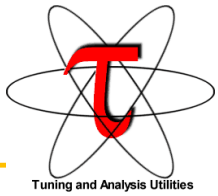
UO Group Overview

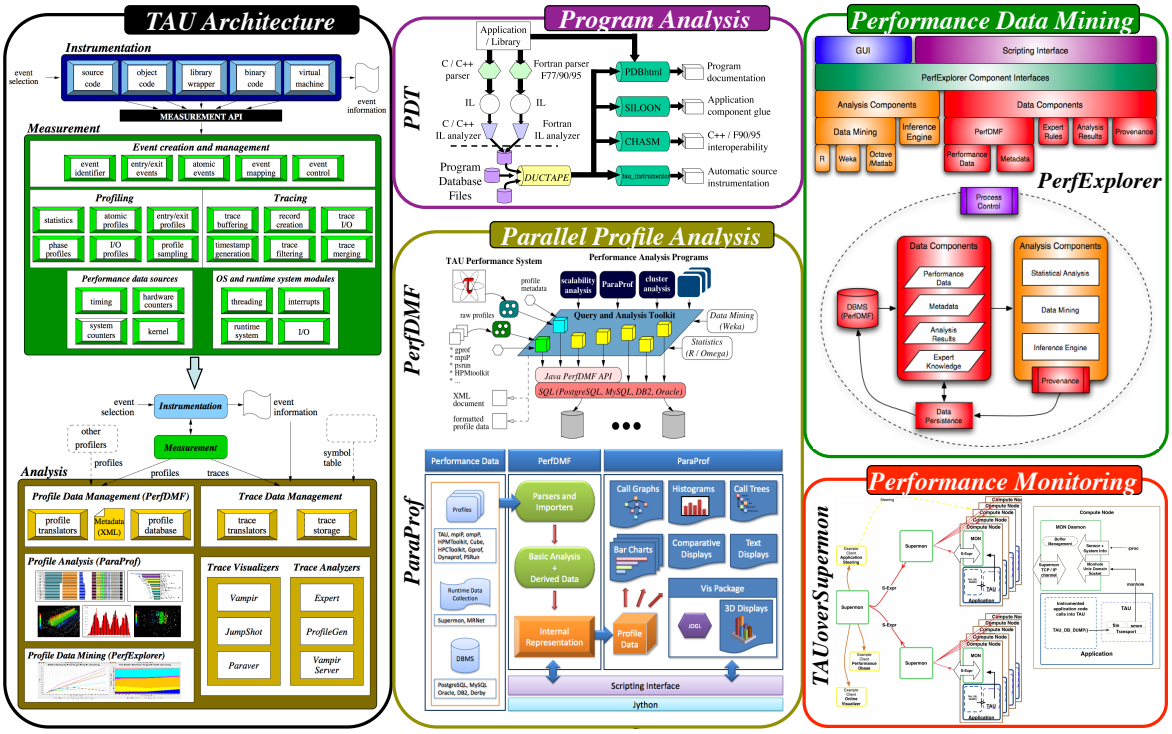
- Personnel
  - Allen D. Malony (co-PI)
  - Sameer Shende, Director, Performance Research Lab
  - Wyatt Spear, Senior software engineer
  - Scott Biersdorff, Software engineer
- Research focus
  - Parallel performance measurement and analysis
  - Integration of performance tools and information in software frameworks
  - Management of multi-experiment performance data
  - Assessment of opportunities and support for optimization
  - Heterogeneous performance analysis/tuning problems

TAU Performance System



- http://tau.uoregon.edu/
- Multi-level performance instrumentation
  - Multi-language automatic source instrumentation
- Flexible and configurable performance measurement
- Widely-ported parallel performance profiling system
  - Computer system architectures and operating systems
  - Different programming languages and compilers
- Support for multiple parallel programming paradigms
  - Multi-threading, message passing, mixed-mode, hybrid
- Integration in complex software, systems, applications

TAU Performance System Components



TAU Instrumentation Approach

- Based on direct performance observation
  - Direct instrumentation of program (system) code (probes)
  - Instrumentation invokes performance measurement
  - Event measurement: performance data, meta-data, context
- Support for standard program events
  - Routines, classes and templates
  - Statement-level blocks and loops
  - Begin/End events (Interval events)
- Support for user-defined events
  - Begin/End events specified by user
  - Atomic events (e.g., size of memory allocated/freed)
  - Flexible selection of event statistics
- Provides static events and dynamic events

Instrumentation: Events in TAU

- Event types
  - Interval events (begin/end events)
    - measures performance between begin and end
    - metrics monotonically increase
  - Atomic events
    - used to capture performance data state
- Code events
  - Routines, classes, templates
  - Statement-level blocks, loops
- User-defined events
  - Specified by the user
- Abstract mapping events

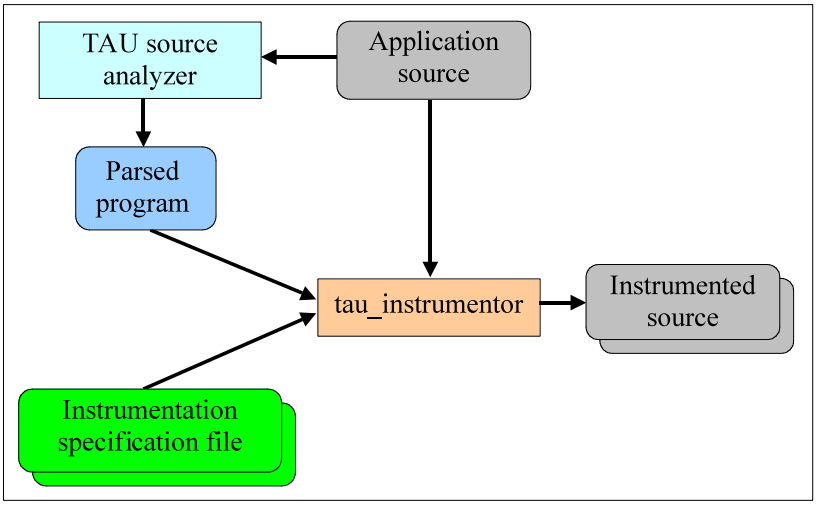
Instrumentation Techniques

- Events defined by instrumentation access
- Instrumentation levels
  - Source code
  - Object code
  - Runtime system
  - Library code
  - Executable code
  - Operating system
- Different levels provide different information
- Different tools needed for each level
- Levels can have different granularity

Instrumentation Techniques

- Static instrumentation
  - Program instrumented prior to execution
- Dynamic instrumentation
  - Program instrumented at runtime
- Manual and automatic mechanisms
- Tool required for automatic support
  - Source time: preprocessor, translator, compiler
  - Link time: wrapper library, preload
  - Execution time: binary rewrite, dynamic
- Advantages / disadvantages

Automatic Source-Level Instrumentation in TAU using Program Database Toolkit (PDT)



Selective Instrumentation File

- Specify a list of routines to exclude or include (case sensitive)
- # is a wildcard in a routine name. It cannot appear in the first column.

```
BEGIN_EXCLUDE_LIST
Foo
Bar
D#EMM
END_EXCLUDE_LIST
```

- Specify a list of routines to include for instrumentation

```
BEGIN_INCLUDE_LIST
int main(int, char **)
F1
F3
END_INCLUDE_LIST
```

- Specify either an include list or an exclude list!

Selective Instrumentation File

- Optionally specify a list of files to exclude or include (case sensitive)
- * and ? may be used as wildcard characters in a file name

```
BEGIN_FILE_EXCLUDE_LIST
f*.f90
Foo?.cpp
END_FILE_EXCLUDE_LIST
```

- Specify a list of routines to include for instrumentation

```
BEGIN_FILE_INCLUDE_LIST
main.cpp
foo.f90
END_FILE_INCLUDE_LIST
```

Selective Instrumentation File

- User instrumentation commands are placed in INSTRUMENT section
- ? and * used as wildcard characters for file name, # for routine name
- \ as escape character for quotes
- Routine entry/exit, arbitrary code insertion
- Outer-loop level instrumentation

```
BEGIN_INSTRUMENT_SECTION
loops file="foo.f90" routine="matrix#"
memory file="foo.f90" routine="#"
io routine="matrix#"
[static/dynamic] phase routine="MULTIPLY"
dynamic [phase/timer] name="foo" file="foo.cpp" line=22 to line=35
file="foo.f90" line = 123 code = "  print *, \" Inside foo\""
exit routine = "int foo()" code = "cout <<\"exiting foo\"<<endl;"
END_INSTRUMENT_SECTION
```


Instrumentation Specification

```
% tau_instrumentor
Usage : tau_instrumentor <pdbfile> <sourcefile> [-o <outputfile>] [-noinline]
[-g groupname] [-i headerfile] [-c|-c++|-fortran] [-f <instr_req_file> ]
For selective instrumentation, use -f option
% tau_instrumentor foo.pdb foo.cpp -o foo.inst.cpp -f selective.dat
% cat selective.dat
# Selective instrumentation: Specify an exclude/include list of routines/files.
BEGIN_EXCLUDE_LIST
void quicksort(int *, int, int)
void sort_5elements(int *)
void interchange(int *, int *)
END_EXCLUDE_LIST

BEGIN_FILE_INCLUDE_LIST
Main.cpp
Foo?.c
*.C
END_FILE_INCLUDE_LIST
# Instruments routines in Main.cpp, Foo?.c and *.C files only
# Use BEGIN_[FILE]_INCLUDE_LIST with END_[FILE]_INCLUDE_LIST
```

Instrumenting a C code

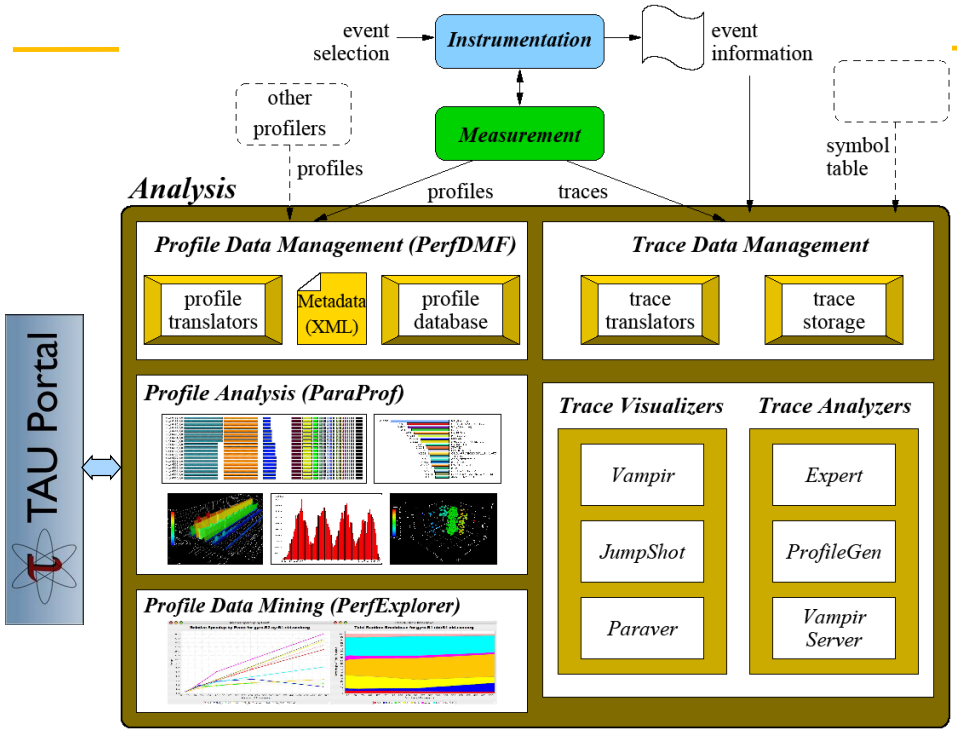
```
#include <TAU.h>
int foo(int x) {
  TAU_START("foo");
   for (i = 0; i < x; i++) { // do work
   }
  TAU_STOP("foo");
}

int main(int argc, char **argv) {
  TAU_INIT(&argc, &argv);
  TAU_START("main");
  TAU_PROFILE_SET_NODE(rank);
  …
  TAU_STOP("main");
}
% gcc -I<taudir>/include foo.c -o foo -L<taudir>/<arch>/lib -lTAU
% ./a.out
% pprof; paraprof
NOTE: Replace TAU_START("foo") with call TAU_START('foo')
       in Fortran. See <taudir>/include/TAU.h for full API.
```

