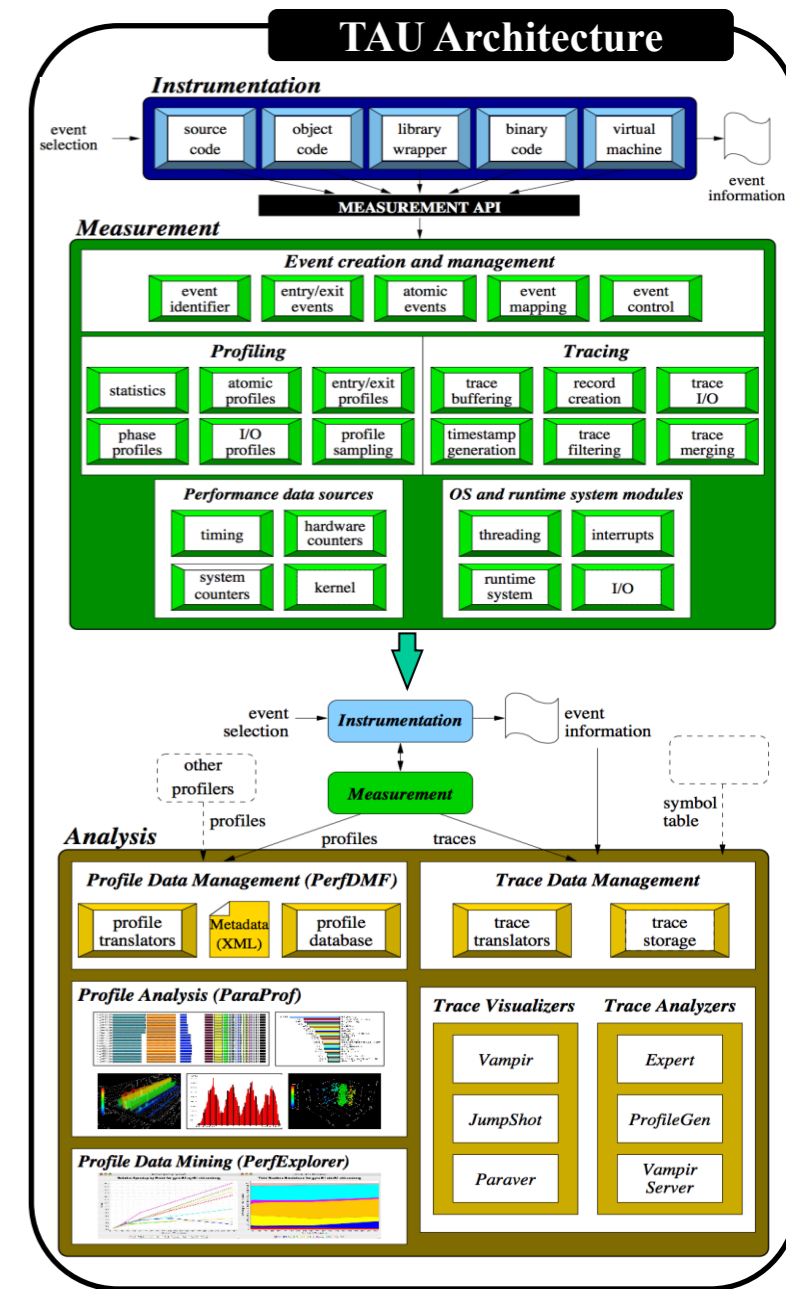


TAU performance system

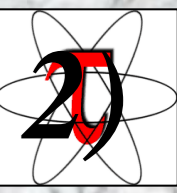
- Robust, scalable integrated performance framework and toolkit
- Parallel profiling and tracing
- Shared and distributed parallel systems
- Open source and portable

TAUcuda

- Extension to support CUDA performance measurement
- Goal is to leverage TAU's infrastructure and analysis capabilities in TAUcuda development
- Deliver heterogeneous parallel performance support