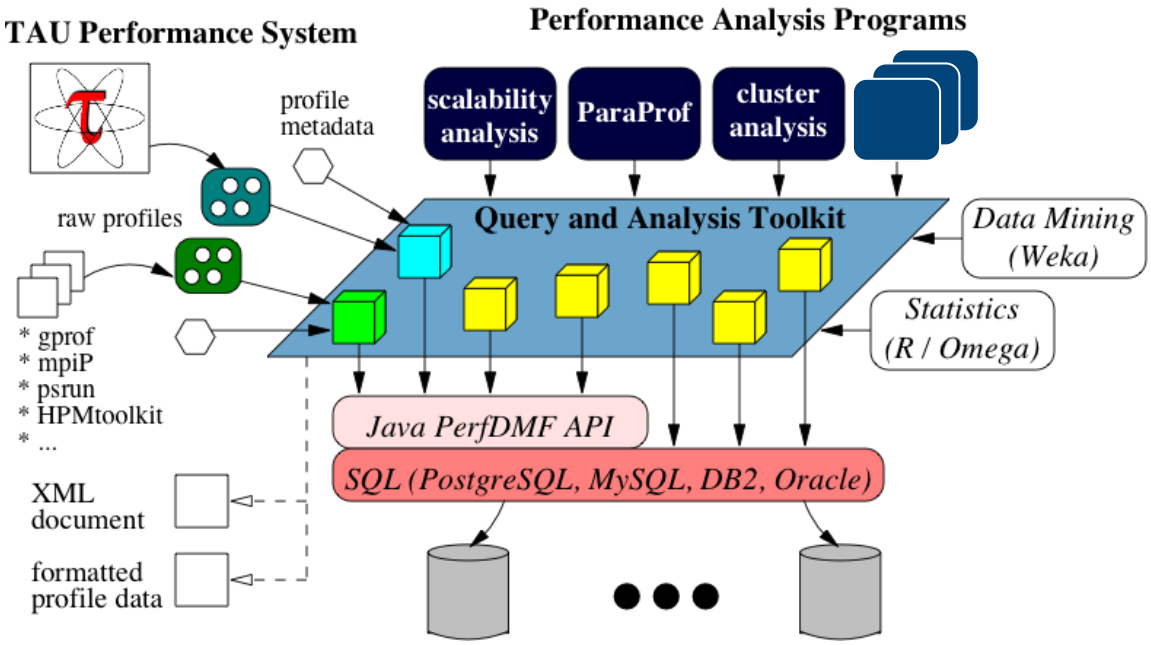

Environment Variables in TAU

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 separted list generates other metrics. (e.g., TIME:linuxtimers:PAPI_FP_OPS:PAPI_NATIVE_<event>)

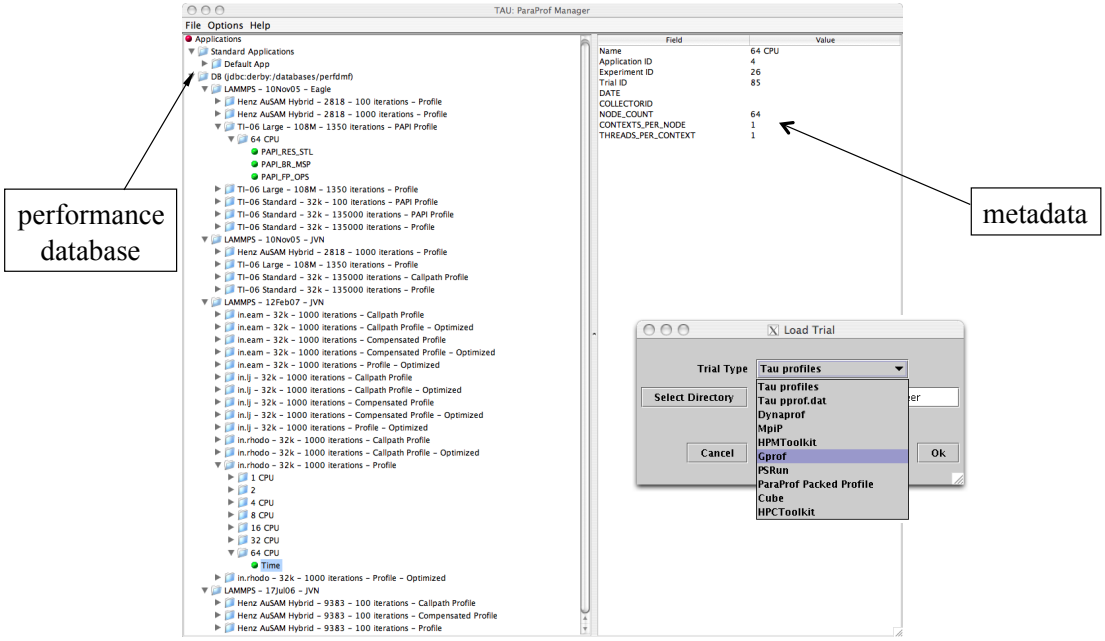
TAU Performance System Architecture



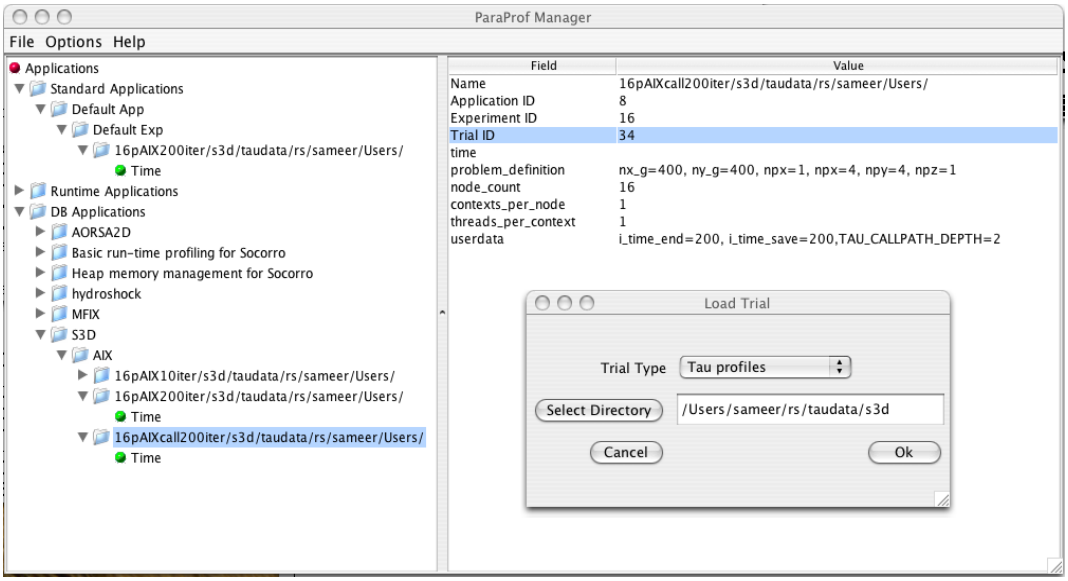
PerfDMF: Performance Data Mgmt. Framework



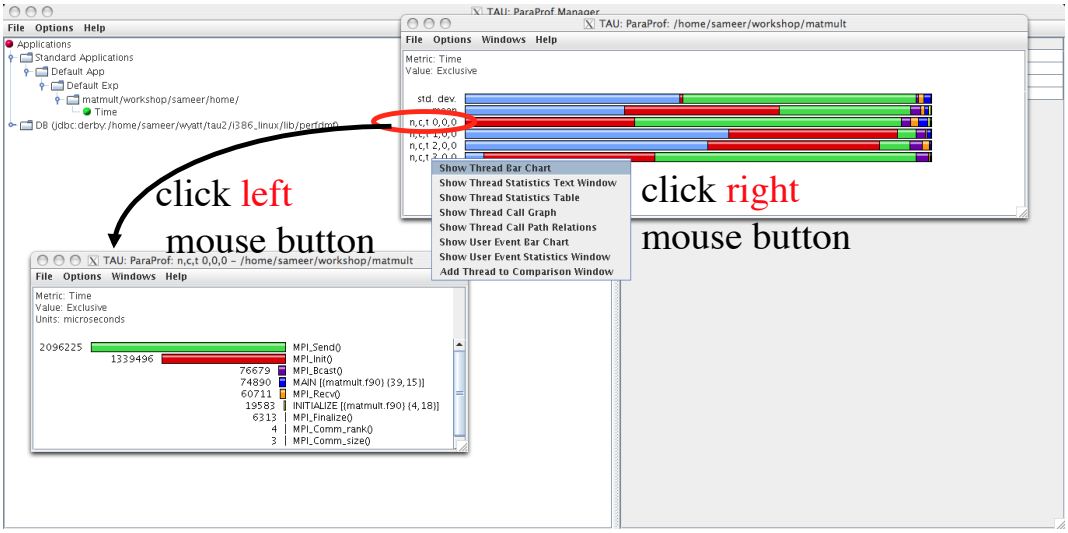
ParaProf – Manager Window



Performance Database: Storage of MetaData

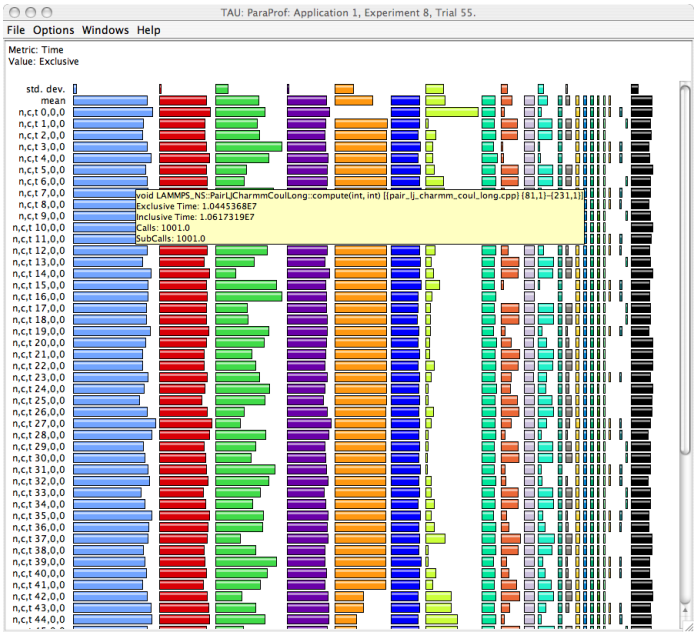


ParaProf Main Window

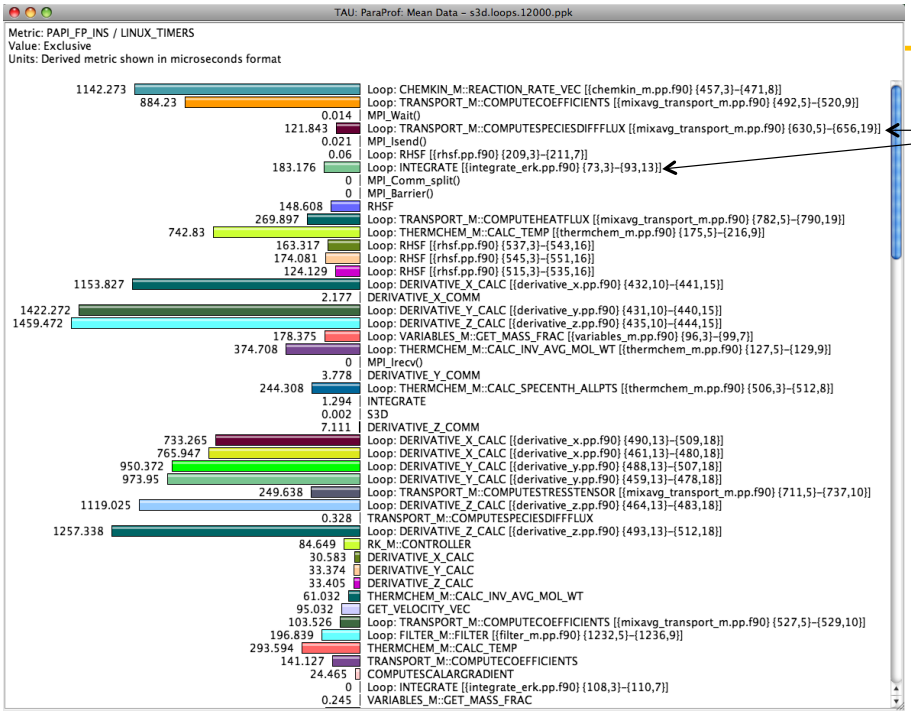



```
% paraprof matmult.ppk
```

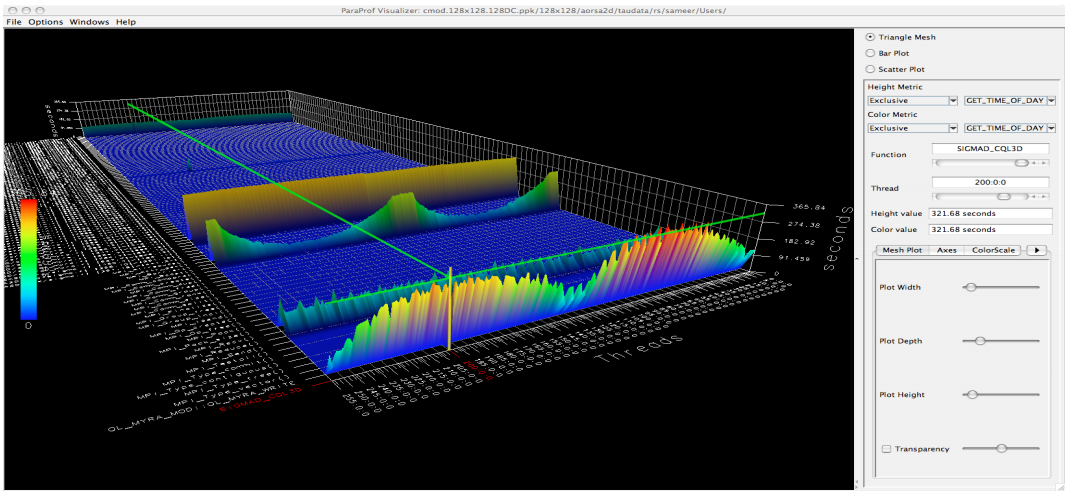

ParaProf Main Window (Lammps)



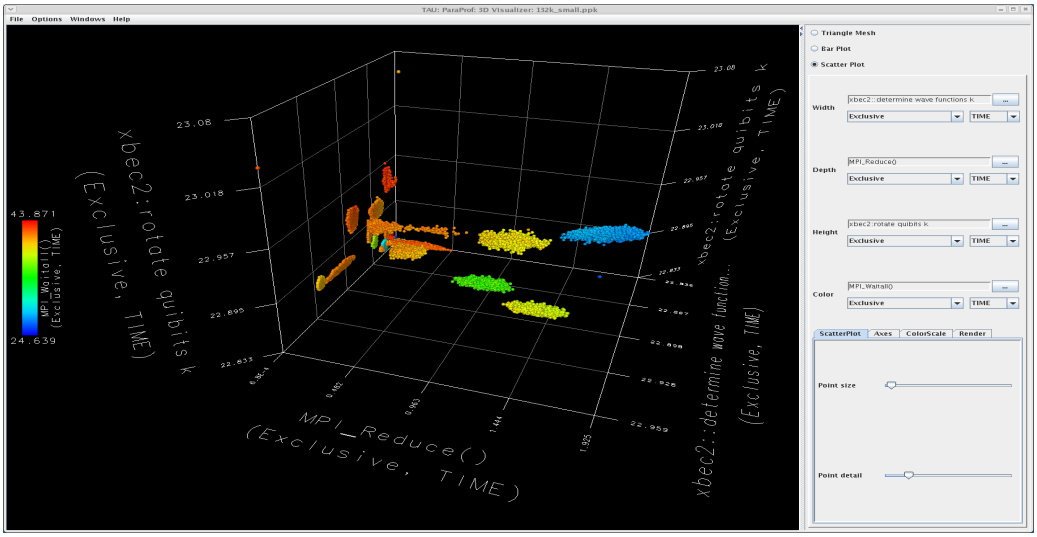
ParaProf: Mflops Sorted by Exclusive Time



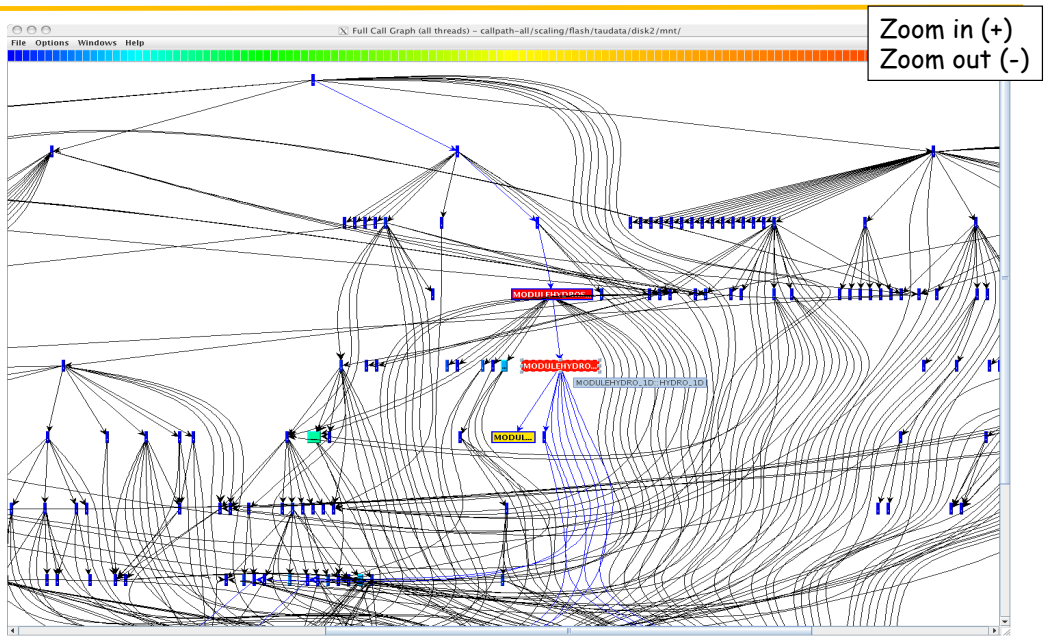
Parallel Profile Visualization: ParaProf



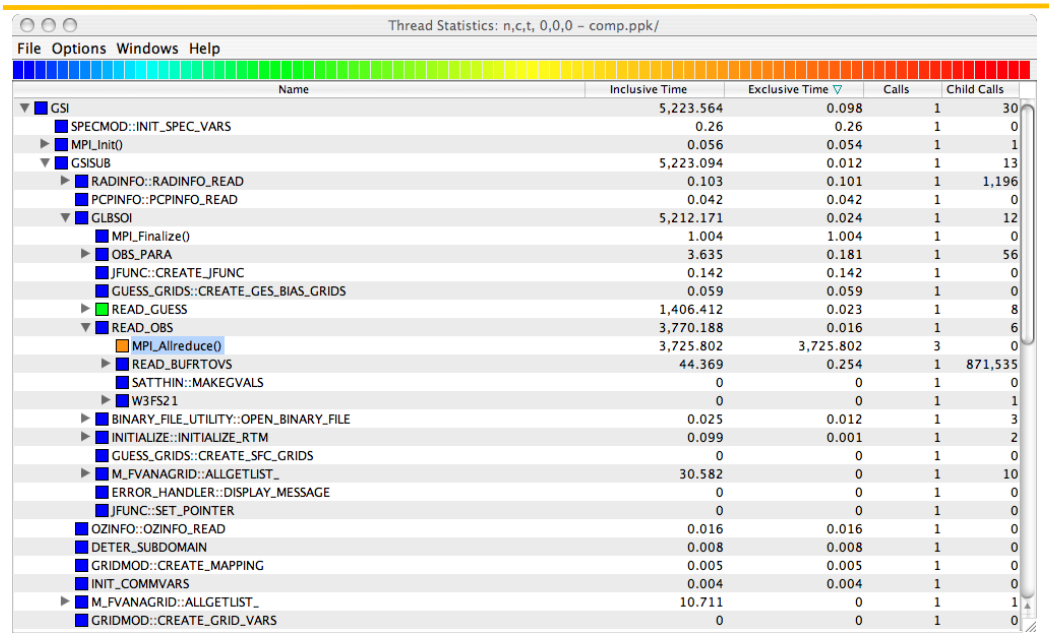
Scatter Plot: ParaProf (128k cores)



ParaProf – Callgraph Zoomed (Flash)

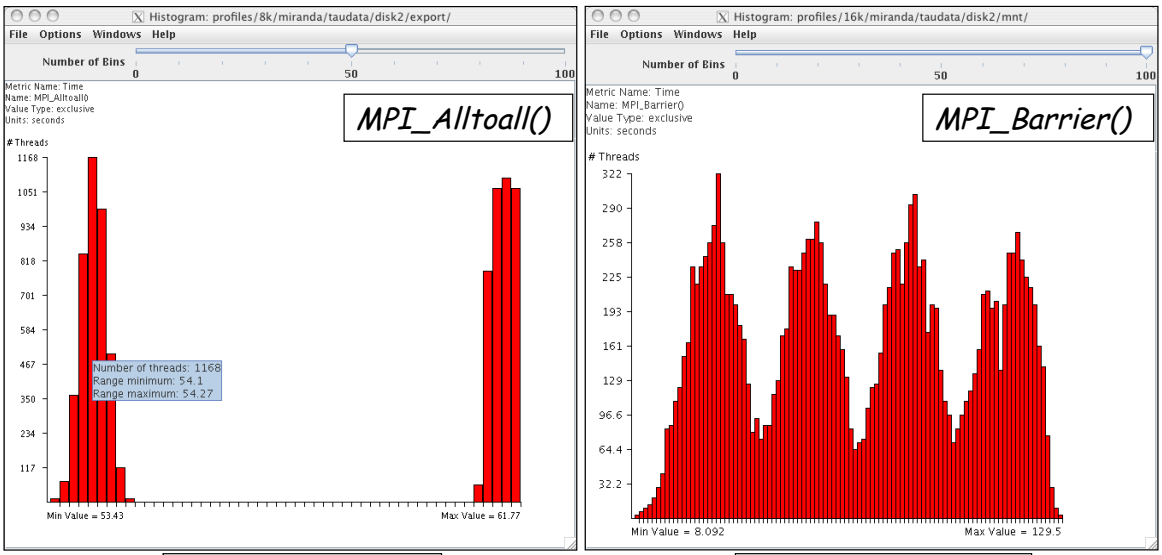


ParaProf - Thread Statistics Table (GSI)