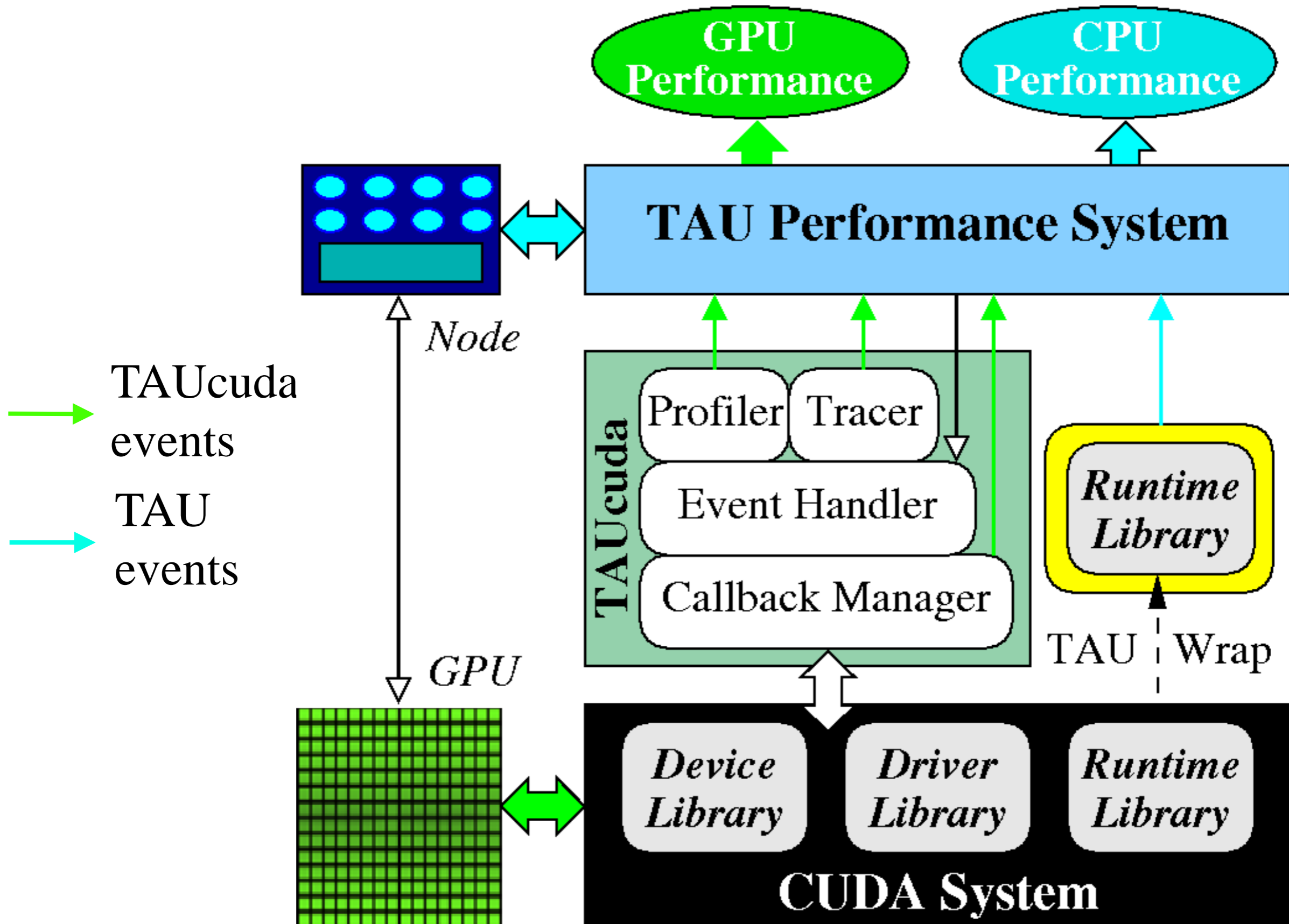
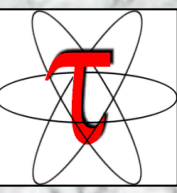