Name	Inclusive Time	Exclusive Time ▽	Calls	Child Calls
GSI	5,223.564	0.098	1	30
SPECMOD::INIT_SPEC_VARS	0.26	0.26	1	0
MPI_Init()	0.056	0.054	1	1
GSISUB	5,223.094	0.012	1	13
RADINFO::RADINFO_READ	0.103	0.101	1	1,196
PCPINFO::PCPINFO_READ	0.042	0.042	1	0
GLBSOI	5,212.171	0.024	1	12
MPI_Finalize()	1.004	1.004	1	0
OBS_PARA	3.635	0.181	1	56
JFUNC::CREATE_JFUNC	0.142	0.142	1	0
GUESS_GRIDS::CREATE_GES_BIAS_GRIDS	0.059	0.059	1	0
READ_GUESS	1,406.412	0.023	1	8
READ_OBS	3,770.188	0.016	1	6
MPI_Allreduce()	3,725.802	3,725.802	3	0
READ_BUFRTOVS	44.369	0.254	1	871,535
SATTHIN::MAKEGVALS	0	0	1	0
W3FS21	0	0	1	1
BINARY_FILE_UTILITY::OPEN_BINARY_FILE	0.025	0.012	1	3
INITIALIZE::INITIALIZE_RTM	0.099	0.001	1	2
GUESS_GRIDS::CREATE_SFC_GRIDS	0	0	1	0
M_FVANAGRID::ALLGETLIST_	30.582	0	1	10
ERROR_HANDLER::DISPLAY_MESSAGE	0	0	1	0
JFUNC::SET_POINTER	0	0	1	0
OZINFO::OZINFO_READ	0.016	0.016	1	0
DETER_SUBDOMAIN	0.008	0.008	1	0
GRIDMOD::CREATE_MAPPING	0.005	0.005	1	0
INIT_COMMVARS	0.004	0.004	1	0
M_FVANAGRID::ALLGETLIST_	10.711	0	1	1
GRIDMOD::CREATE_GRID_VARS	0	0	1	0

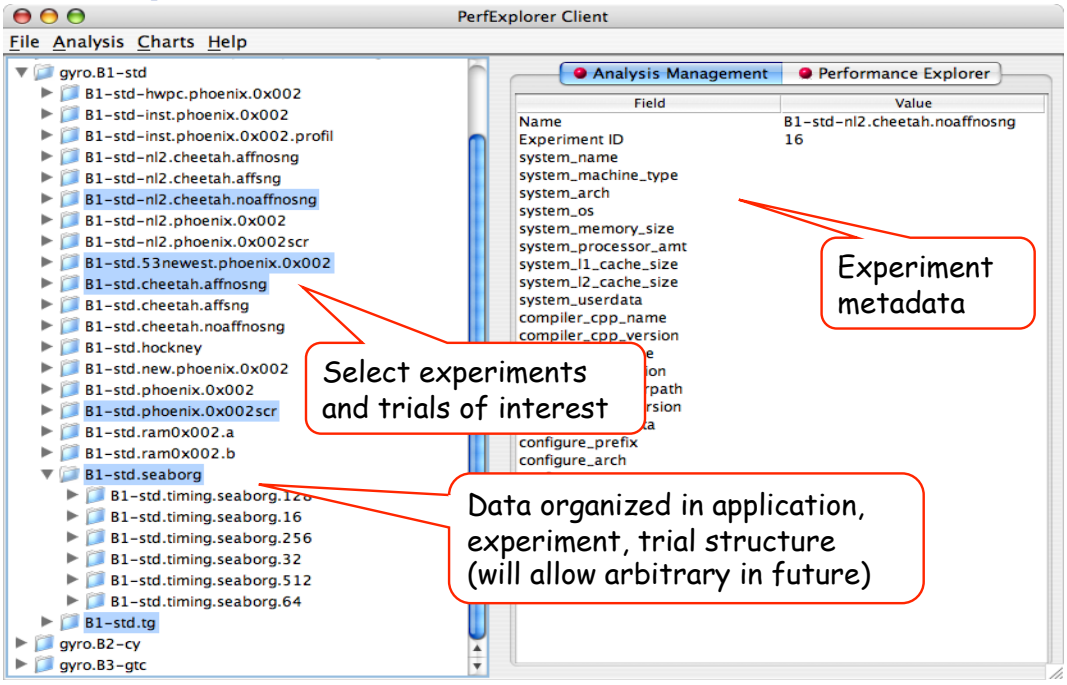
ParaProf – Histogram View (Miranda)



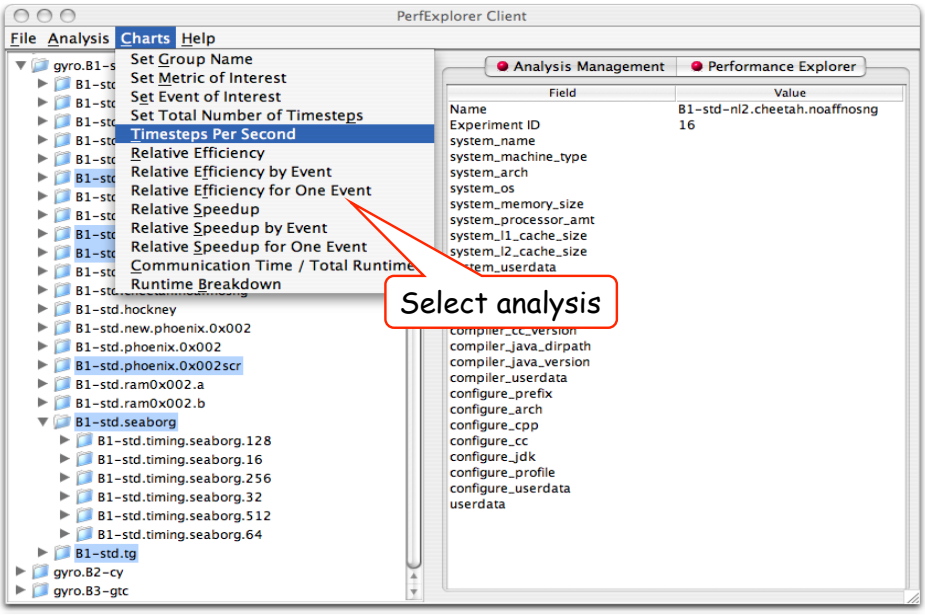
8k processors

16k processors

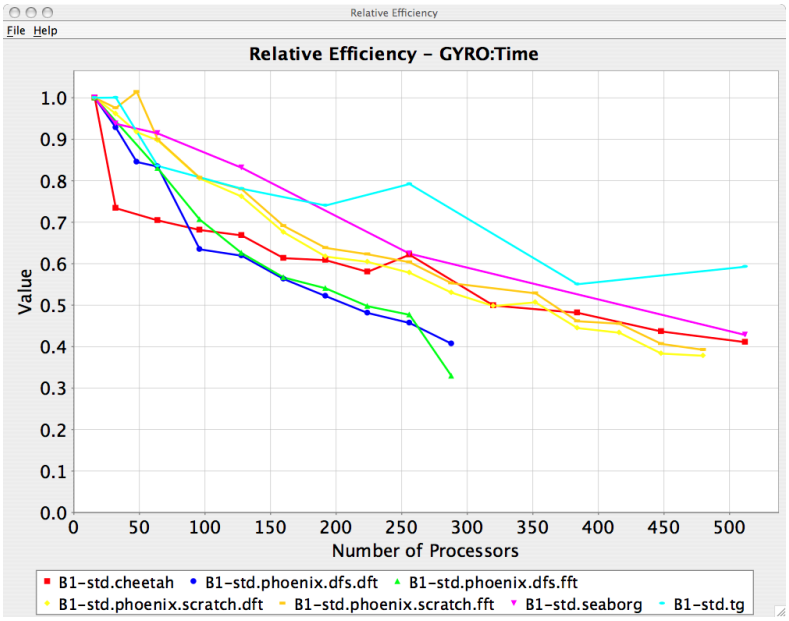
PerfExplorer - Interface



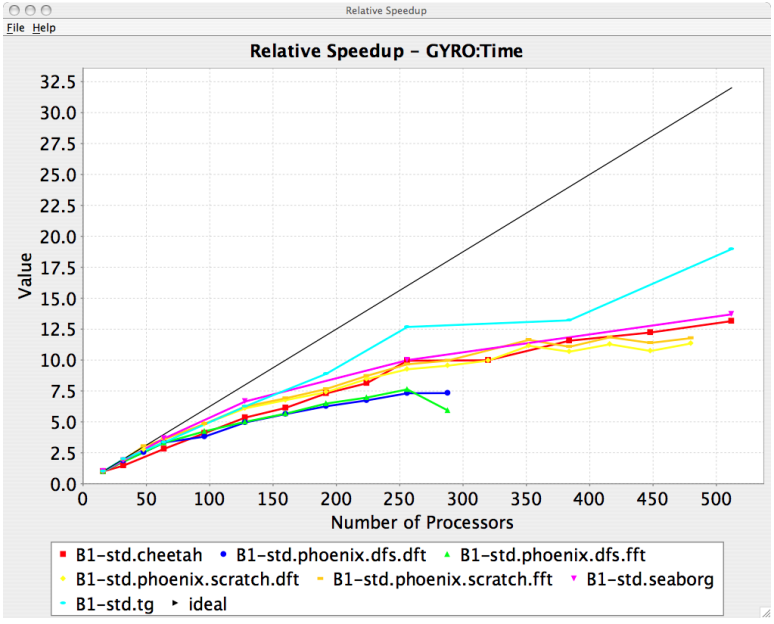
PerfExplorer - Interface



PerfExplorer - Relative Efficiency Plots

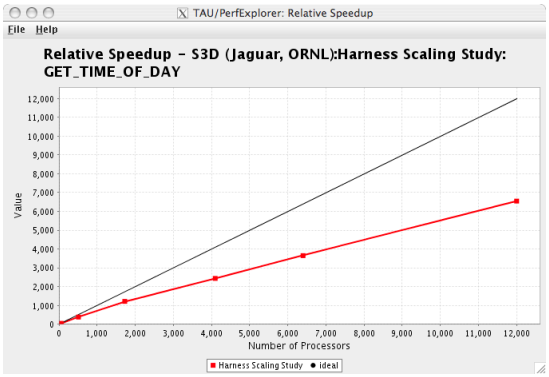


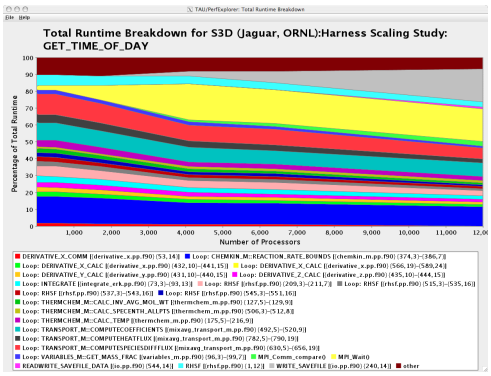
PerfExplorer - Relative Speedup



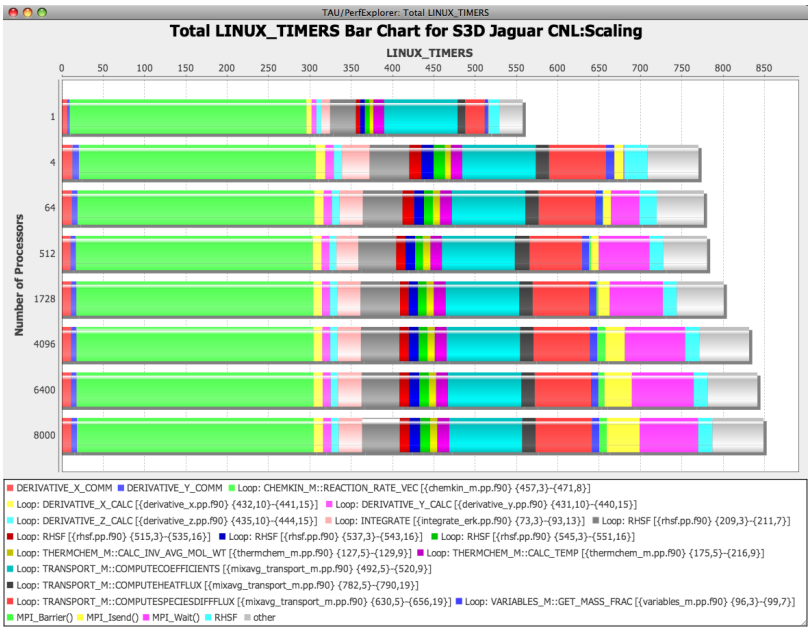
Usage Scenarios: Evaluate Scalability

- Goal: How does my application scale? What bottlenecks occur at what core counts?
- Load profiles in PerfDMF database and examine with PerfExplorer