TAUcuda Performance Measurement (Version 2)

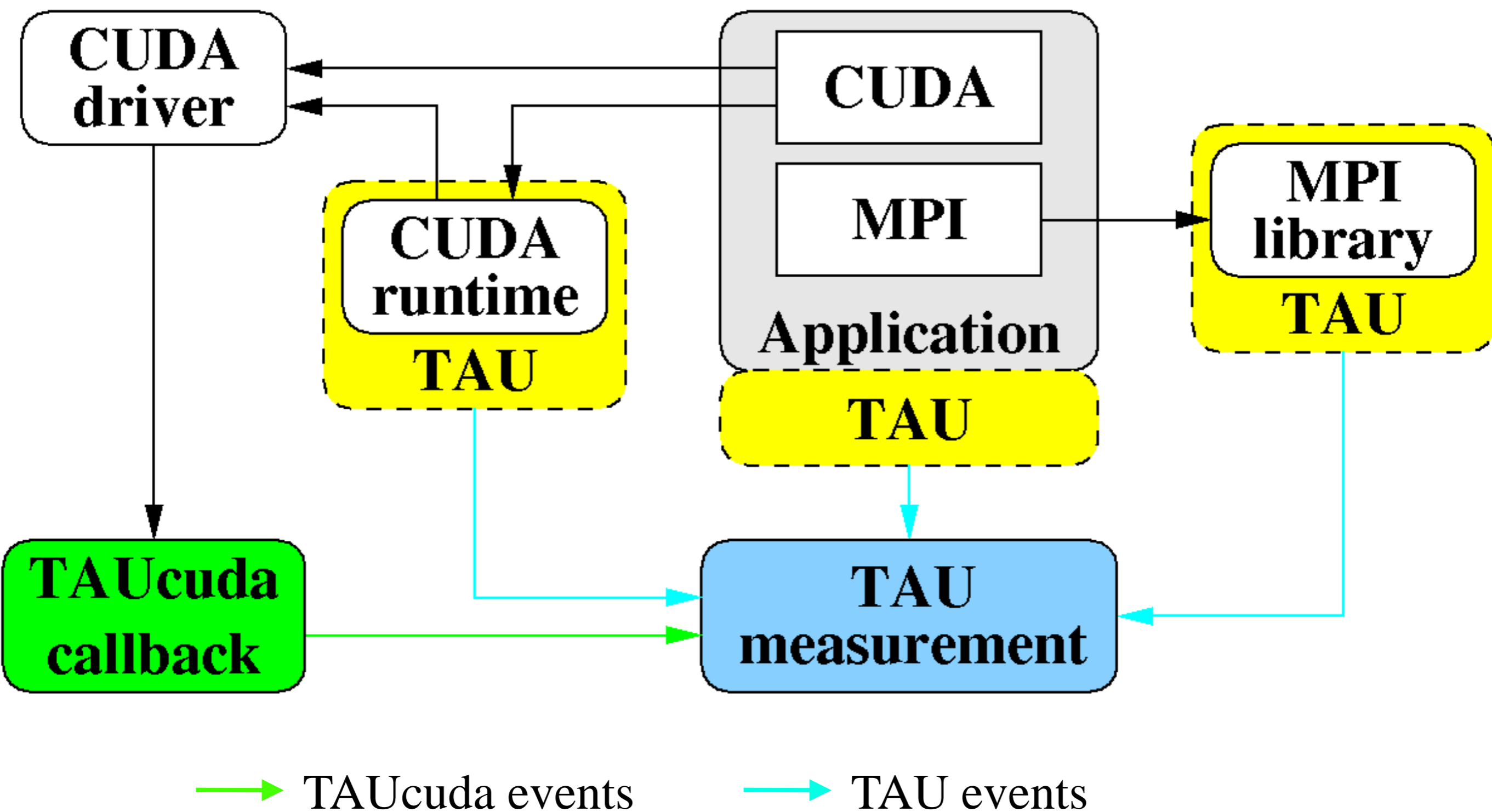
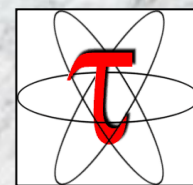


- ❑ Performance measurement of heterogeneous applications using CUDA
- ❑ CUDA system architecture
 - Implemented by CUDA libraries
 - driver and device (*cuXXX*) libraries
 - runtime (*cudaYYY*) library
 - Tools support (Parallel Nsight (Nexus), CUDA Profiler)
 - not intended to integrate with other HPC performance tools
- ❑ TAUcuda (v2) built on experimental Linux CUDA driver
 - Linux CUDA driver R190.86 supports a callback interface!!!
- ❑ Currently working with NVIDIA to develop version with production performance tools API

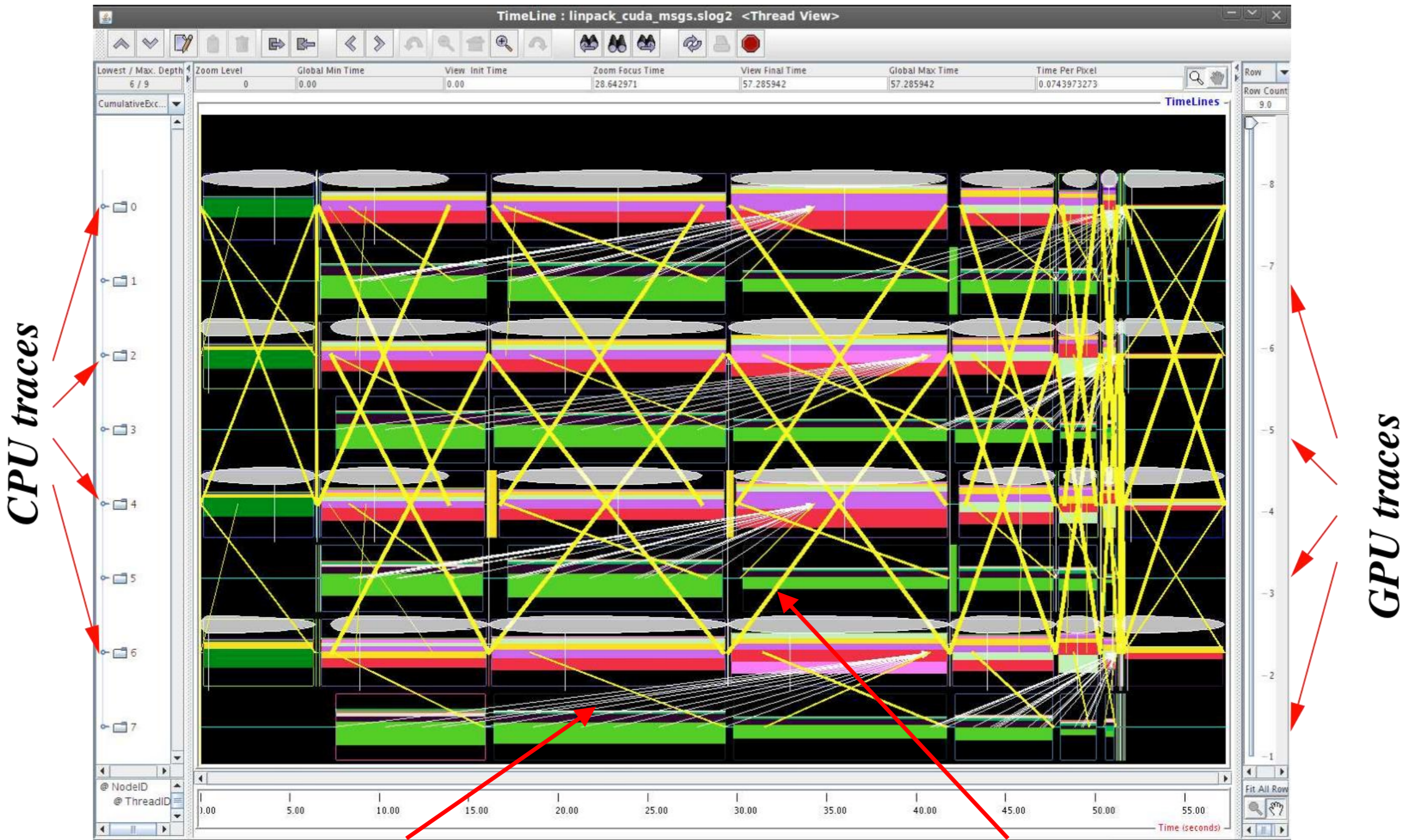
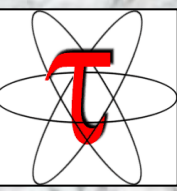
TAUcuda Architecture