Usage Scenarios: Evaluate Scalability

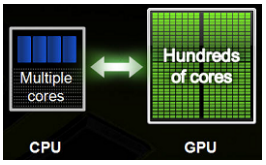


Heterogeneous Performance Views

- 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



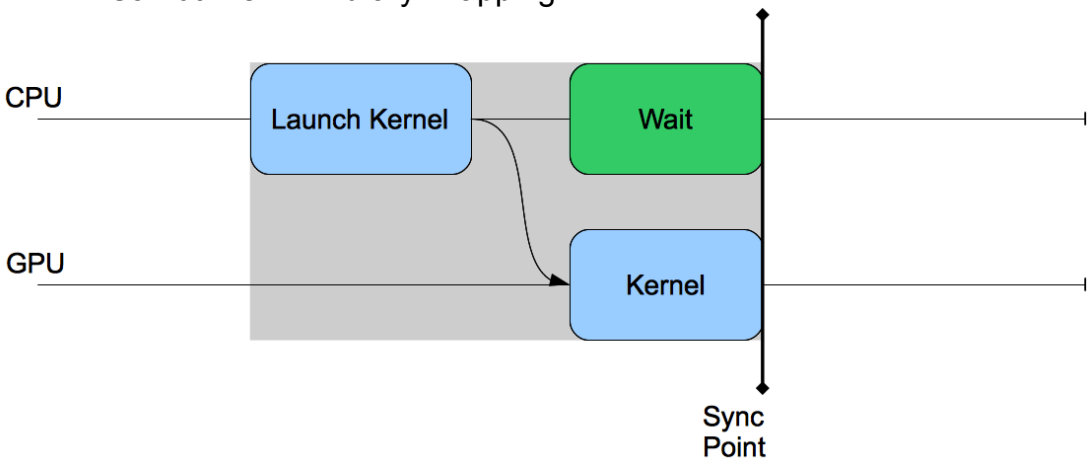
- Host creates measurement perspective for external task
  - Maintains local and remote performance data
  - Tasks may have limited measurement support
  - May depend on host for performance data I/O
  - Performance data might be received from external task
- How to create a view of heterogeneous external performance?

GPU Performance Observation Methods

- GPU API
  - GPU programming libraries expose opportunities for observing performance events
  - We can wrap library to capture CPU events
  - Preload wrapped library with program code
- GPU kernel (GPU event) measurement is more difficult
  - Problem is how to observe GPU clock and counters
  - May need to infer GPU events from their side effects on CPU
- Performance measurement techniques (logical)
  - Synchronous performance measurement
  - Event queue performance measurement
  - Callbacks

Synchronous GPU Performance Measurement

- This method uses only CPU events (available everywhere)
  - Does not depend on anything in CUDA or OpenCL
  - Can utilize API library wrapping



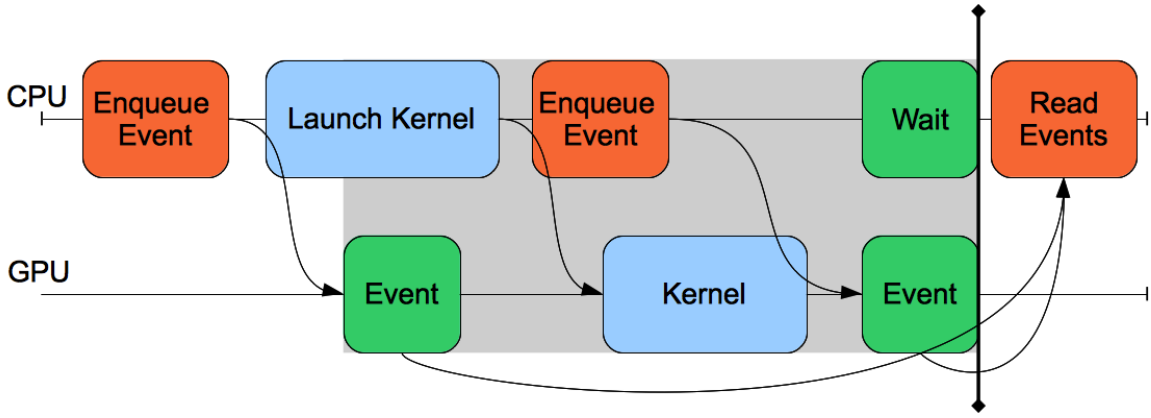
Wrapping Runtime Libraries

- CUDA
  - cudaMalloc
  - cudaFree
  - cudaMemcpy
  - cudaLaunch
  - cudaGetDeviceCount
  - cudaSetDevice
  - cudaDestroy
  - ...

- OpenCL
  - clBuildProgram
  - clCreateContext
  - clReleaseContext
  - clGetDeviceInfo
  - clEnqueueCopyBuffer
  - clEnqueueNDRangeKernel
  - clCreateKernel
  - ...

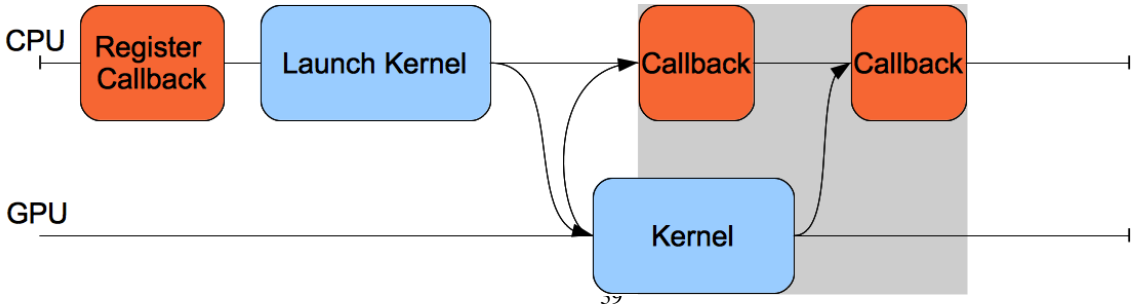
Event Queue Performance Measurement

- This method utilizes support in CUDA and OpenCL
  - CUDA: `cuEvent` interface
    - GPU device timestamps and CUDA device driver manages
  - OpenCL: `clGetProfilingInfo` interface
    - events are all on the CPU
    - not timestamped on GPU device (depends on implementation)

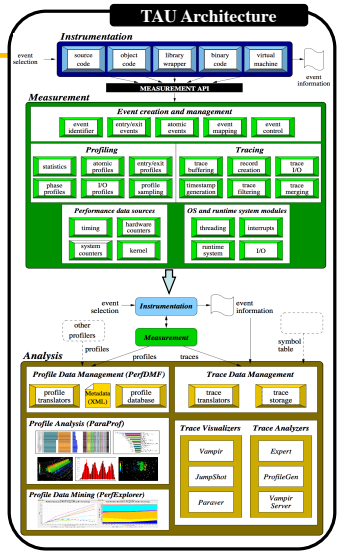


Callback Profiling (CUPTI)

- Callbacks are provided to capture events
  - Runtime and GPU
- CUPTI (stay tuned)
  - CUDA runtime
  - GPU events (hopefully)
- OpenCL specification 1.1 (NVIDIA development version)



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

TAU and TAUcuda Performance Events

- TAU measures events during execution
  - Events are made visible as a result of code instrumentation
  - Records event *begin* and *end* for profiling and tracing
  - *TAU events* are measured by the CPU when they happen
- TAU can not measure events on the GPU
  - *TAUcuda events* are measured by CUDA and the GPU device
  - TAUcuda events occur asynchronously to TAU events
- TAUcuda is integrated with TAU measurement infrastructure
  - Must transform TAUcuda events into TAU events
  - Associate TAUcuda events with application CPU operation
    - samples the TAU context to link to application call site

TAUcuda Performance Measurement (Version 1)

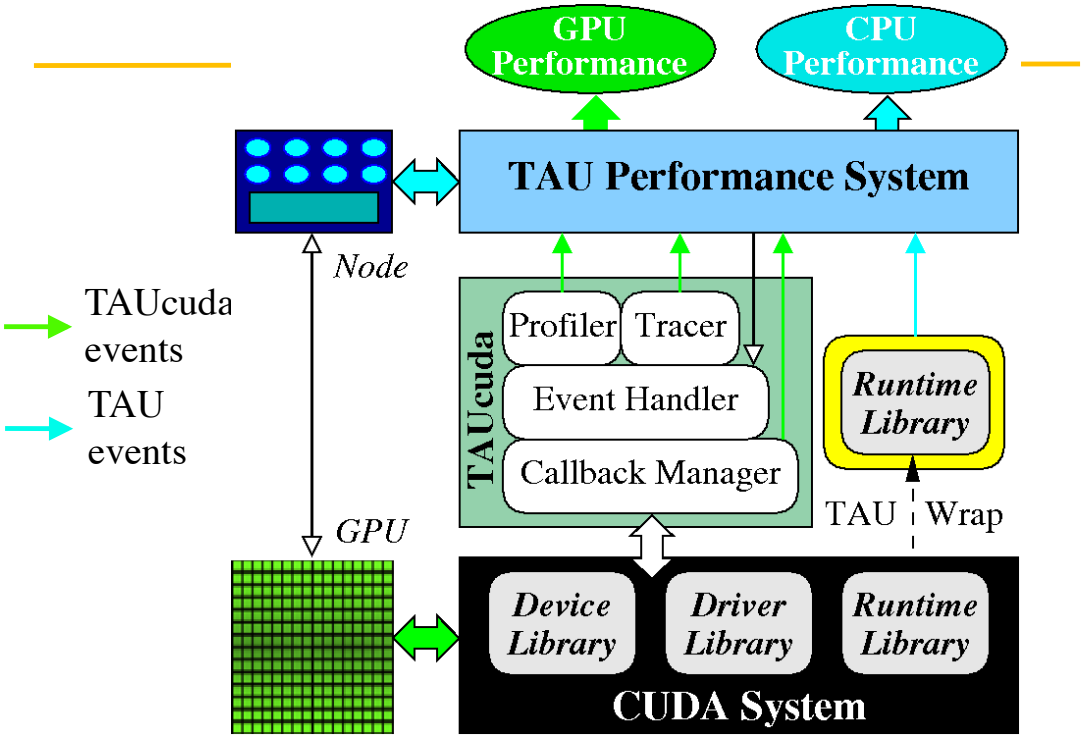
- Build on CUDA event interface
  - Allow “events” to be placed in streams and processed
    - events are timestamped by CUDA driver
  - CUDA driver reports GPU timing in event structure
  - Events are reported back to CPU when requested
    - use begin and end events to calculate intervals
- CUDA kernel invocations are asynchronous
  - CPU does not see actual CUDA “end” event
- Want to associate TAU event context with CUDA events
  - Get top of TAU event stack at begin (*TAU context*)

S. Mayanglambam, A. Malony, M. Sottile, "Performance Measurement of Applications with GPU Acceleration using CUDA," *ParCo 2009*, Lyon, France, September 2009.