TAUcuda Instrumentation

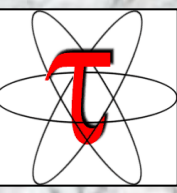


CUDA Linpack Trace

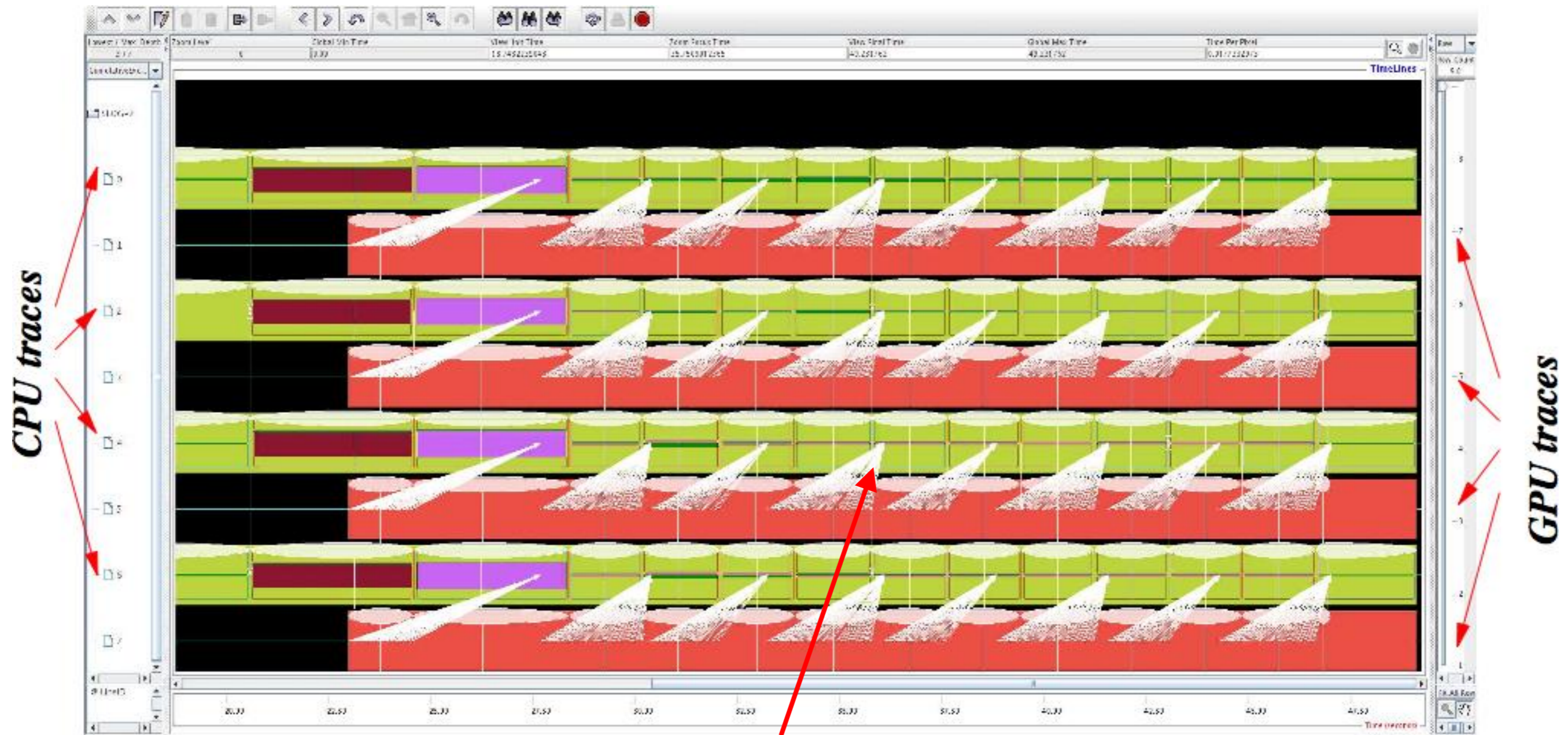


CUDA memory transfer (white) MPI communication (yellow)

SHOC Stencil2D (512 iterations, 4 CPUxGPU)

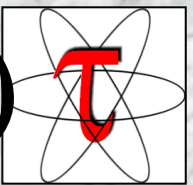


- Scalable Heterogeneous Computing benchmark suite
 - CUDA / OpenCL kernels and microbenchmarks (ORNL)

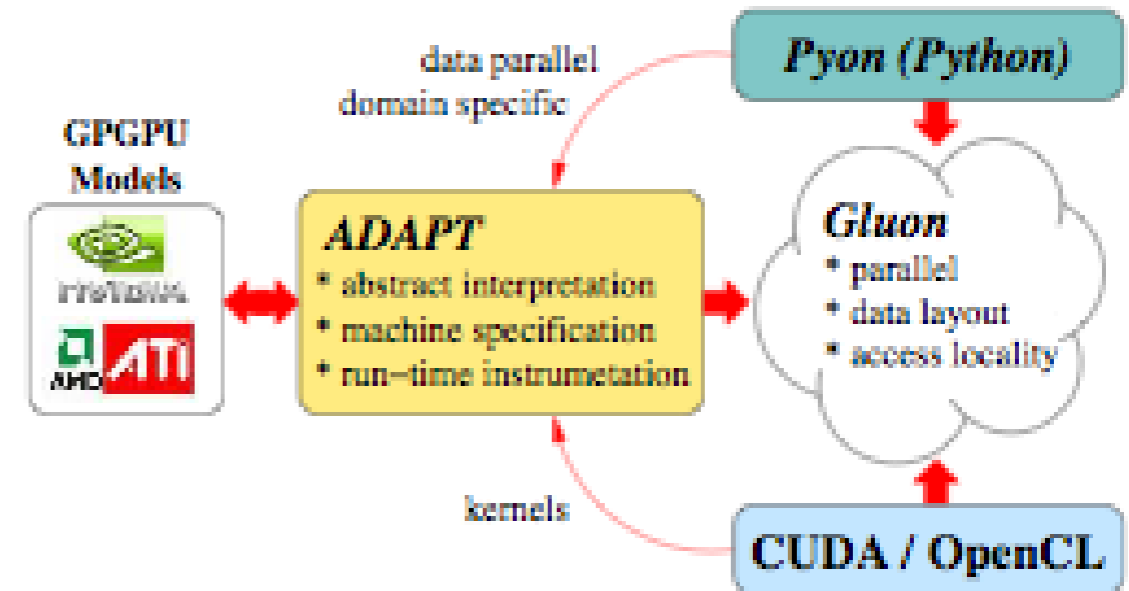


CUDA memory transfer (white)

Heterogeneous Exascale Software (Vancouver)