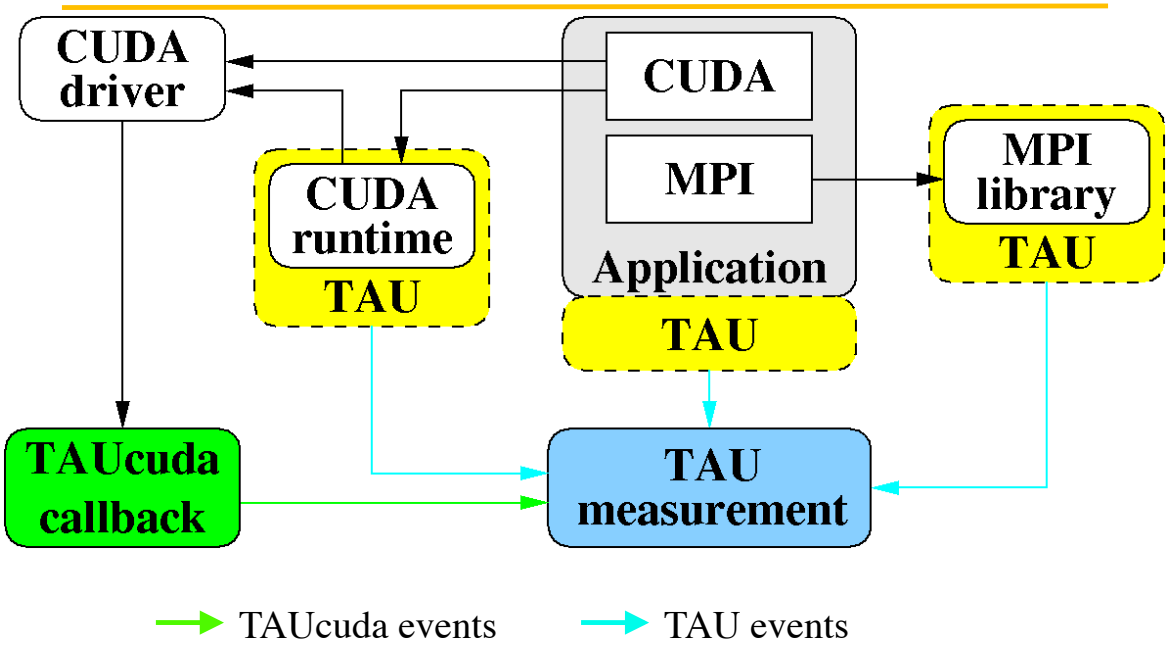
TAUcuda Performance Measurement (Version 2)

- Overcome TAUcuda (v1) deficiencies
  - Required source code instrumentation (not entirely true)
  - Event interface only perspectives
    - could not see memory transfer or CUDA system execution
- TAUcuda (v2) built on *experimental* Linux CUDA driver
  - Linux CUDA driver R190.86 supports a callback interface!!!
  - Drive and device libraries and events
- Utilized TAU library wrapping technology
  - Wrapped CUDA runtime (*cudaYYY*) library

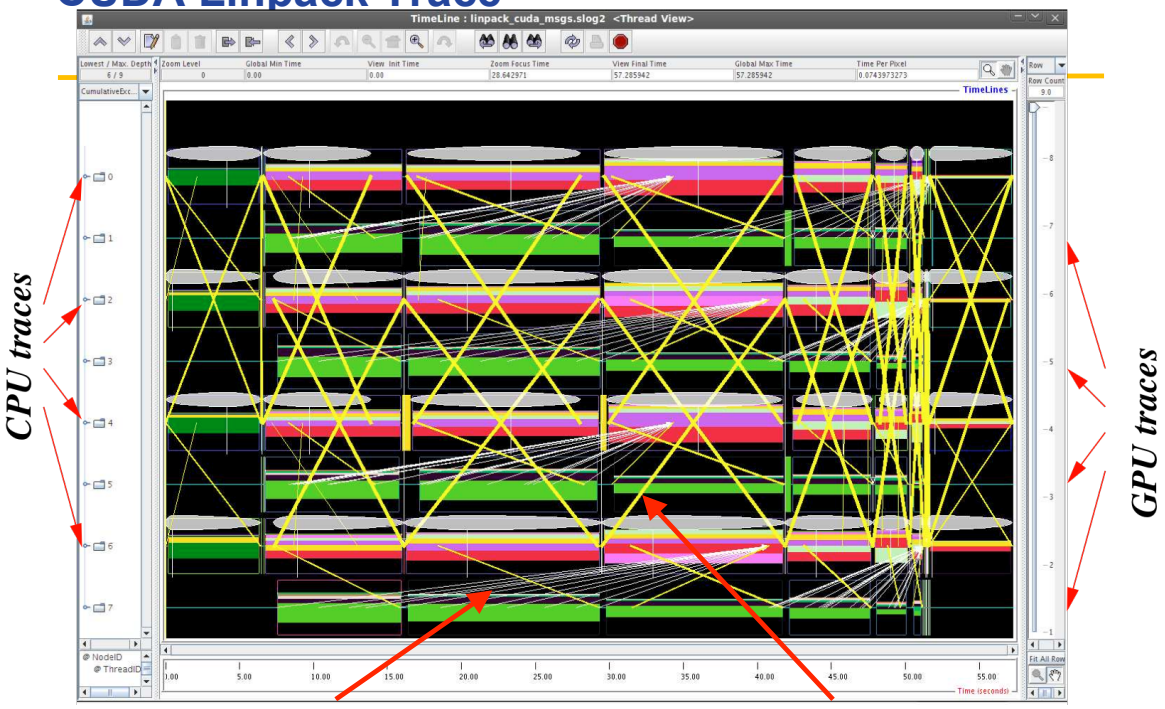
TAUcuda Architecture



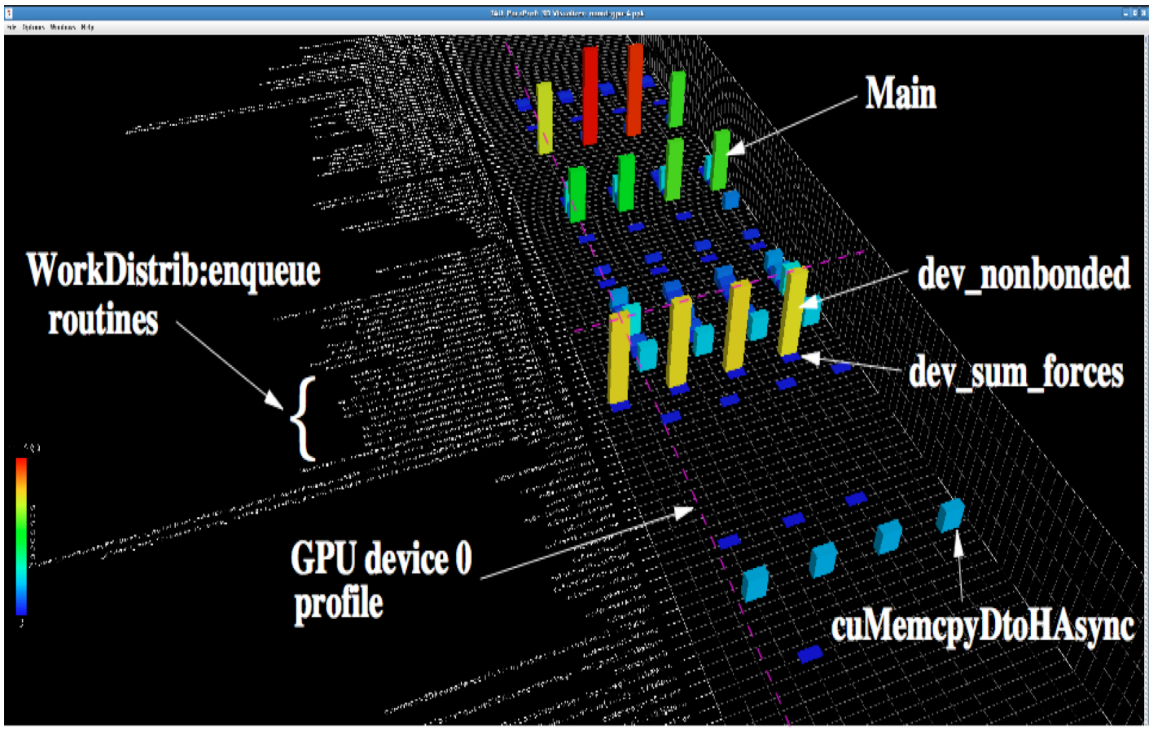
TAUcuda Instrumentation



CUDA Linpack Trace

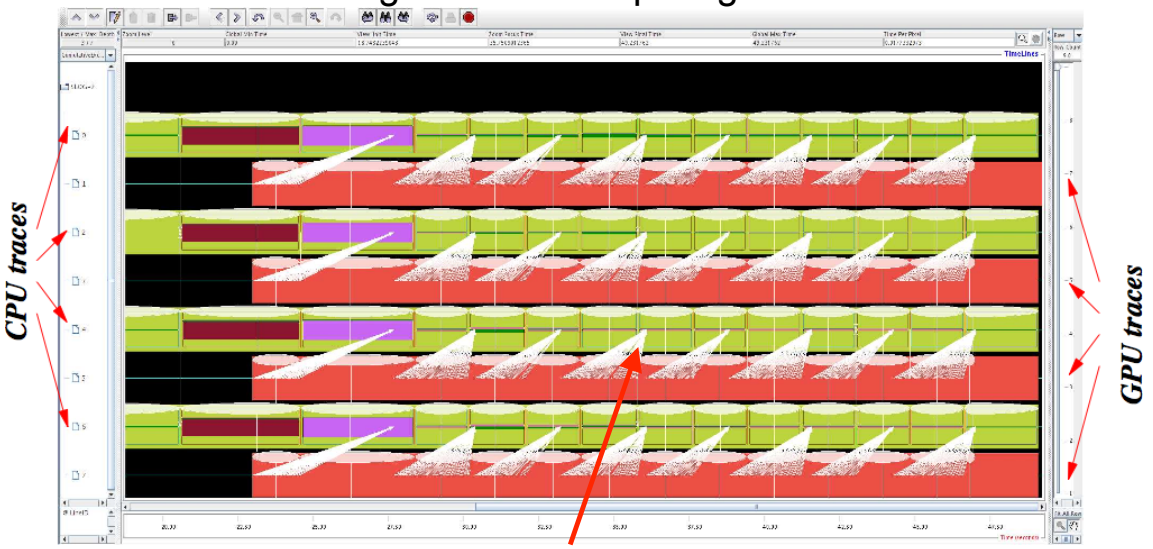


NAMD Profile (4 processes, 4 GPUs)



SHOC Stencil2D (512 iterations, 4 CPUxGPU)

- Scalable HeterOgenerous Computing benchmark suite



CUDA memory transfer (white)

Current Activities

- Experience with experimental driver was very positive
  - Library wrapping plus callbacks is a good strategy
- NVIDIA CUPTI started to provide production driver release
- TAU version 2.20 (released in November at SC10)
  - API runtime library wrapping for CUDA and OpenCL
  - CUPTI callbacks for runtime libraries
  - Kernel event callsback in OpenCL 1.1 (with NVIDIA driver)
- In development
  - Event queue support for CUDA and OpenCL
  - Interfacing with PAPI for improved CUPTI counter access
- Need to have a broader strategy

CUPTI (CUDA Performance Tools Interface)

- Currently available only to select developers
  - University of Oregon
  - University of Tennessee, Knoxville
  - Rice University
  - TU Dresden
- Based on callback model for observing GPU performance
  - Includes callbacks for CUDA's runtime and driver API
  - Current no support for kernel callbacks
- Supports GPU counter access
  - Method for retriving GPU counters:
    - instructions, branches, load and stores