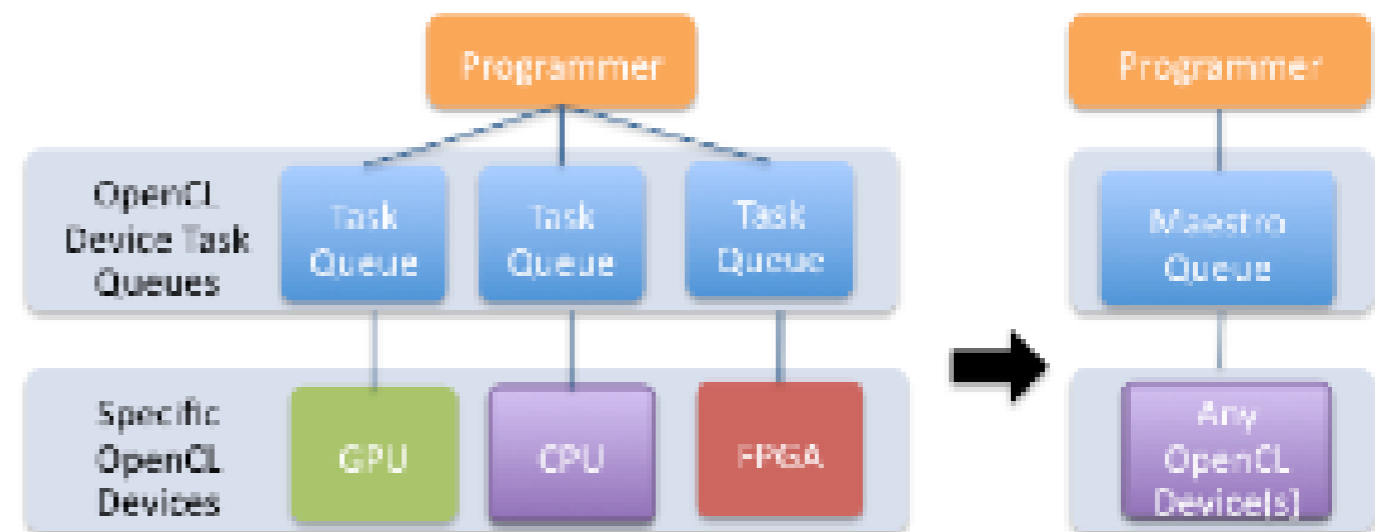


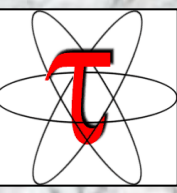
- DOE X-stack program
- Partners:
 - Oak Ridge National Laboratory
 - University of Oregon
 - University of Illinois
 - Georgia Institute of Technology



□ Components

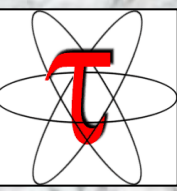
- Compilers
- Scheduling and runtime resource management
- Libraries
- Performance measurement, analysis, modeling





- ❑ Heterogeneous parallel systems will require parallel performance tools that integrate performance perspectives
- ❑ Need to rely on hardware and software support in heterogeneous components to access performance
- ❑ Experimental Linux CUDA driver provided by NVIDIA facilitates access to CUDA / GPU performance information
- ❑ TAUcuda merges with TAU (CPU) performance data
- ❑ TAU/TAUcuda provides powerful scalable heterogeneous performance measurement and analysis
- ❑ NVIDIA is incorporating performance tools requirements in next-generation driver/device libraries
- ❑ TAUopencl is in development (working prototype)

Support Acknowledgements



- Staff:
 - Shangkar Mayangalambam (Qualcomm)
 - Alan Morris (U. Utah)
- Department of Energy (DOE)
 - Office of Science, ASC/NNSA
- Department of Defense (DoD), HPCMO
- NSF (SDCI, SI2)
- Research Centre Juelich
- Argonne National Laboratory
- Technical University Dresden
- ParaTools, Inc.
- NVIDIA



ParaTools