TAU with CUPTI

- TAU works with CUPTI library callbacks
  - Equivalent to wrapping CUDA library
  - runtime and driver libraries
- TAU collects some GPU counters with CUPTI
- PAPI component for CUPTI provides improved support
- Will switch TAU to work directly with PAPI

OpenCL Callback

- OpenCL specification 1.1 includes kernel event callback
  - *clSetEventCallback*
- Register callbacks for the following events:
  - CL_QUEUED
  - CL_SUBMITTED
  - CL_RUNNING
  - CL_COMPLETE
- Creating a CL_COMPLETE callback for a kernel event should allow us to track the performance of this kernel after it has been executed on the GPU
- NVIDIA provided UO a beta driver supporting OpenCL callbacks

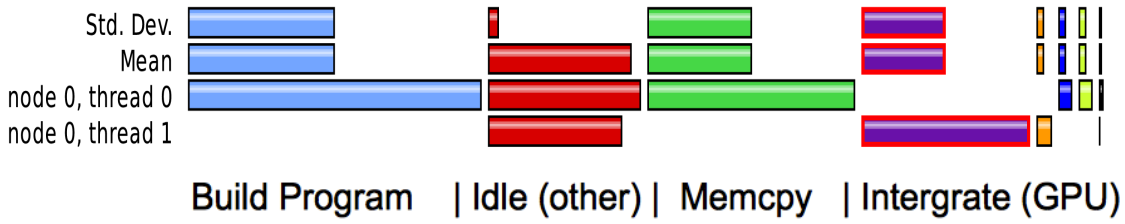
CUDA Profiling with Library Wrapping

Name Δ	Exclusive TIME	Inclusive TIME	Calls	Child Calls
.TAU application	1,049.457	1,403.923	1	22
cudaError_t cudaBindTexture(size_t *, const struct textureReference *, const void *, co	0.012	0.012	2	0
cudaError_t cudaConfigureCall(dim3, dim3, size_t, cudaStream_t) C	0.002	0.002	1	0
cudaError_t cudaFree(void *) C	349.925	349.925	4	0
cudaError_t cudaGetDevice(int *) C	0.001	0.001	1	0
cudaError_t cudaGetDeviceProperties(struct cudaDeviceProp *, int) C	0.051	0.051	1	0
cudaError_t cudaGetLastError(void) C	0.001	0.001	2	0
cudaError_t cudaLaunch(const char *) C	0.056	0.056	1	0
cudaError_t cudaMalloc(void **, size_t) C	0.14	0.14	3	0
cudaError_t cudaMemcpy(void *, const void *, size_t, enum cudaMemcpyKind) C	4.273	4.273	4	0
cudaError_t cudaSetupArgument(const void *, size_t, size_t) C	0.002	0.002	1	0
cudaError_t cudaUnbindTexture(const struct textureReference *) C	0.003	0.003	2	0

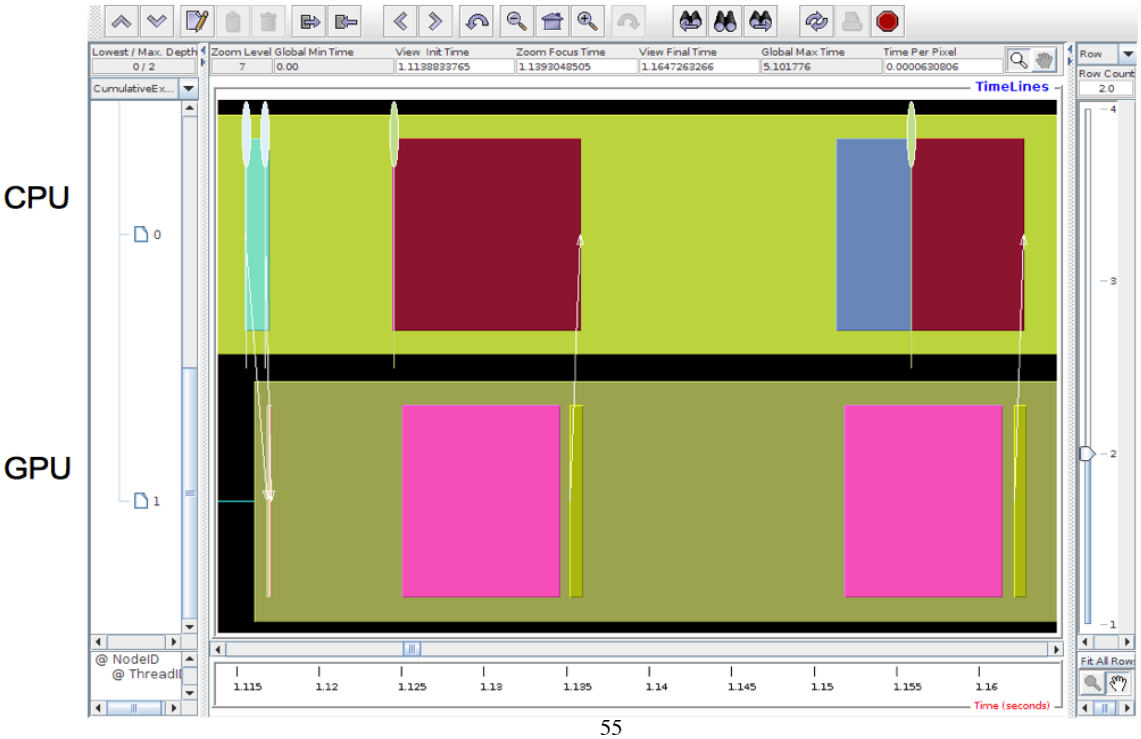
Name Δ	Total	NumSamp...	MaxValue	MinValue	MeanValue	Std. Dev.
.TAU application						
cudaError_t cudaMemcpy(void *, const void *, size_t, enum cudaMen						
Bytes copied from Host to Device	907,500	3	302,500	302,500	302,500	0
Bytes copied from Device to Host	302,500	1	302,500	302,500	302,500	0
Bytes copied from Device to Host	302,500	1	302,500	302,500	302,500	0
Bytes copied from Host to Device	907,500	3	302,500	302,500	302,500	0

OpenCL Profiling with Kernel Callbacks

Metric: TAUGPU_TIME
Value: Exclusive

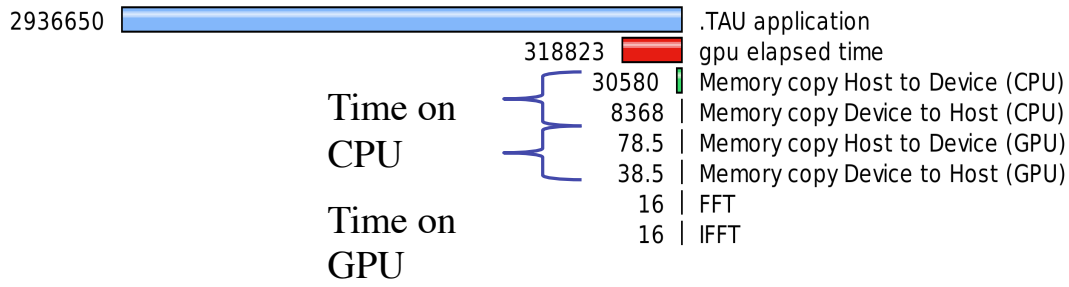


OpenCL Library Wrapping with Kernel



SHOC OpenCL FFT Program Profile

Metric: TIME
Value: Exclusive
Units: microseconds



Sorted By: Number of Samples

NumSamples	Max	Min	Mean	Std. Dev	Name
3	4	4	4	0	Memory copied from Device to Host
2	1.0486E8	4	5.2429E7	5.2429E7	Memory copied from Host to Device

CUPTI Runtime API Callback

Name Δ	Exclusive TIME	Inclusive TIME	Calls	Child Calls
.TAU application	1,850.497	1,998.823	1	238
cudaConfigureCall (RuntimeAPI)	0.179	0.179	62	31
cudaFree (RuntimeAPI)	0.175	0.175	6	3
cudaGetDevice (RuntimeAPI)	0.005	0.005	2	1
cudaGetDeviceCount (RuntimeAPI)	0.035	0.035	2	1
cudaGetDeviceProperties (RuntimeAPI)	0.181	0.181	10	5
cudaGetLastError (RuntimeAPI)	0.006	0.006	2	1
cudaLaunch (RuntimeAPI)	0.363	0.363	62	31
cudaMalloc (RuntimeAPI)	90.369	90.369	6	3
cudaMemcpy (RuntimeAPI)	0.34	0.34	6	3
cudaSetDevice (RuntimeAPI)	0.016	0.016	2	1
cudaSetupArgument (RuntimeAPI)	0.902	0.902	310	155
cudaThreadExit (RuntimeAPI)	55.721	55.721	2	1
cudaThreadSynchronize (RuntimeAPI)	0.034	0.034	4	2

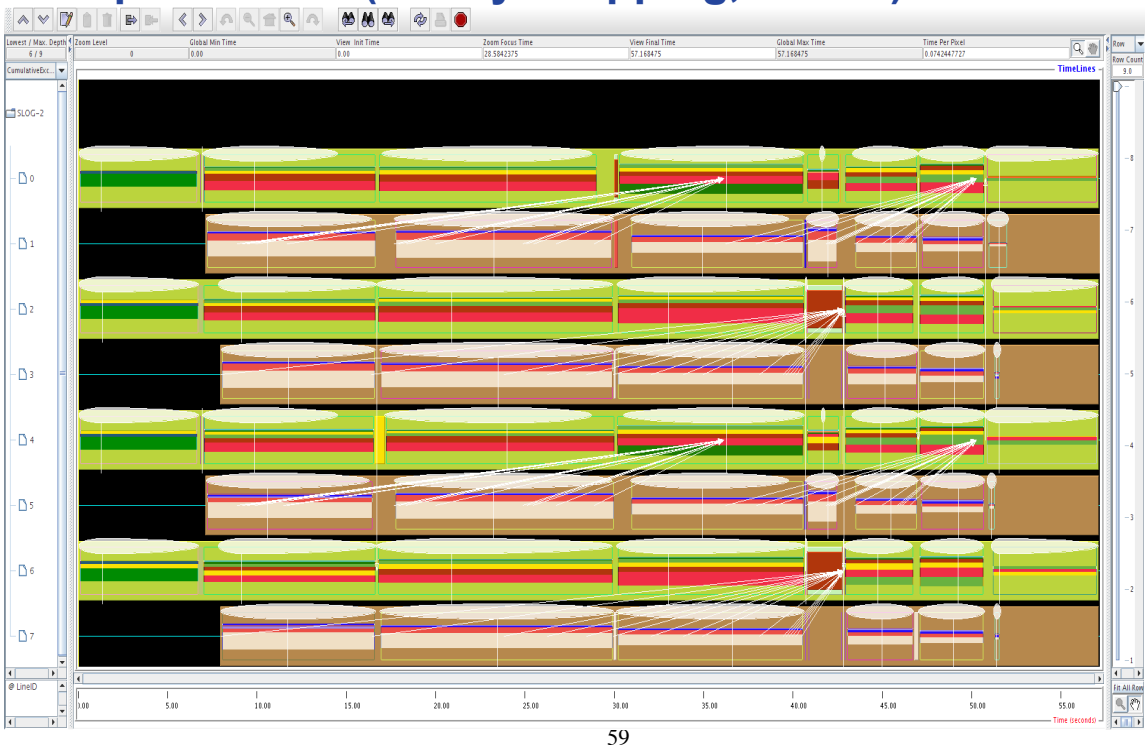
Name Δ	Total	NumSamp...	MaxValue	MinValue	MeanValue	Std. Dev.
.TAU application						
cudaMemcpy (RuntimeAPI)						
Bytes copied	128,000	3	51,200	25,600	42,666.667	12,067.956
Bytes copied	256,000	6	51,200	25,600	42,666.667	12,067.956
cudaMemcpy (RuntimeAPI)						

CUPTI Driver API Callback

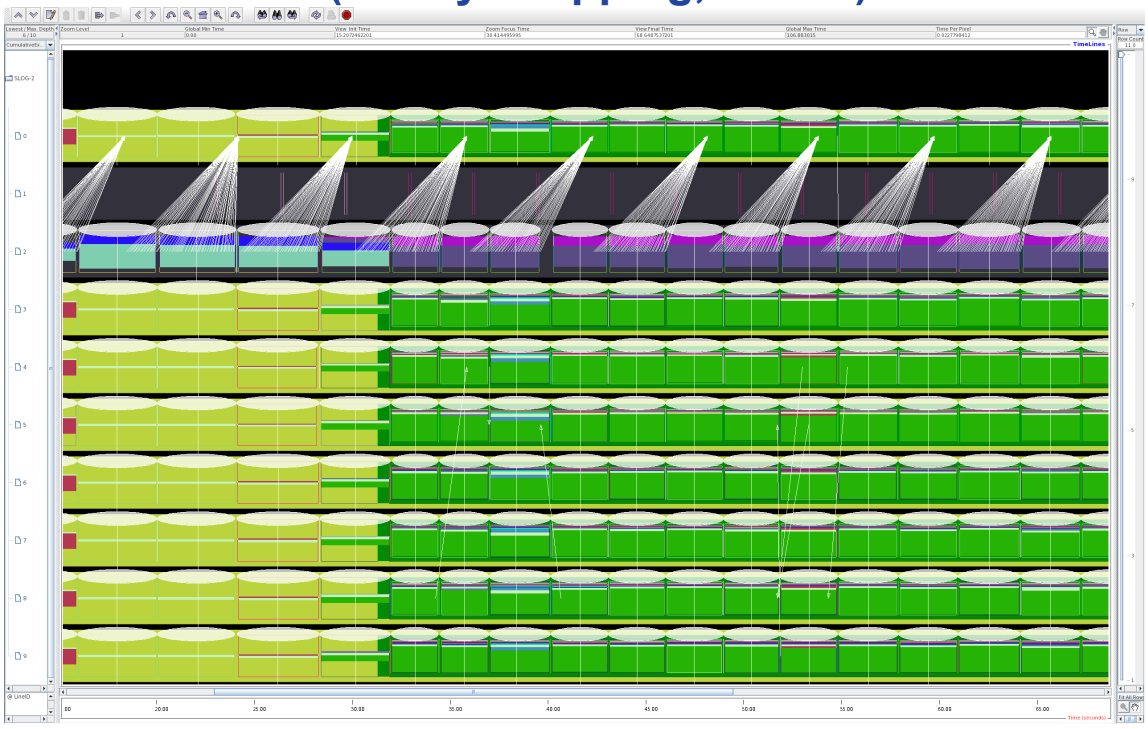
Name Δ	Exclusive TIME	Inclusive TIME	Calls	Child Calls
.TAU application	1,853.669	1,990.913	1	822
cuCtxAttach (DriverAPI)	1.027	1.027	466	233
cuCtxCreate_v2 (DriverAPI)	76.783	76.783	2	1
cuCtxDetach (DriverAPI)	56.477	56.477	458	229
cuCtxSynchronize (DriverAPI)	0.021	0.021	4	2
cuDeviceComputeCapability (DriverAPI)	0.009	0.009	4	2
cuDeviceGet (DriverAPI)	0.016	0.016	6	3
cuDeviceGetAttribute (DriverAPI)	0.478	0.478	104	52
cuDeviceGetCount (DriverAPI)	0.03	0.03	2	1
cuDeviceGetName (DriverAPI)	0.052	0.052	4	2
cuDeviceGetProperties (DriverAPI)	0.16	0.16	4	2
cuDeviceTotalMem_v2 (DriverAPI)	0.058	0.058	4	2
cuFuncSetBlockShape (DriverAPI)	0.142	0.142	62	31
cuFuncSetSharedSize (DriverAPI)	0.153	0.153	62	31
cuGetExportTable (DriverAPI)	0.012	0.012	4	2
cuLaunchGridAsync (DriverAPI)	0.279	0.279	62	31
cuMemAlloc_v2 (DriverAPI)	0.12	0.12	6	3
cuMemFree_v2 (DriverAPI)	0.159	0.159	6	3
cuMemcpyDtoH_v2 (DriverAPI)	0.114	0.114	2	1
cuMemcpyHtoD_v2 (DriverAPI)	0.215	0.215	4	2
cuModuleGetFunction (DriverAPI)	0.007	0.007	2	1
cuModuleLoadFatBinary (DriverAPI)	0.114	0.114	2	1
cuModuleUnload (DriverAPI)	0.018	0.018	2	1
cuParamSetSize (DriverAPI)	0.138	0.138	62	31
cuParamSetv (DriverAPI)	0.662	0.662	310	155

Name Δ	Total	NumSamp...	MaxValue	MinValue	MeanValue	Std. Dev.
.TAU application						
cuMemcpyDtoH_v2 (DriverAPI)						
Bytes copied	51,200	1	51,200	51,200	51,200	0
cuMemcpyHtoD_v2 (DriverAPI)						
Bytes copied	76,800	2	51,200	25,600	38,400	12,800
Bytes copied	256,000	6	51,200	25,600	42,666.667	12,067.956
cuMemcpyDtoH_v2 (DriverAPI)						
cuMemcpyHtoD_v2 (DriverAPI)						

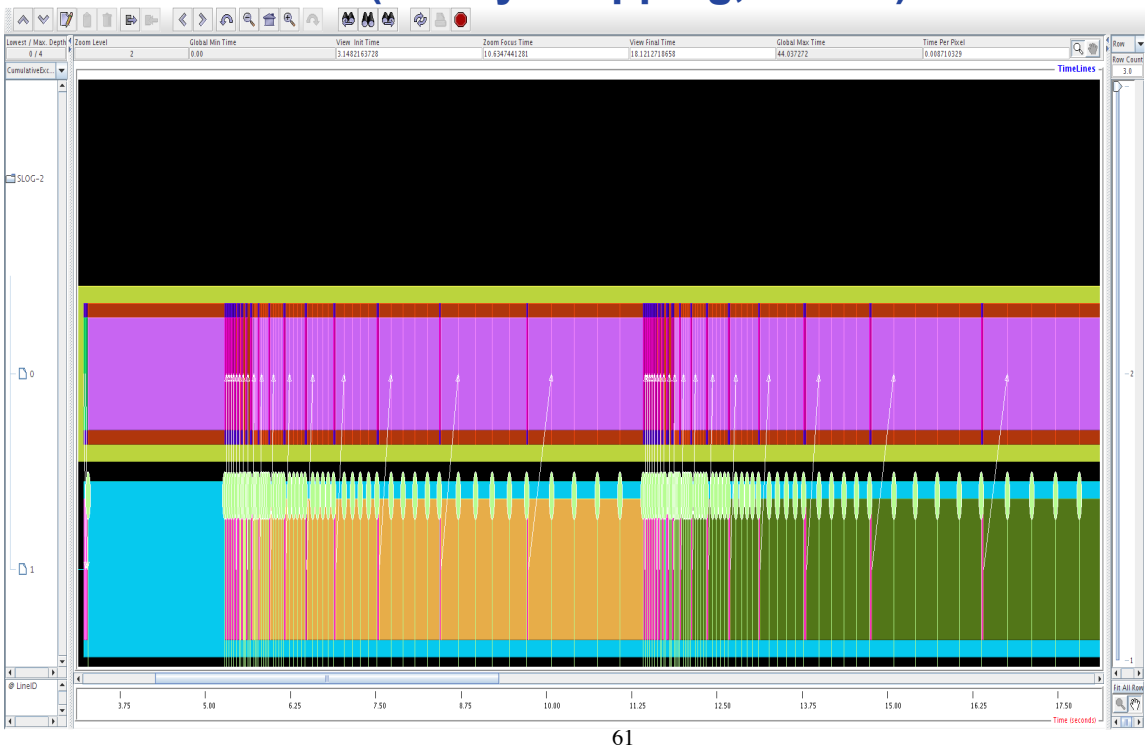
Linpack Trace (library wrapping, CUPTI)



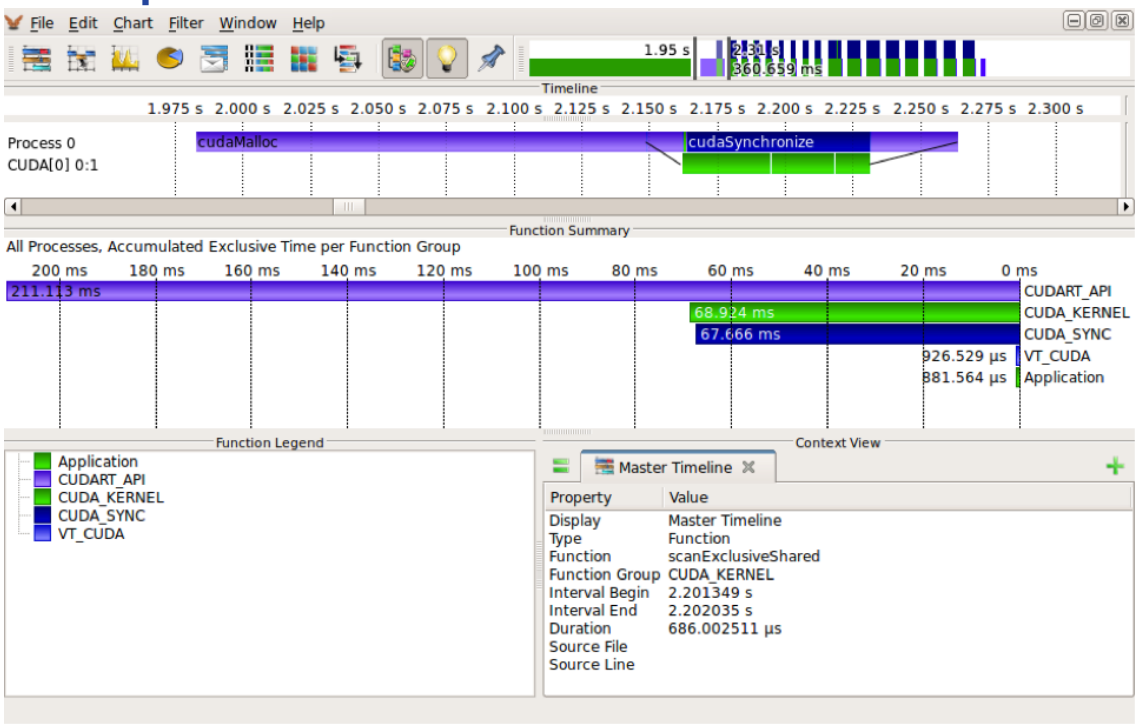
NAMD Trace (library wrapping, CUPTI)



SGEMM Trace (library wrapping, CUPTI)



VampirTrace CUDA with CUDA Events



Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science contracts
  - LLNL-LANL-SNL ASC/NNSA Level 3 contract
  - LLNL ParaTools contracts
- Department of Defense (DoD)
  - PETTT, HPTi
- National Science Foundation (NSF)
  - POINT
- University of Oregon
  - A. Malony, W. Spear, S. Biersdorff, A. Nataraj
- University of Tennessee, Knoxville
  - Dr. David Cronk and Dr. Shirley Moore
- T.U. Dresden, GWT
  - Dr. Wolfgang Nagel and Dr. Holger Brunst
- Research Centre Juelich
  - Dr. Bernd Mohr, Dr. Felix Wolf